Imperial College London

MEng Individual Project

Imperial College London

Department of Computing

Cryptocurrency Statistical Arbitrage on Decentralised Exchanges

Author:
Devam Savjani

Supervisor:
Dr. Thomas E. Lancaster

Second Marker:
Mr. Ivan Procaccini

June 17, 2023

Abstract

SOME ABSTRACT

Acknowledgements

SOME ACKNOWLEDGEMENTS

Contents

Chapter 1

Introduction

1.1 Motivation

Since the introduction of Bitcoin, a peer-to-peer payment network and cryptocurrency, there have been countless new cryptocurrencies that are used and traded. Along with this, their high volatility has piqued a lot of retail investor's interest with some investors gaining a high return of interest and others losing a lot of money [1]. In addition to this, larger investment institutions have also sought to gain profits from this new type of tradable asset [2]. This has lead to more sophisticated forms of cryptocurrency trading.

However, trading cryptocurrencies was initially very difficult for people without technical know-how, and the first recorded transaction was on 12^{th} October 2009 via a paypal transaction [3]. Since then, other cryptocurrency exchanges have emerged, many of which are centralized, which provide a more traditional trading terminal and support for investors with little technical know-how, and others are decentralised, which operate on blockchain networks and allow users to directly with each other using smart contracts. Centralized exchanges are predominant due to their easy-to use and familiar interface for traders. We can see in Figure 1.1 the proportion of trades on decentralised exchanges compares to the number of trades on centralized exchanges, we can also see that the volume traded in decentralised exchanges (DEXes) have been near 0% until the summer of 2020 and has been increasing since.

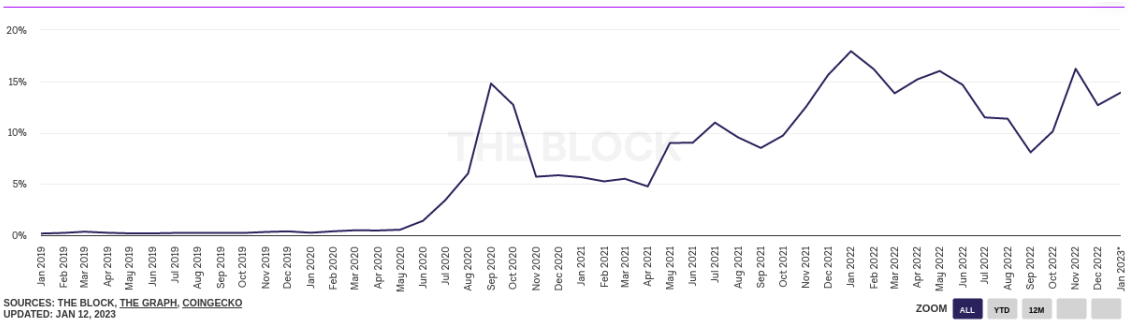

Figure 1.1: DEX to CEX Spot Trade Volume [4]

This poses the question that about which trading strategies can exploit arbitrage opportunities on decentralised exchanges. There has been some research on this topic, mainly focussing on triangular and cyclic arbitrage on DEXes such as Uniswap and SushiSwap, however there has been no research into analysing the performance of statistical arbitrage methods on decentralised exchanges.

1.2 Ethical Issues

The ethics of cryptocurrencies are widely debated for reasons such as anonymity, leading it to be the choice of currency used by criminals and illegal institutions, volatility and lack of regulation. The high volatility makes cryptocurrencies and decentralised finance very risky for retail investors that don't have the technical or financial know-how making investing in cryptocurrencies.

Another aspect of cryptocurrencies that has raised ethical questions is the energy consumption and carbon dioxide emission from the mining of cryptocurrencies. Formal research about this has also been completed and found that 'approximately 69 million metric tons of CO2 (Carbon dioxide) emission as a result of bitcoin mining' [5]. Thus, this is an ethical concern that I have thought about when designing the strategies so that the number of transactions that don't result in a profit, i.e. do not add value to the project, is limited.

In addition to the concerns above, although this project aims to find riskless profits, *'free lunches'*, it is not, in any form, of financial advice, and those who use the research or software that used in the development and research process to attempt to get favourable results, are liable for the losses or gains.

1.3 Contributions - TODO

Chapter 2

Background

2.1 Cryptocurrencies

Before delving into the financial side of the project, it is important to understand the underlying assets and the technology that drives them.

2.1.1 Blockchain

The building blocks of cryptocurrencies come from blockchain technology. Blockchain is a distributed ledger that stores data, in blocks, in a chain, comprising the data itself as well as a full transaction history [6]. They can be thought of as a State Transition System where the state is the store of data, i.e. the owners of each token, and the state-transition function is a function of state and a transaction [7]. The transition function defines how the transaction, T, should affect the state, σ.

$$\sigma' = transition\_func(\sigma, T)$$

Thus the state, σ, and transition function $transition\_func$ are defined by the blockchain implementation, which tends to be more complex.

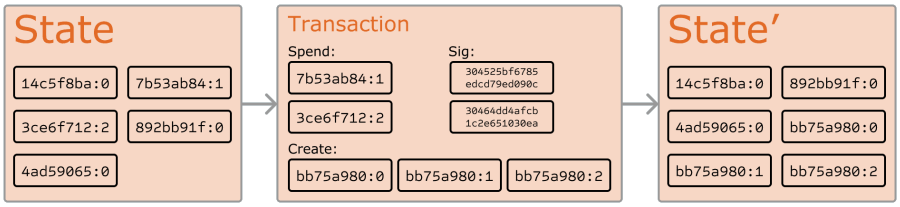

Figure 2.1: State Transition Diagram [7]

In reality, a collection of transactions are batched together into a block which is then commited to the network. The components of a block can be seen below.

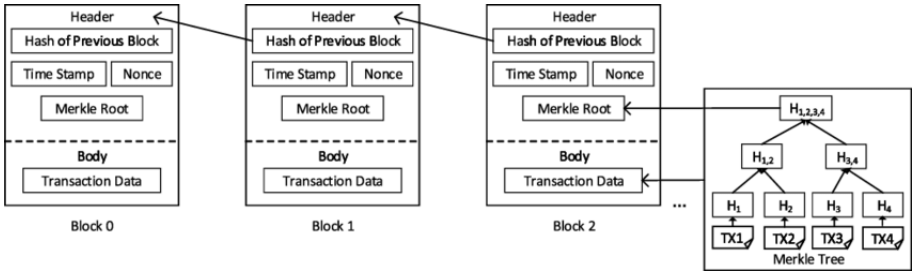

Figure 2.2: Blockchain Diagram [8]

A block can two significant components, the header and the body. The body of the block contains the list of transactions that are to be applied. Whereas, the components of the

header may depend on the blockchain's implementation however they typically have the following key components.

- **Block number** - A unique identifier assigned to each block in the chain. It serves as a chronological ordering mechanism, indicating the position of the block within the blockchain.
- **Timestamp** - Timestamp of the block's creation time.
- **Merkle Root** - A Merkle root is also stored in each block to validate transactions in the body of the block efficiently, in terms of storage and searching [9]. A Merkle tree is a tree of hashes where each leaf node is its data hash and its parent node is the hash of their children's hashes. In storing the Merkle root, we do not need to directly store each transaction in each block, and also allows a quick search for any malicious alterations in differing blocks.
- **Hash of Previous Block** - Acts as a pointer to the predecessing block. Hence, the term "Blockchain" is coined as one would be able to track back to the first block from the latest block in a chain like manner.

In addition to these components, a major feature of blockchain technology is its property of being decentralised. The state of the blockchain is managed by a distributed network of nodes that are maintained by voluntary node operators. Node operators maintain the blockchain by synchorizing with the network, validating, propogating transactions and blocks and finally participating in consensus mechanisms to ensure all of the nodes agree on the current state before updating the blockchain. For this blockchain networks are typically built on a peer-to-peer (P2P) architecture. Each participant in the network, or node, has an equal status and communicates directly with other nodes. Transactions and data are shared among participants without the need for intermediaries or central servers. When transitioning to the next state, the node broadcasts the transaction, T, to the blockchain network and each node applies the transition function to obtain the new and same state, $\sigma' = transition\_func(\sigma, T)$. All nodes arrive to the same state as the transition function is deterministic. Once each node has applied the transition function, the network uses a consensus mechanism to the globally agreed σ' and the transactions become actualised.

Consensus mechanisms are used to achieve agreement among participants on the validity of transactions and the order in which they are added to the blockchain. Examples of consensus mechanisms include Proof of Work (PoW), Proof of Stake (PoS), and Delegated Proof of Stake (DPoS), however current blockchains predominantly use Proof of Work or Proof of Stake [10]. In addition to reaching consensus, the consensus algorithms also determine which node should be the create of a new block, which node should be granted a monetary reward.

The Proof of Work (PoW) consensus algorithm was first used in Bitcoin and is where the nodes compete to solve complex mathematical problems to verify that the transactions from a new block are valid in order to add them to the blockchain. These nodes are called miners. Once the first miner obtains the solution of the mathematical puzzle, it broadcasts the new block on the network so that the other nodes verify the solution and update their local replica. Consensus is then reaches when 51% of the nodes agree on the new state of the blockchain. The main drawback of this algorithm is that it requires miners to perform extensive computational calculations repeatedly until they find the solution to the mathematical problem, hence once the solution has been found by a miner, after expensive work, the solution is sent along in the broadcast to the other nodes to easily and cheaply verify the solution of the problem.

To address this issue, some blockchains have opted to use the Proof of Stake (PoS) consen-

sus algorithm as it is more energy efficient and scalable. Unlike Proof of Work, Proof of Stake use validators to create new blocks. Validators stake an amount of cryptocurrency as collatoral and the creator of a block is chosen at random from the validators, with the probability of selection based on the amount they have staked. Due to the advantages that PoS has over PoW, Ethereum has transitioned from PoW to PoS in Ethereum 2.0.

2.1.2 Ethereum

One of them was proposed by Vitalik Buterin, the co-founder of Ethereum, in a whitepaper that proposed the idea of using smart contracts to create financial products and services that could operate independently of traditional financial institutions, hence decentralised finance was birthed [11].

Smart Contracts

Smart contracts are programs that are self-executing contracts between buyers and sellers that deploy on a blockchain. The programs are reactive in the sense that they are only executed on the blockchain providing that certain conditions are met [12]. These conditions are set in the lines of code in the smart contract. To posses the reactive property smart contracts are stored and executed on every participating node on the network as part of the state σ. They are also immutable and decentralised which make smart contract advantageous as they can be used to facilitate, verify, and enforce the negotiation or performance of a contract [13, 14].

Ethereum's Architecture

Ethereum's architecture is similar to bitcoin's but has a few differences, one of which is rather than managing a distributed ledger, it uses a distributed state machine. The Ethereum Virtual Machine (EVM) defines the rules of changing states from block to block. Each node on the Ethereum blockchain contains an immutable instance of the EVM [15].

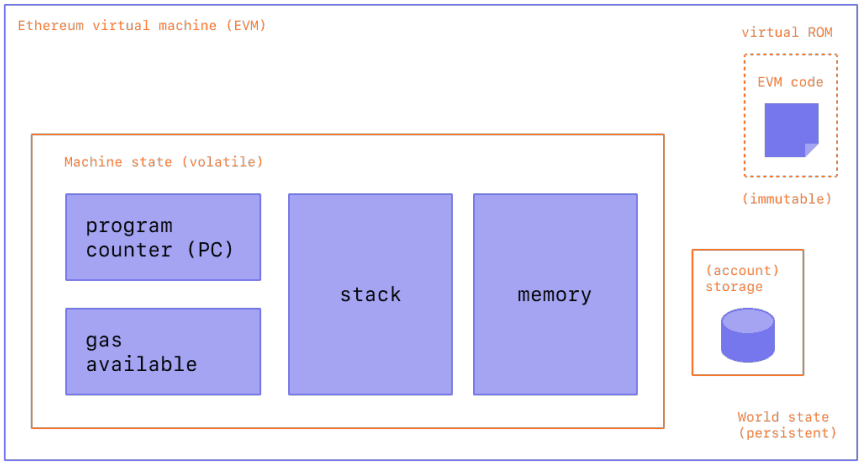

Figure 2.3: EVM components [15]

Ethereum's state σ consists of the states of all accounts where the state of an account is represented by four components: balance of ETH (the native cryptocurrency on Ethereum), a nonce, hash of the smart contract code (if account is a smart contract) and the Merkle root of smart contract storage which maintains the contracts variables (again, if account is a smart contract) [16]. A nonce, 'Number only used once', is a number that is added to a hashed block to make the transaction more secure. It is randomly generated and used to validate a transaction. A miner first guesses a nonce and appends the guess to the hash

of the current header. The miner then rehashes the value and compares this to the target hash. If the guess was correct, the miner is granted the block [17].

Ethereum supports two types of transactions: contract creation and message calls. Message calls are used to transfer ETH from account to account and also for invoking smart contract. The process of how transactions are validated is below:

1. Validate the parent block
2. Validate that the current timestamp is greater than the previous timestamp
3. Check that the Ethereum concepts are valid
4. Perform Proof of Stake on the block
5. Check for errors and gas
6. Validate the final state

Gas Fees

Due to the capability of expressing loops in EVM bytecode, there is a possibility that smart contract execution could continue indefinitely. This presents a challenge as invoking a contract with an infinite loop would cause all Ethereum network nodes to become stuck executing it, resulting in a denial of service. To mitigate this issue, Ethereum has implemented a pricing mechanism. Every computation performed by a smart contract requires the payment of a fee called gas, which is denominated in ether.

To measure how much computational effort is required to execute operations on the Ethereum network, gas is used [18]. Every block has a base fee, derived from the demand for the block space, which is burnt. Therefore, users of the network are expected to set a tip (priority fee) to reimburse miners for adding their transaction in blocks, thus the higher the tip, the greater the incentive for miners to validate the transaction. Using gas means that the Ethereum network is tolerant to spam and also has a maximum gas fee to make Ethereum tolerant to malicious code that would be used to waste resources.

2.1.3 Decentralised Finance

One of the applications of Ethereum and smart contracts is Decentralised Exchanges (DEXes). Before delving into DEXes it is important to understand centralized exchanges.

Centralized Exchanges

Centralized exchanges(CEXes) allow agents to discover and trade assets. CEXes facilitate trading between buyers and sellers by proving an online platform that manages and maintains an order book. An order book aggregates buy and sell orders and execute matching buy and sell orders. The order book and transactions are typically managed on a central database. When trading, exchanges charge trading fees for the maker and the taker to operate the exchange and do not charge any gas fees as there is no interaction with the blockchain.

Decentralized Exchanges

In contrast, DEXes utilize blockchain technology and smart contracts to execute trades thus providing a high level of determinism, by nature of the technology. These trades are executed on the blockchain via smart contracts and on-chain transactions. There are two types of DEX: order book DEXes and Automated Market Makers (AMMs). An order book DEX is less common and is akin to a CEX, an order book is stored on the blockchain rather than on a central database. This means each order placed requires the order book

to be posted on the blockchain at each transaction. Automated Market Makers are more common and provide instant liquidity by using liquidity pools so that users can swap their tokens for a price that is determined by the portions within the liquidity pool [19]. DEXes have multiple pros including lower transaction fees, privacy, diversity and trustless transactions but they also have their drawbacks such as scalability and poor liquidity a lot of the DEXes are quite new [20].

2.2 Arbitrage

Arbitrage is the process in which a trader simultaneously buys and sells an asset to take advantage of a market inefficiency [21]. Arbitrage is also possible in other types of securities by finding price inefficiencies in the prices of options, forward contracts and other exotics.

Sources have shown that the word "*Arbitrage*" has been used as early as the Renaissance era when surviving documents showed a large number of bills being exchanged [22]. There has also been some evidence to suggest that arbitrage was used as early as the Greek and Roman eras. Early forms of arbitrage would likely have been purchasing a commodity then transporting them to a foreign land and selling them at a higher price. This type of arbitrage is called commodity arbitrage and is still applicable today. With the example above, transporting the goods takes a significant amount of to the merchant, or trader, which could cause variations in the price, however, in the modern day this has been reduced and with electronic exchanges, this time to buy and sell is very small. This means inefficiencies in the market, where a trader can profit purely by buying and selling, should not exist. This is called the "Law of One Price". The "Law of One Price" states that every identical commodity or asset should have the same price regardless of exchange or location, given there are no transaction costs, no transportation costs, no legal restrictions, the exchange rates are the same and no market manipulation occurs [23]. This is because if this were not the case, an arbitrage opportunity would arise and someone would take advantage of the scenario causing the prices on both markets to converge due to the market forces. In the real world arbitrage opportunities are tremendously common, thus allowing a risk-free investment [24, 25].

There are countless types of arbitrage such as spatial arbitrage, which profits off of different prices on exchanges in different locations, temporal arbitrage, which takes advantage of price differences at different times, risk arbitrage, which profits from perceived discrepancies in their risk-return profiles and finally market arbitrage which takes advantages of different prices on different exchanges/markets. Statistical methods include pairs trading, which involves buying and selling assets that are believed to be mispriced relative to one another, momentum trading, which identifies if assets have a strong momentum (either up or down) and profiting off of that, and finally, algorithmic trading which uses algorithms to analyze data and trades based on statistical analysis. This project shows how these opportunities can be exploited both in a pure manner as well as using statistical methods.

2.3 Pure Arbitrage Techniques

Research into cryptocurrency arbitrage is still in its infancy and previous research has mainly focussed on the economics of cryptocurrencies, i.e. miner/trader behaviour and influence of cryptocurrency trading [26, 27, 28, 29, 30, 31, 32]. Furthermore, there has been very limited research comparing statistical strategies and pure methods of arbitrage of cryptocurrencies. However, it is still important to understand the basics of pure arbitrage, this form of arbitrage takes advantage of price discrepancies between three or more different currencies in the foreign exchange market. It involves executing a series of trades to profit

from the imbalance in exchange rates between the currencies involved. An example of this can be seen in Figure 2.4, given the exchange rates are \$1 = €0.85, €1 = £0.75, £1 = \$1.20, we can make a series of exchanges (trades) such that by starting with \$100, the result of this cyclic arbitrage I am left with \$130.72, hence a \$30.72 risk-free profit. Although the example uses fiat currencies (traditional currencies that are issued by governments, such as the US Dollar (USD) and the Great British Pound Sterling (GBP)), the same also holds for cryptocurrencies. Additional research into the cryptocurrency arbitrage can be found in Appendix A.1.

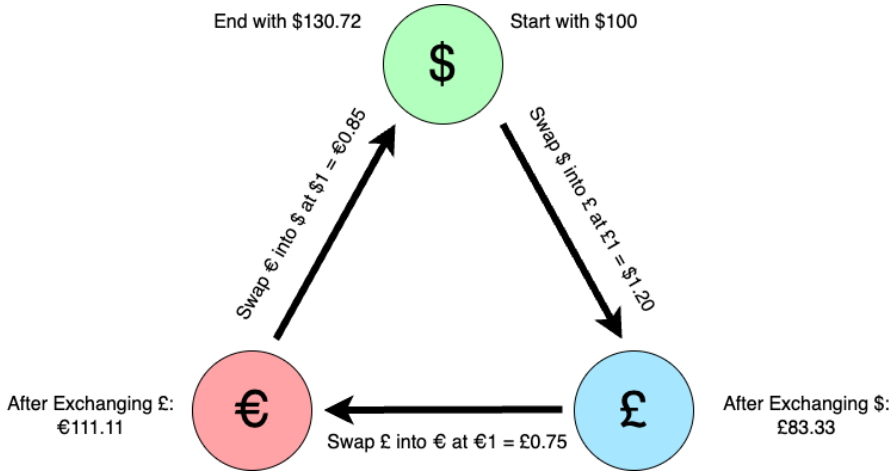

Figure 2.4: Triangular Arbitrage Diagram

2.4 Statistical Arbitrage Techniques

As mentioned previously mentioned and the subject of the project is to optimize statistical arbitrage methods to be able to compete with a purer form of arbitrage, i.e. cyclic arbitrage. As previously mentioned there are many methods of stat arb, pairs trading, momentum trading and algorithmic trading. Within these methods there are countless strategies to adopt and profit from, thus to limit the scope, this project I will be investigating strategies within pair trading. Research within Pair trading has been vast with many streams of approaches emerging; distance approach, cointegration approach, time-series approach, stochastic approach and some others, including using machine learning [33]. However, for this project, we will only look at cointegration/co-correlation approaches.

2.4.1 Mean Reversion

The cointegration approach follows three key steps. The first is the selection of pairs based on similarity measures, the next is assessing the tradability and finally, thresholds are set for trading. The spread is defined as

$$\varepsilon_{ij,t} = P_{i,t} + \gamma P_{j,t}$$

where $P_{i,t}$ and $P_{j,t}$ denote the $I(1)$ non-stationary price processes of the assets i and j, γ is the cointegration coefficient, also referred to in literature as the hedge ratio. $\varepsilon_{ij,t}$ is the linear combination of the non-stationary prices and is $I(0)$ stationary and hence mean-reverting, note that stationary processes are those of which have a constant mean. Rad's implementation of this approach on stocks results in a 0.83% return before considering transaction costs [34]. Another paper, [35], looked into setting the thresholds and setting a minimum profit, MP_{ij,t_c}:

$$MP_{ij,t_c} = \frac{n(\varepsilon_{ij,t_0} - \varepsilon_{ij,t_c})}{\mid \gamma \mid}$$

Where t_0 and t_c are the opening and closing times, n is the volume longed of asset j. Additional research on the optimal portfolio designed for mean reversion can be found in Appendix A.2.

2.4.2 Statistical Arbitrage using the Kalman Filter

Another method that is used in statistical arbitrage is using the Kalman Filter. Recall the equation for spread:

$$\varepsilon_{ij,t} = P_{i,t} + \gamma P_{j,t}$$

The Kalman filter is a recursive algorithm for estimating the state of noisy data and that needs to be filtered to be able to estimate the state of a system based on a sequence of observations, taking into account both current measurements and the system dynamics [36]. This makes it very useful to estimate the hedge ratio γ. Initially, a book by Vidyamurthy discusses best practices for choosing cointegrated equities and found that the Kalman filter was found optimal when the state-space and observation equations are linear and the noise is Gaussian [37]. Since then there have been many extensions of the filter such as the Extended Kalman Filter (EKF) and Unscented KF aimed to handle when the state-space and observation equations are non-linear and the noise is not Gaussian.

The Kalman Filter works in 2 phases, prediction and update. The prediction phase is as follows

$$\hat{\mathbf{x}}_k = \mathbf{F}_k\hat{\mathbf{x}}_{k-1} + \mathbf{B}_k\vec{\mathbf{u}}_k + \mathbf{w}_k$$
$$\mathbf{P}_k = \mathbf{F}_k\mathbf{P}_{k-1}\mathbf{F}_k^T + \mathbf{Q}_k$$

Where $\hat{\mathbf{x}}_k$ is the new best estimate (prediction) that is derived from $\hat{\mathbf{x}}_{k-1}$, the previous estimate and the prediction function $\mathbf{F}_k$. $\vec{\mathbf{u}}_k$ is the correction term, called the control vector, that is used when it is known that there are external influences in combination with $\mathbf{B}_k$ which is called the control matrix. In addition to this, the new uncertainty (covariance matrix), $\mathbf{P}_k$, is calculated using the previous uncertainty and additional uncertainty from the environment, $\mathbf{Q}_k$, $\mathbf{w}_k$ is called the state noise. The update is as follows

$$\hat{\mathbf{x}}'_k = \hat{\mathbf{x}}_k + \mathbf{K}'(\vec{\mathbf{z}}_k - \mathbf{H}_k\hat{\mathbf{x}}_k)$$

$$\mathbf{P}'_k = \mathbf{P}_k - \mathbf{K}'\mathbf{H}_k\mathbf{P}_k$$
$$\mathbf{K}' = \mathbf{P}_k\mathbf{H}_k^T(\mathbf{H}_k\mathbf{P}_k\mathbf{H}_k^T + \mathbf{R}_k)^{-1}$$

Where $\mathbf{K}'$ is defined as the Kalman gain, $\mathbf{H}_k$ is the measurement matrix, $\vec{\mathbf{z}}_k$ is mean of the observed values, which is also calculated by $\vec{\mathbf{z}}_k = \mathbf{H}_k\hat{\mathbf{x}}_k + \mathbf{v}_k$ where $\mathbf{v}_k$ is the measurement noise, and $\mathbf{R}_k$ is the covariance of the uncertainty of the observed values [38].

A paper that investigated the use of the Kalman Filter on ETFs found that the strategy it employed worked well for in-sample data points and worse, but still profitable, results for out-of-sample data [39]. The paper adapted the Kalman Filter to be able to use it for pairs trading to the following:

$$\mathbf{y}_t = \mathbf{x}_t\beta_t + \epsilon_t$$
$$\beta_t = \mathbf{I}\beta_{t-1} + \omega_t$$

Then calculating the Kalman Gain:

$$\text{Kalman Gain} = \frac{\text{Error in the estimate}}{\text{Error in the estimate} + \text{Error in the measurement}}$$

Then to calculate the estimate:

$$\text{Estimate}_t = \text{Estimate}_{t-1} + \text{Kalman Gain} \times (\text{Measurement - Estimate}_{t-1})$$

And finally, calculating the new error:

$$E_{\text{estimate}_t} = \frac{E_{\text{measurement}} \times E_{\text{estimate}_{t-1}}}{E_{\text{measurement}} + E_{\text{estimate}_{t-1}}}$$

$$E_{\text{estimate}_t} = E_{\text{estimate}_{t-1}} \times (1 - \text{Kalman Gain})$$

The author states "pairs trading strategies have gained widespread acceptance thus making profitability much more elusive" to justify the disappointing results, however, the author fails to find evidence or provide sufficient evidence to justify the claim [39].

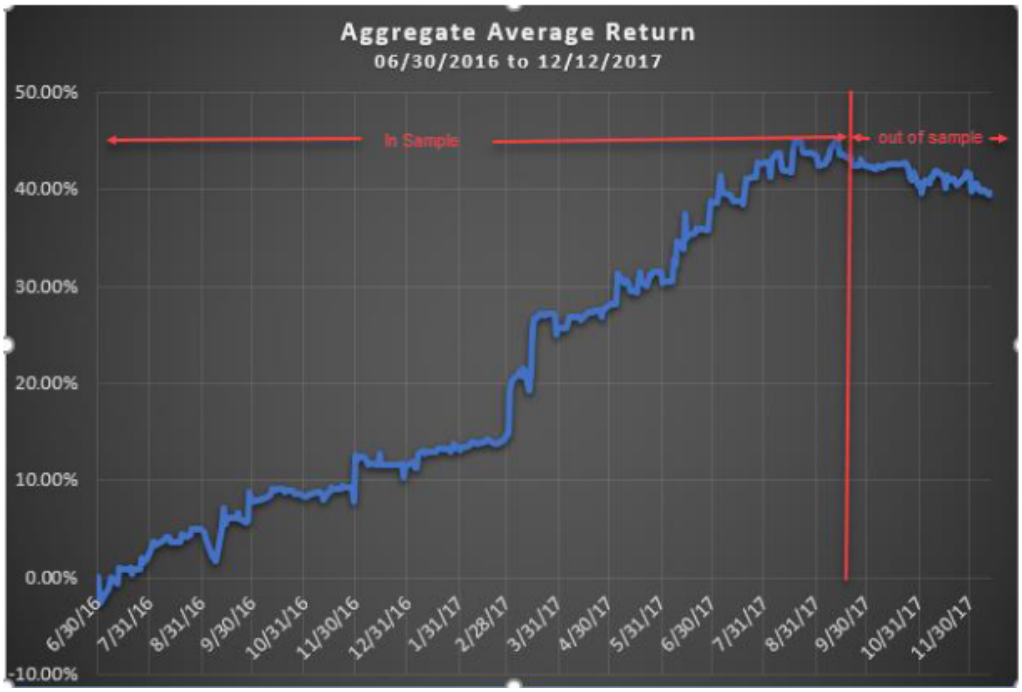

Figure 2.5: Aggregate average return of using the Kalman filter for pairs trading on ETFs [39]

Another paper used the combination of the Kalman Filter and Machine Learning, more specifically Extreme Learning Machine and Support Vector Regression (SVR) to build a statistical arbitrage strategy on the Brazilian Stock Exchange [40]. The strategies can simply be explained as using SVR and ELM to forecast returns and using the Kalman Filter to improve the forecast.
The paper also compares methods, such as LASSO, BMA, and GRR, to benchmark the performance of the Kalman Filter. The research found that using simply ELM and SVR forecasts results in a return of 20.19% and 21.32% respectively for out-of-sample data points and using a combination with the Kalman Filter gives a return of 26.13% for out-of-sample data points. The full results can be seen below in Figure 2.7. In addition to this, it can be seen that the volatility of the return also decreases which is ideal for investment managers.

Other papers/articles such as [41, 42, 43] have designed, compared and analysed other statistical arbitrage techniques using Machine Learning algorithms and revealed that some algorithms are profitable. The majority of research on machine learning trading strategies has been on assets such as stocks on centralized exchanges. The little research that has been done on statistical arbitrage on cryptocurrencies has all been on analysing arbitrage on centralized exchanges and not decentralised exchanges. One of the research projects that analysed machine learning methods of statistical arbitrage on cryptocurrencies on a centralized exchange, compared a logistic regression approach with a random forest approach [43].

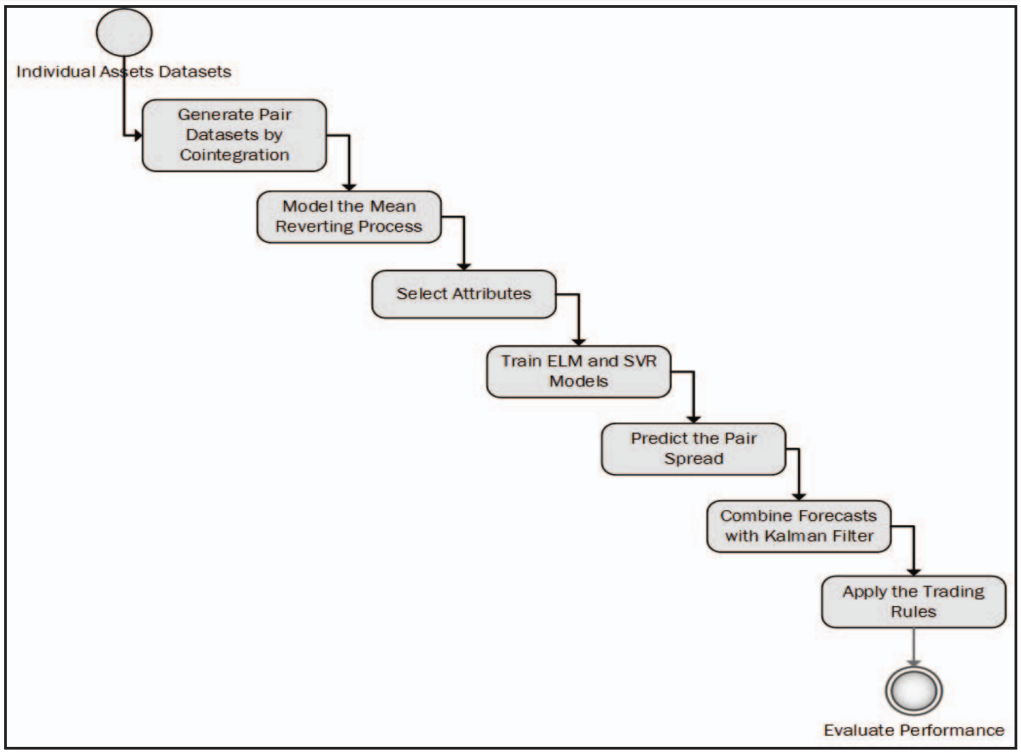

Figure 2.6: Visualisation of the trading strategy used in [40]

TABLE IV. ECONOMETRIC PERFORMANCE – ELM AND SVR – IN-SAMPLE

MODEL	MAX. DD	SHARPE	VOLATILITY	RETURN
ELM	-2.31%	1.80	**3.05%**	5.83%
SVR	**-2.18%**	1.73	3.20%	5.39%

TABLE V. ECONOMETRIC PERFORMANCE ELM AND SVR – OUT-OF-SAMPLE

MODEL	MAX. DD	SHARPE	VOLATILITY	RETURN
ELM	-2.80%	3.83	5.64%	20.18%
SVR	-2.72%	4.31	5.29%	21.32%

TABLE VI. ECONOMETRIC PERFORMANCE – COMBINATION MODELS– IN-SAMPLE

MODEL	MAX. DD	SHARPE	VOLATILITY	RETURN
BMA	-2.24%	1.38	**3.05%**	4.94%
GRR	-2.34%	**2.07**	3.20%	6.37%
KALMAN	-2.30%	1.97	3.57%	**6.89%**
LASSO	-2.43%	2.06	3.39%	6.30%

TABLE VII. ECONOMETRIC PERFORMANCE – COMBINATION MODELS – OUT-OF-SAMPLE

MODEL	MAX. DD	SHARPE	VOLATILITY	RETURN
BMA	-2.97%	3.83	**5.12%**	19.33%
GRR	**-2.53%**	4.76	5.49%	23.69%
KALMAN	-2.64%	**5.29**	5.17%	**26.13%**
LASSO	-2.64%	4.76	5.49%	23.79%

Figure 2.7: Econometric results [40]

2.4.3 Analysis on Cryptocurrency Arbitrage on Centralized Exchanges

Although the research in the papers previously mentioned does not investigate the cointegration approach on cryptocurrencies, the takeaways are the mathematical fundamentals that are used in statistical arbitrage. Kristoufek and Bouri researched the sources of stat. arb. of bitcoin in multiple centralized exchanges. The Grey correlation is built on top of the Grey system theory [44], and can capture non-linear correlations without assuming a Gaussian distribution, thus using the Grey correlation provides a more robust metric to understand correlations between both series. The Grey correlation $\gamma(X_0, X_i)$ is defined with two steps:

1. $\gamma(x_0(k), x_i(k)) = \frac{min_i min_k |x_0(k) - x_i(k)| + \varepsilon max_i max_k |x_0(k) - x_i(k)|}{|x_0(k) - x_i(k)| + \varepsilon max_i max_k |x_0(k) - x_i(k)|}$

2. $\gamma(X_0, X_i) = \frac{1}{n}\sum_{i=1}^{n}\gamma(x_0(k), x_i(k))$

With $\varepsilon \in [0, 1]$, the standard is set to $\varepsilon = 0.5$.

The DCC-GARCH(1,1), [45], model is also used to obtain conditional correlations for Bitcoin exchanges. The model was designed to use a combination of parameters such as the standard deviation of Bitcoin returns, traded volume, the volume of on-chain transactions, fees paid to miners, the ratio of current price and recent price history and internet hype/trends.

Upon analysis of Grey and DCC-GARCH(1,1) correlations, it is found that the DCC correlations show a little variability whereas the Grey correlations are a lot more variable ranging from 0.29 to 1. In addition, the paper then further investigates these sources and finds that these opportunities are introduced when there is a large number of inter-exchange transfer requests, i.e. the network is congested, and high price volatility. In contrast, the high volume of exchanges and on-chain activity cause the arbitrage opportunities to decrease. This paper finds and explains these sources of statistical arbitrage however does not implement or devise an algorithm that uses statistical arbitrage to generate a profit from price discrepancies of Bitcoin on different exchanges.

A paper that investigates statistical arbitrage on multiple cryptocurrencies is [46]. The authors of this paper analysed co-movements and cointegration of different cryptocurrencies on a centralized exchange using Augmented Dickey-Fuller (ADF) and Kwiatkowsky-Phillips-Schmidt-Shin (KPSS), Ljung-Box autocorrelation tests on both stationary forms ($I(0)$) and the original form ($I(1)$). The paper then develops a dynamic factor model based on the assumption that the price dynamics of cryptocurrencies are driven by Bitcoin [47], this is then evidenced by similar paths found in cryptocurrencies shown in Figure 2.8.

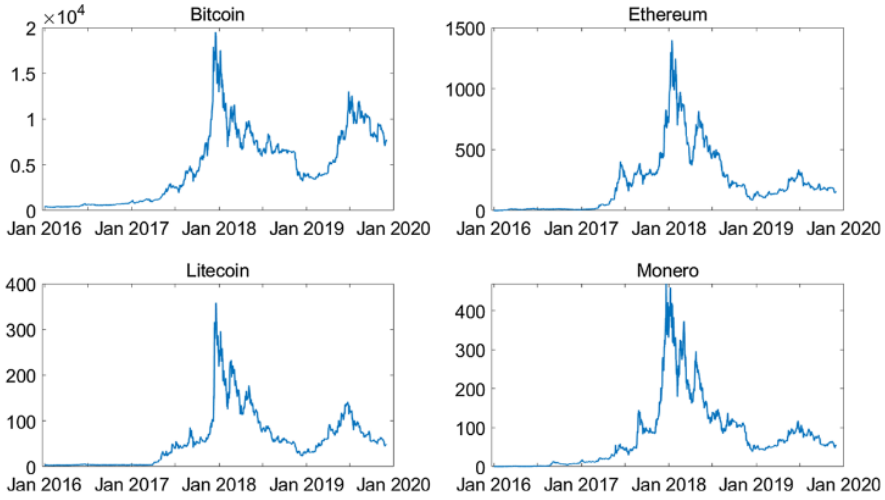

Figure 2.8: Price behaviour of Bitcoin, Ethereum, Litecoin, Monero [46]

For simplicity the authors set the number of hidden factors to 2 and upon analysis f_1 is a $I(1)$ process and the second factor f_2 is a stationary process that is independent of f_1. It is also found after overlaying f_1 with the price of Bitcoin, that the first factor strongly correlates with the price of Bitcoin.

The paper then uses this model to build an investment strategy, using forecasting using the estimated parameters:

$$\hat{p}_{i,\tau+1} = \mathbb{E}_\tau(p_{i,\tau+1}) = \hat{\alpha}_i + \hat{\beta}_{i1}\mathbb{E}_\tau(f_{1,\tau+1}) + \hat{\beta}_{i2}\mathbb{E}_\tau(f_{2,\tau+1})$$

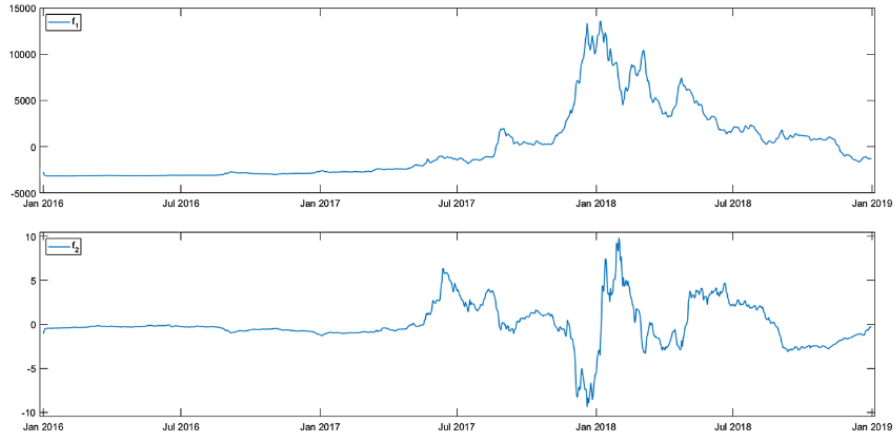

Figure 2.9: Hidden factors f_1 and f_2 from Jan 2016 to Dec 2018 [46]

Where

$$f_{1,t} = \lambda_1 f_{1,t-1} + \eta_{1,t}$$
$$f_{2,t} = \lambda_2 f_{2,t-1} + \eta_{2,t}$$

The expected gains one day ahead are given by:

$$g_{\tau+1} = \mathbb{E}_\tau[v_{\tau+1}] = \sum_{i=1}^{\lfloor I/2 \rfloor} \hat{p}_{\tau+1}^{(i)} - \sum_{i=\lceil I/2 \rceil+1}^{I} \hat{p}_{\tau+1}^{(i)}$$

Using this and a threshold which is calculated by the combination of the current price and standard deviation of the trading position value:

- if $g_{\tau+1} > v_\tau + c\sigma_\tau^v$, go long
- if $g_{\tau+1} < v_\tau - c\sigma_\tau^v$, go short
- if $v_\tau - c\sigma_\tau^v \leq g_{\tau+1} \geq v_\tau + c\sigma_\tau^v$, no trade

The researchers of the paper evaluated their trading strategy for 334 days and a moving window of 3 years, 1096 observations, every day to estimate the parameters for the dynamic factor model. We can see in Figure 2.10 that the strategy was able to consistently generate a profit even when considering transaction costs.

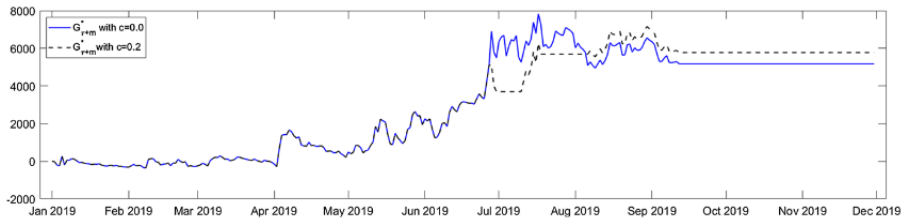

Figure 2.10: Net gains taking transaction fees into account [46]

Overall although there have been countless studies that explored statistical analysis of arbitrage opportunities on centralized exchanges, little attention has been given to investigating such opportunities on decentralized exchanges. Profitable methods like mean reversion, mean reversion utilizing the Kalman filter, and analyzing cointegrated cryptocurrencies have been identified in centralized exchange settings. Consequently, it would be intriguing to explore whether these findings hold true when applied to decentralized exchanges.

Chapter 3

Design Decisions

3.1 Mean Reversion Strategies

Considering the current state of the art and the limited exploration of statistical arbitrage in decentralized exchanges, the objective of this project is to examine the profitability of mean reversion techniques in pairs trading and determine if they offer a viable investment opportunity. Additionally, by employing various methods to estimate the hedge ratio and effectively manage risk, we can further analyze the overall performance and appeal of the trading strategy to potential traders.

In the mean reversion strategies, the hedge ratio refers to the ratio or proportion between the positions of two assets involved in a pairs trading strategy. It determines the optimal allocation of capital between the assets to minimize risk and create a market-neutral position. The hedge ratio represents the number of units or shares of one asset that should be held for each unit or share of the other asset in order to create a balanced or hedged position. It is derived through statistical techniques such as regression analysis. The methods that are explored in this paper are the use of a Constant Hedge Ratio, Sliding Window Ordinary Least Squares, Lagged Ordinary Least Squares, the unrestricted OLS model returned by the Granger Causality Test and also the Kalman Filter. These methods are selected due to their ability to test whether underlying dynamics effect the relationships between liquidity pools, hence effecting the hedge ratio. To further see the details and implementation of each strategy, refer to Section 5.

3.2 Buying and Selling

The process of buying and selling in traditional markets is relatively straightforward as brokers and exchanges play a crucial role in executing orders on behalf of individuals. However, when it comes to trading cryptocurrencies, the responsibility falls directly on the trader, such as myself. Therefore, it becomes essential to clearly define what buying and selling entail in the context of cryptocurrency trading.

3.2.1 Buying

Prices on Uniswap are represented as ratios for example 1 USDC = P WETH. With this understanding, let's delve into the process of buying one unit of USDC/WETH on Uniswap:

- Opening a Buy position:
  - Starts with P_0 WETH
  - Swaps the WETH for USDC, hence ends with 1 USDC
- Closing a Buy position:

- Starts with 1 USDC
- Swaps the USDC for WETH, hence ends with P_1 WETH

Consequently, if the price of USDC/WETH increases from the moment the buy position is opened to the time it is closed, the trader realizes a profit. On the other hand, if the price declines during this period, the trader incurs a loss.

3.2.2 Selling

Selling an asset is a more complex process compared to buying because it involves trading an asset that the trader doesn't initially possess. In traditional markets, this is facilitated by the trader borrowing the desired asset from a broker or another party. When the trader decides to close the position, they repurchase the same amount of the borrowed asset and return it to the lender, hoping that its value has declined. This borrowing and returning process is typically managed automatically by brokers and exchanges. However, in decentralized exchanges (DEXes), such mechanisms are not in place.

To simplify this process, I have opted to utilize Aave as a lending platform to borrow any required assets. It is worth noting that Aave supports only a limited range of tokens available for borrowing, namely DAI, EURS, USDC, USDT, AAVE, LINK, WBTC, and WETH. Therefore, shorting would be feasible only if I focus on liquidity pools that involve these cryptocurrencies. By leveraging Aave as a lending platform, I can access the necessary assets for shorting. The process of selling is as follows:

- Opening a Sell position:
  - Borrow 1 USDC
  - Swaps the USDC for WETH, hence ends with P_0 WETH
- Closing a Sell position:
  - Starts with P_0 WETH
  - Swaps the WETH for USDC, hence ends with $\frac{P_0}{P_1}$ USDC
  - Return the borrowed USDC, leaving $\frac{P_0}{P_1} - 1$ USDC

Consequently, if the price of USDC/WETH decreases, i.e. $P_1 < P_0$ from the moment the sell position is opened to the time it is closed, the trader realizes a profit. On the other hand, if the price increases during this period, the trader incurs a loss.

Note that this is is merely an illustrative example and does not account for any potential fees that the trader might incur.

3.3 Protocols of Interest

3.3.1 Uniswap

Overview

Uniswap is a decentralized exchange protocol built on the Ethereum blockchain. It allows users to trade ERC-20 tokens directly from their wallets without the need for intermediaries or traditional order books. Uniswap utilizes automated market-making, where liquidity providers contribute funds to liquidity pools, earning fees on trades made in the pool. The protocol employs a mathematical formula called the constant product formula to maintain balanced token ratios in the pool. When users want to make a trade, Uniswap calculates the conversion based on the pool's token ratios and executes the trade through a smart contract.

Automated Market Maker (AMM) Model and Liquidity Pools

The AMM model is a system that replaces traditional order books with liquidity pools to facilitate trading between different tokens. Automated Market Makers do this with the aid of liquidity pools. They are pools of tokens contributed by liquidity providers (LPs) to facilitate trading between different tokens within the exchange. One paper describes pools as "a smart contract that holds at least two cryptoassets and allows trading through depositing a token of one type and thereby withdrawing tokens of the other type" [48]. These liquidity pools are maintained by smart contracts on Ethereum and Liquidity Providers (LPs). Liquidity providers are individuals that voluntarily contribute an equal amount of cryptocurrencies liquidity pools, for example, in a pool for trading ETH and DAI, LPs would contribute an equal value of ETH and DAI tokens. LPs are incentivised to do provide liquidity in exchange for a share of the trading fees that occur in the liquidity pool. LPs can later withdraw their shares along with the accumulated fees. On Uniswap V3, the fee is 0.3% on Ethereum, however they can be any of 0.01%, 0.05%, 0.3%, or 1% depending on blockchain network.

Constant Product Formula

To maintain a balanced ratio between the tokens in the pool and price any swaps, the AMM model relies on a mathematical formula called the constant product formula ($xy = k$). As trades occur, the product of the token balances remains constant. When one token's value increases, its proportion in the pool decreases, ensuring an automatic adjustment in prices. When a trade is executed, the change in prices can be described in this formula $(x + \Delta x)(y + \Delta y) = k$. Hence, after rearranging: $\Delta y = \frac{k}{x+\Delta x} - y$. Under this model, the balance of the tokens in a liquidity pool can never be depleted as the token will get infinitely more expensive as the reserves approach 0.

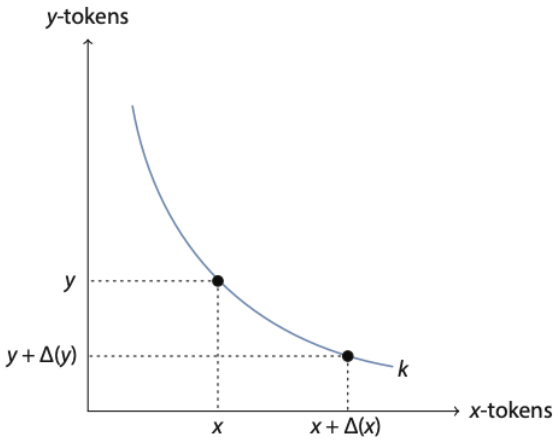

Figure 3.1: Constant Product Formula [48]

Slippage

As seen in Figure 3.2, we can see how the constant product formula is used in Uniswap. Uniswap also allow users to set slippage tolerance levels, which determine the maximum acceptable difference between the expected and executed prices. Slippage refers to the difference between the expected price of a trade and the executed price due to market volatility and liquidity conditions. Slippage happens because the constant product formula adjusts prices based on the ratio of tokens in the pool. As trades are executed, the token balances change, and the prices change accordingly. Thus, larger trades can cause more significant price impact, resulting in slippage.

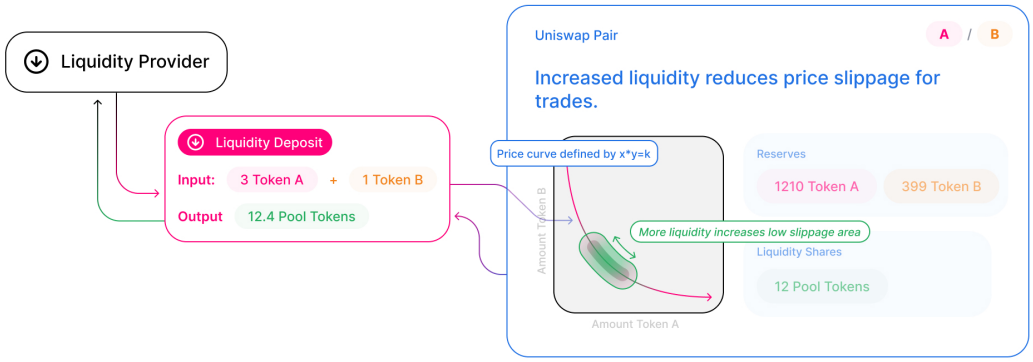

Figure 3.2: How Uniswap works [49]

3.3.2 Aave

Overview

Aave is a decentralized lending and borrowing protocol built on the Ethereum blockchain. It enables users to lend and borrow a wide range of cryptocurrencies directly, without the need for intermediaries such as banks. Aave operates through liquidity pools and smart contracts, providing a secure, transparent, and efficient platform for decentralized finance (DeFi) activities.

Users can deposit their cryptocurrency assets into Aave's liquidity pools and earn interest on their deposits. These funds contribute to the available liquidity for borrowers. On the other hand, borrowers can use their deposited assets as collateral to borrow other assets from the pool. The amount they can borrow is determined by the value of their collateral and specific borrowing parameters set by the protocol.

3.4 Fees that are incurred when trading

As previously mentioned, when engaging in trading activities, there are several fees that are deducted. These fees serve multiple purposes, including compensating the liquidity providers and addressing the potential lack of access to assets from the lender. Therefore, in this section, we will delve into the various costs associated with implementing and executing the strategy.

3.4.1 Gas Fees

Gas fees in Ethereum are a crucial component of the network's operation. They serve two main purposes: to prevent spam and denial-of-service attacks and to incentivize miners to include transactions in the blockchain. They are a measure of computational effort required to execute a transaction or perform a smart contract operation on the Ethereum network. Each operation, such as sending tokens, executing a smart contract function, or interacting with decentralized applications, consumes a certain amount of gas.

The fee is calculated by multiplying the gas price (expressed in Gwei, where 1 Gwei is equal to 0.000000001 ETH) by the gas used [18]. As mentioned in previous sections, when a transaction is initiated by a user, the user specifies the gas limit which defines the maximum amount of gas that a user is willing to pay for a transaction, if the gas limit is too low and the gas used is greater than its limit, the transaction will fail and be reverted. This means that the state changes made by the transaction will not be recorded on Ethereum, and the fees paid for that transaction will be lost.

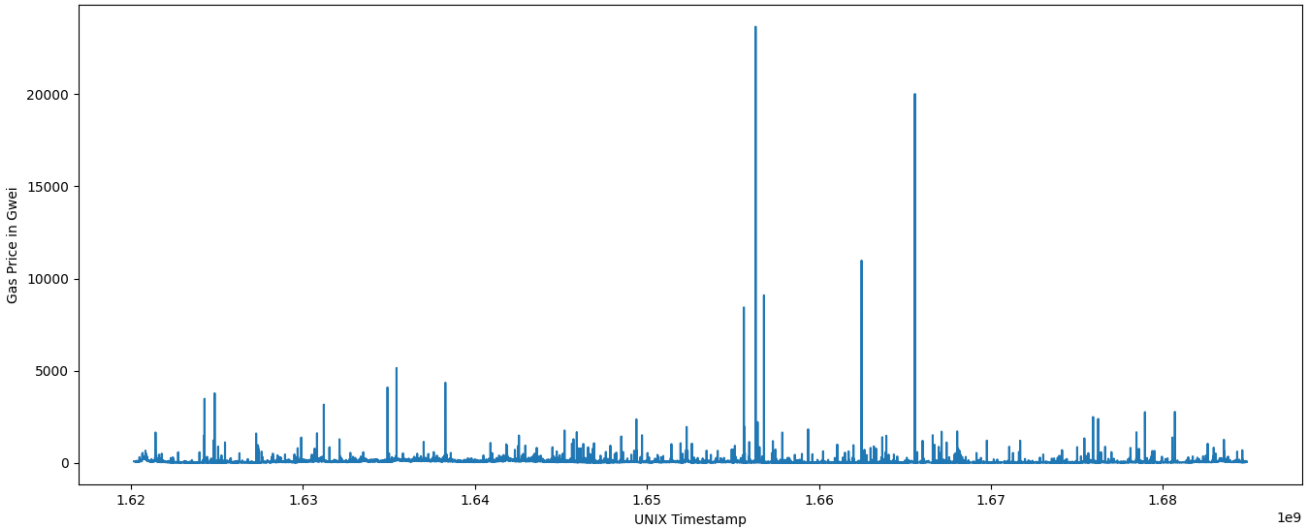

Figure 3.3: The gas price over time

Figure 3.3 the gas price history over time, showing the price surges and trophs. The reason for these fluctuations that it is determined by the market forces of supply and demand. It is the amount of Ether (ETH) that a user is willing to pay for each unit of gas consumed in a transaction. Miners have the option to prioritize transactions with higher gas prices, and thus are incentivized to include transactions with higher gas prices because they receive the gas fees as a reward for their mining efforts. Therefore, setting a higher gas price increases the chances of a transaction being executed.

The second part of the gas fee is how much gas is used, it is calculated by multiplying the gas cost of each operation in the transaction by the corresponding gas price. Each operation has a predefined gas cost associated with it, which reflects the complexity and resource requirements of that operation. When a transaction or smart contract execution is initiated, the Ethereum Virtual Machine (EVM) processes each operation and deducts the corresponding gas cost from the gas limit. In addition to this, Ethereum also has a base fee with all of their transactions thus having multiple small transaction becomes more costly than using a large combined transaction [18].

3.4.2 Uniswap Fees

In addition to transaction fees, there is another fee mechanism in place within the system when swapping on Uniswap. This fee is designed to incentivize and reward liquidity providers on the platform. When a swap occurs, a fee is deducted from the amount of tokens expected to be returned to the user.

The fee is calculated as a percentage of the swap volume, meaning that the larger the swap, the higher the fee. For example, let's consider a scenario where the exchange price is 1 TKNA = 100 TKNB and the fee is set at 0.3%. If someone exchanges 1 TKNA for TKNB, instead of receiving the full 100 TKNB based on the exchange rate, they will receive 100×(1 - 0.3%) TKNB = 99.7 TKNB. The fee reduces the total amount of tokens received in the swap.

It's important to note that the specific fee percentage may vary between different liquidity pools on Uniswap. Each liquidity pool can have its own fee tier, and the fees associated with each interested liquidity pools can be seen in Table 3.1.

Pool Address	Token0	Token1	Fee Tier as a %
0x88e6a0c2ddd26feeb64f039a2c41296fcb3f5640	USDC	WETH	0.05
0x8ad599c3a0ff1de082011efddc58f1908eb6e6d8	USDC	WETH	0.30
0x11b815efb8f581194ae79006d24e0d814b7697f6	WETH	USDT	0.05
0x4e68ccd3e89f51c3074ca5072bbac773960dfa36	WETH	USDT	0.30
0x60594a405d53811d3bc4766596efd80fd545a270	DAI	WETH	0.05
0xc2e9f25be6257c210d7adf0d4cd6e3e881ba25f8	DAI	WETH	0.30
0xe0554a476a092703abdb3ef35c80e0d76d32939f	USDC	WETH	0.01
0xc5af84701f98fa483ece78af83f11b6c38aca71d	WETH	USDT	0.1

Table 3.1: Uniswap fees for each liquidity pool of interest

3.4.3 Aave Fees & Collatoral

Aave facilitates lending and borrowing transactions among users, and as part of its operations and incentive mechanisms, it imposes certain fees. In the context of this trading strategy, the only fees applicable would be the interest rates on the loan. The interest rate is typically expressed as an Annual Percentage Yield (APY) and is accrued continuously. Currently, Aave supports variable interest rates, which fluctuate based on market conditions, the borrowed asset, and supply and demand dynamics within the Aave.

Another aspect of Aave for the trading strategy is collateralization and the avoidance of liquidation. Liquidation occurs when a borrower's position is forcibly closed, and their collateral assets are sold off in cases of default or insufficient collateral. When borrowing from Aave, borrowers are required to allocate a certain percentage of the loan value as collateral, known as the Loan-to-Value (LTV) ratio [50]. For instance, if a token has a 75% LTV, borrowers can borrow 0.75 ETH worth of the corresponding token for every 1 ETH worth of collateral provided. However, as token prices fluctuate, the ratio between the borrowed token value and the collateral value changes, posing risks for both lenders and Aave. To safeguard lenders and maintain protocol solvency, a liquidation threshold is established. If the collateral value falls below this threshold, the borrower's position becomes vulnerable to liquidation. In such cases, the collateral is auctioned off to repay the outstanding debt to the lenders. Typically, the liquidation threshold is set 5-10% higher than the asset's LTV. For WETH, the Loan-to-Value ratio is 82.5%, and the liquidation threshold is 86%.

Liquidity Pools

The fundamental mechanism that enables Aave's functionality of lending and borrowing is liquidity pools. Users deposit their cryptocurrency assets into Aave's liquidity pools. These assets serve as collateral and contribute to the overall liquidity of the protocol. The deposited assets in the liquidity pools create reserves of available liquidity. These reserves are utilized to fulfill borrowing demands from other users within the Aave ecosystem.

Lending and Borrowing

Lending works by lenders depositing their cryptocurrency assets into Aave's liquidity pools. These assets act as collateral and are represented by interest-bearing tokens called aTokens. The aTokens represent the user's share of the deposited assets. Interest is earnt immediately and is accrued in real-time and reflected in an increase in the quantity of aTokens held by the depositor.

Borrowing works by borrowers using their deposited assets as collateral to borrow other assets from the liquidity pools. The value of the collateral determines the borrowing capacity. The borrowing capacity is calculated by paramters set by Aave, one of which is the maximum loan-to-value (LTV) ratio. Once the borrower requests a loan, if the

borrower's collateral meets the necessary requirements, they can proceed with the loan and the borrowed funds are transferred to the borrower's wallet. In addition to this, borrowers pay interest on the borrowed amount based on the prevailing interest rates. Aave offers both variable and stable interest rates for borrowers. Variable rates fluctuate based on market dynamics, while stable rates remain fixed. This flexibility allows users to choose the borrowing option that best suits their needs. The interest on a loan is accrued in real-time, second by second, and the borrower decides when to repay it, as long as the loan is safe from liquidation. Liquidation is what happens if the value of a borrower's collateral falls below the liquidation threshold due to market volatility or other factors, the collateral may be liquidated. Liquidators can purchase the collateral at a discounted price to ensure the solvency of the liquidity pool.

3.5 Liquidity Pool Pairs of Interest

As previously mentioned, Uniswap is a decentralized exchange protocol that facilitates swapping of cryptocurrencies through the use of liquidity pools. One of the remarkable features of Uniswap is its support for even the smallest cryptocurrencies. In fact, it currently possesses 12,182 liquidity pools, as obtained via the Uniswap V3 subgraph. The subgraph provides valuable insights into the liquidity pools within the Uniswap ecosystem.

Table 3.2 showcases the diversity of these pools. Some pools exhibit high trading volumes, indicating significant activity and demand for those specific pairs. On the other hand, there are also pools with zero liquidity, which could be attributed to various reasons. It could be due to a lack of demand for one of the token pairs offered in the pool, or it may indicate that the pool is relatively new and has not attracted significant participation yet.

To enhance the chances of successful swaps and maximize potential profitability, my strategy is centered on liquidity pools with a trading volume surpassing $10,000,000 (or $10 million) and pools that include tokens supported by Aave. Additionally, I exclude any liquidity pools that do not involve WETH, which is a tokenized form of ETH (Ethereum's native cryptocurrency). WETH's key advantage lies in its ability to align with ERC-20 standards, enabling broader compatibility and utilization across the Ethereum ecosystem.

3.5.1 Correlated and Cointegrated Liquidity Pools

Once the liquidity pools of interested in have been identified, it is crucial to filter the pool pairs based on their correlation. This is because correlated pairs tend to exhibit similar pricing movements thus would be possible to apply the mean reversion trading strategies. However, it is important to strike a balance between a highly correlated pair and a low correlated pair, as excessively high correlation can result in minimal price deviations, leading to fees that exceed the deviations and resulting in overall losses instead of profits. To address this, I have set a condition where the correlation coefficient (ρ) should fall within the range of $0.992 \leq \rho \leq 0.997$. Figure 3.4 shows the correlation matrix of the liquidity pools that meet the initial requirement of having a volume greater than $10 million volume ratio and include WETH. Notably, the pools associated with tokens aiming to be pegged to the US Dollar, such as USDT, USDC, and DAI, exhibit the expected high correlation with each other. However, an exception is observed with pool 0xe0554a476a092703abdb3ef35c80e0d76d32939f, which exhibits a lower correlation with other liquidity pools containing these stable coins. Consequently, this lower correlation opens up additional avenues for potential arbitrage opportunities within the pool.

To determine which liquidity pools are cointegrated, I employed the Engle-Granger approach. The Engle-Granger test for cointegration involves several steps. Firstly, a unit

pool address	token0	token1	volume in USD	created at timestamp	feetier
0x88e6a0c2d...	USDC	WETH	375230561243.465	1620250931	500
0x8ad599c3...	USDC	WETH	70454095868.0967	1620169800	3000
0x11b815efb...	WETH	USDT	62385006691.8387	1620251172	500
0x3416cf6c7...	USDC	USDT	57192593471.8346	1636825557	100
0x4585fe772...	WBTC	WETH	49170385539.9928	1620246230	500
0x4e68ccd3e...	WETH	USDT	30135014933.0963	1620232628	3000
0x60594a40...	DAI	WETH	26075053939.434	1620237823	500
0xcbcdf9626...	WBTC	WETH	21870989841.1326	1620158974	3000
0x5777d92f2...	DAI	USDC	16143305036.8948	1636771503	100
0x7858e59e0...	USDC	USDT	15473402409.0591	1620159478	500
0x99ac8ca70...	WBTC	USDC	12568187132.1649	1620241995	3000
0xc2e9f25be...	DAI	WETH	12519316091.9979	1620159368	3000
0xe0554a476...	USDC	WETH	9381529300.20357	1636926269	100
0x6c6bc977e...	DAI	USDC	7219493916.70291	1620158293	500
0xac4b3dac...	APE	WETH	6621032721.87519	1647516735	3000
0x8c54aa2a3...	FEI	USDC	6206853090.73714	1621839430	500
0x53dd58b3...	sOHM	gOHM	0	1652914688	10000
0xbc90c4de...	SHIB	NSTIC	0	1652910391	100
0xaa1297b0...	BUSD	DPC	0	1674655715	100
0x75087e533...	DAI	ICAP	0	1652899566	10000
0xc65a68019...	stkAAVE	FRAX	0	1652817927	10000
0x94589b18...	VVV	SOL	0	1674701603	10000
0xf42f0def92...	APEFI	ApeUSD	0	1652808878	3000
0xb9ba65f15...	FRAX	ApeUSD	0	1652808680	500
0x1dfb167f1...	GHD	WETH	0	1652784327	3000

Table 3.2: A selection of liquidity pools on Uniswap

root test is performed individually on each time series using methods like the Augmented Dickey-Fuller (ADF) test or the Phillips-Perron (PP) test. These tests assess whether the time series are stationary or exhibit unit roots (non-stationary) individually. For cointegration to hold, both time series must be non-stationary.

Once it is established that both time series are non-stationary, an estimation of the cointegration equation is derived. Typically, a linear regression model is employed to capture the long-term relationship between the time series variables. This equation provides an understanding of how the variables are linked over a long period.

Subsequently, a unit root test is conducted on the residuals obtained from the regression analysis. If the residuals are stationary (indicating the absence of unit roots), it suggests the presence of cointegration between the time series. This implies that the variables move together in the long run, even if short-term deviations occur.

By following the aforementioned sequence of steps in the Engle-Granger test, it enables us to determine which liquidity pools demonstrate cointegration, thereby emphasizing their interconnectedness and mutual relationship over time. The outcome of the cointegration tests for all possible combinations of liquidity pools is presented in Table 3.3. This table provides a comprehensive list of the liquidity pool pairs along with the results of their respective cointegration tests and the correlation coefficient of each combination.

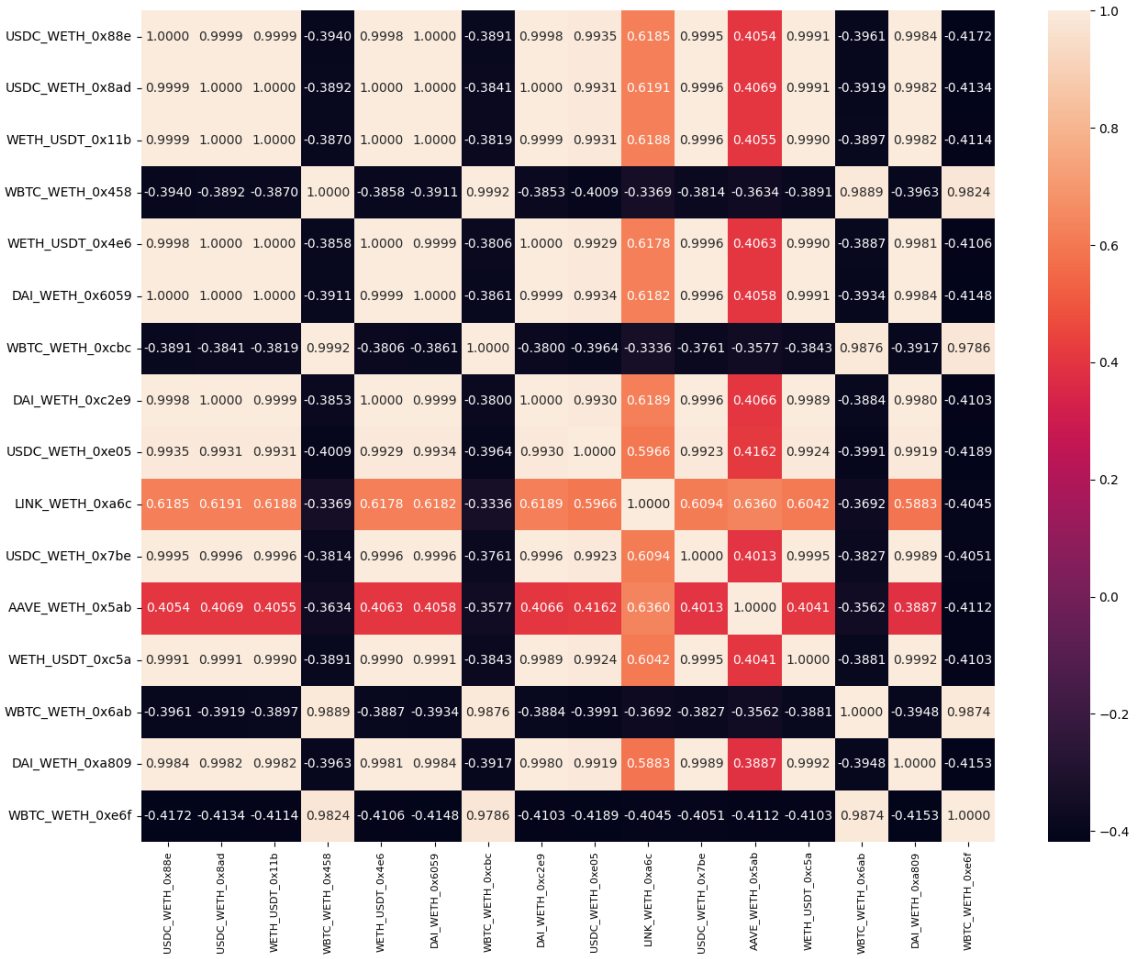

Figure 3.4: Correlation Matrix of the Liquidity Pools that meet the $10 million volume and include WETH

pool1	pool2	t-statistic of unit-root test on residuals	Critical Values			Corr Coeff
			1%	5%	10%	
USDC_WETH_0x88e6...	USDC_WETH_0xe055...	-11.306176	-3.898069	-3.337038	-3.045081	0.99347
USDC_WETH_0x8ad...	USDC_WETH_0xe055...	-11.210146	-3.898082	-3.337046	-3.045086	0.99307
WETH_USDT_0x11b...	USDC_WETH_0xe055...	-10.010746	-3.898069	-3.337038	-3.045081	0.99315
WETH_USDT_0x4e68...	USDC_WETH_0xe055...	-9.913205	-3.898081	-3.337045	-3.045085	0.99291
DAI_WETH_0x60594...	USDC_WETH_0xe055...	-11.395276	-3.898069	-3.337038	-3.045081	0.99338
DAI_WETH_0xc2e9f2...	USDC_WETH_0xe055...	-10.418722	-3.89831	-3.337173	-3.045174	0.99298
USDC_WETH_0xe055...	WETH_USDT_0xc5af...	-9.7789	-3.901414	-3.338902	-3.046374	0.99243

Table 3.3: Cointegration test on Liquidity Pool pairs

Based on the results obtained, we focus our further investigation solely on the pairs of liquidity pools that exhibit cointegration under a 1% confidence level. This filtering process allows us to narrow down our analysis to those pairs where a long-term relationship exists, suggesting that they move together in a concerted manner.

By concentrating on the cointegrated pairs, we can delve deeper into exploring their dynamics, assessing their behavior, and leverage the interconnectedness between these liquidity

pools. This approach enables us to prioritize our analysis and concentrate on the most relevant liquidity pool combinations that offer potential opportunities for profitable trading or other related activities.

Chapter 4

Implementation of the Trading Systems

This section provides a comprehensive overview of the two trading system implementations. The first one focuses on the backtesting system, which involves simulating and evaluating strategies in a controlled environment. In this system, historical market data is utilized to test and analyze the performance of various strategies. The second implementation is the live system, designed to execute trades on the Ethereum blockchain using smart contracts. This system operates in real-time and interacts with the live market, allowing for actual trades to be executed based on predefined strategies. Together, these two implementations provide a comprehensive framework for testing, evaluating, and deploying trading strategies, both in simulated environments and on the live blockchain.

4.1 Backtesting System

To develop a resilient backtesting system, a dedicated class was constructed to streamline the process of testing trading strategies, the system is entirely written in Python as it possesses countless libraries including Pandas and Numpy which are extensively used to data manipulation and analysis, it also is good with integrating with many data sources such as GraphQL which is required for this project. To evaluate a particular strategy, the `backtest` function requires `cointegrated_pair`, a tuple of the names of the liquidity pools that the strategy is to be evaluated on, `strategy`, the strategy, and finally, the `initial_investment` in ETH. The first argument is required and used to retrieve the relevant historical prices. The second parameter is self-explanatory, the strategy to test. Finally, the inclusion of the initial investment amount as an input parameter enables users to simulate the performance of their strategies with a specific starting capital, it also helps analyze how the various transaction costs affect the ability to trade if any trades do result in a loss. The `backtest` function iterates through historical data calling the strategy's `generate_signal` and executing the trade orders it receives at each timestep. However, to do this historical data is required hence, the first step is to collect the historical data. The logical process of backtesting a pair can be seen in Figure 4.1.

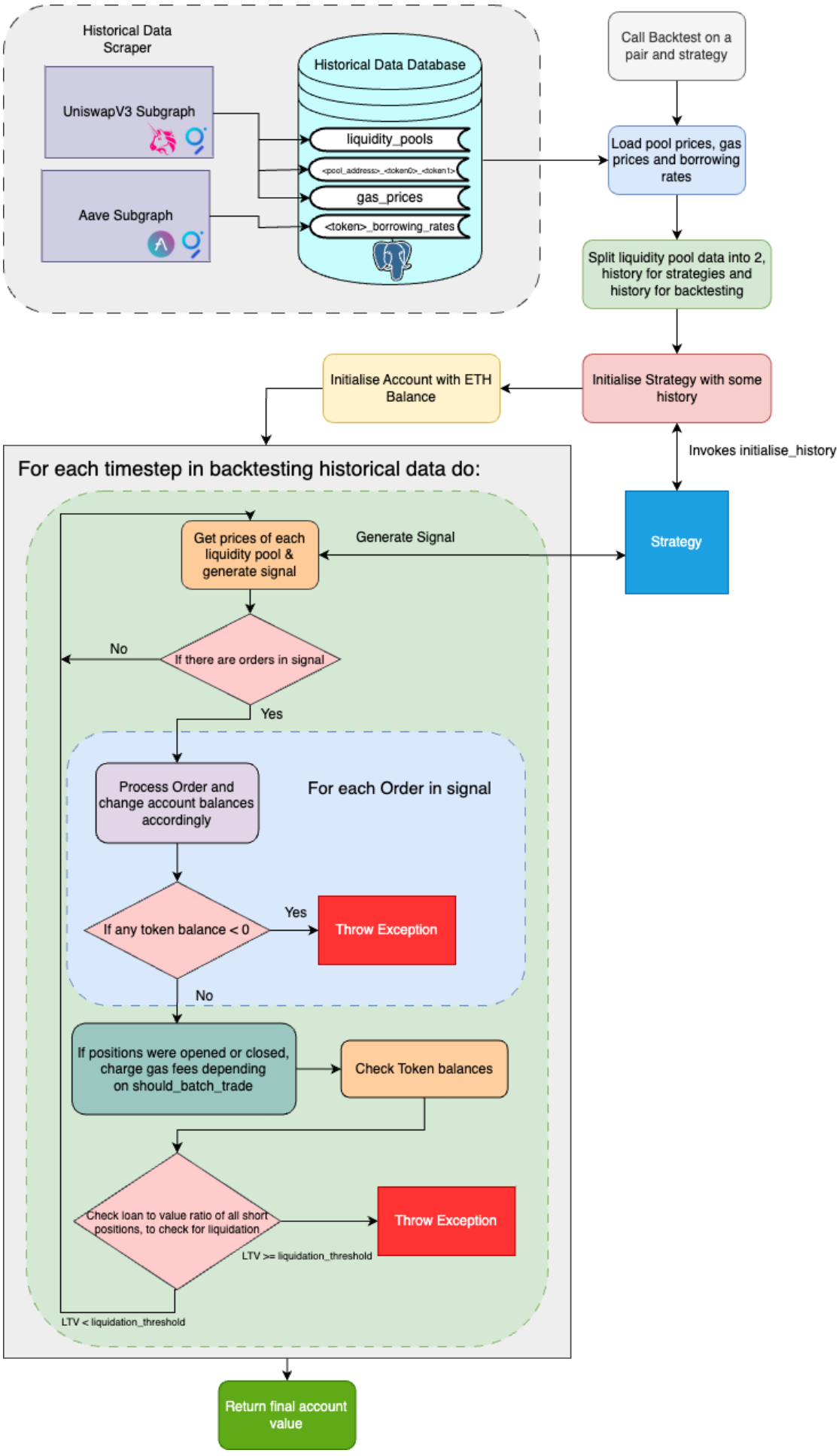

Figure 4.1: Flowchart of backtesting a strategy on a liquidity pool pair

4.1.1 Data Collection and Storage

In order to simulate the market as accurately as possible the system should have access to reliable and accurate historical market data, including price, volume, and other relevant indicators. Therefore, I retrieve all of my data from the Uniswap and Aave protocols' subgraph using the Graph [51]. The Graph is a decentralized protocol for indexing and querying blockchain data hence making the data provided 100% reliable as it indexes directly on the Ethereum blockchain. Subgraphs serve as GraphQL APIs designed to facilitate querying and extracting data from the blockchain. These APIs adhere to a specific schema outlined by the protocol, enabling seamless communication between the protocol and the underlying blockchain. Therefore, I employ the Uniswap V3, Aave V2, and Aave V3 subgraphs to obtain the data required for the backtesting system.

To store the necessary data for backtesting and trading, a PostgreSQL database is used this is to provide consistency, flexibility with numerous data types and also supports a large set of SQL functions that allow for advanced querying. The database adopts the following schema: The `liquidity_pools` contains data about the liquidity pools that ex-

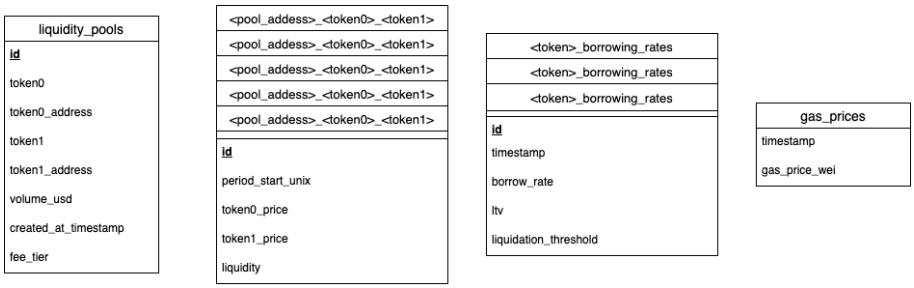

Figure 4.2: Tables contained in the database

ist on Uniswap V3. After obtaining all of the data it is found that it possesses 12,182 liquidity pools. However, a substantial portion of these pools exhibit minimal or negligible trading volume. As a result, a criterion is established to selectively include only those liquidity pools that involve tokens supported by Aave and possess a trading volume exceeding $10,000,000 (or $10 million). This filtering condition ensures that the collected and stored data holds significance and relevance for research purposes, as these pools would allow for short selling.

Once these pools have been identified, pricing data about each pool that meets this condition is collected again using the Uniswap V3 subgraph. The following shows the GraphQL query:

```
query ($id: ID!, $prev_max_time: Int!) {
    pool(id: $id) {
        poolHourData(where: {periodStartUnix_gt: $prev_max_time}, orderBy:
    periodStartUnix, first: 1000) {
            id
            token0Price
            token1Price
            periodStartUnix
            liquidity
            feesUSD
        }
    }
}
```

This query returns an array of dictionaries containing the pre-specified pricing datapoints at a frequency of every hour. Due to the limitations imposed by the subgraph, the re-

sults are constrained to a maximum size of 1000 entries. To overcome this limitation and obtain the complete dataset, the query is executed iteratively. The previous maximum time, referred to as `prev_max_time`, is passed as an argument in subsequent queries to fetch the remaining data, i.e. `prev_max_time = hourlyData[-1]['periodStartUnix'] if len(hourlyData) > 0 else prev_max_time`. This data is stored in tables of the form *<pool_ address>_<token0>_<token1>*.

In a similar manner, obtaining the interest rates for borrowing necessitates the utilization of two of Aave's subgraphs. This requirement arises due to Aave's migration to Version 3 in March 2022, resulting in a transitional period where both Uniswap V3 and Aave V2 were concurrently utilized. The schemas for the two are different, however for the data we require, the borrowing rate, loan-to-value and liquidity threshold, the schema is consistent and the same query can be used:

```
query ($symbol: String!, $prev_max_time: Int!) {
    reserves(where: {symbol: $symbol}) {
        id
        symbol
        lifetimeBorrows
        baseLTVasCollateral
        reserveLiquidationThreshold
        borrowHistory(
            where: {timestamp_gt: $prev_max_time}, first: 1000, orderBy:
    timestamp, orderDirection: asc) {
            id
            timestamp
            borrowRate
        }
    }
}
```

During the table initialization process, requests are made to both the V2 and V3 GraphQL endpoints. However, when updating the table, only the V3 endpoint is utilized for sending requests. The data is stored in tables of the form *<symbol>*_borrowing_rates, where *<symbol>* are all of the tokens that are present in the liquidity pools that are of interest.

To collect the gas price history, the transaction history is queried at each hour since Uniswap migrated to V3. This is because querying in the same as the pricing data and borrowing rate history is too exhaustive as transactions occur every second. Therefore, it is more efficient to query at each hour with a window to retrieve the gas price at the closest hour as follows:

```
query ($min_time: Int!, $max_time: Int!) {
    transactions(where: {timestamp_gt: $min_time, timestamp_lt: $max_time},
     first:1000, orderBy: timestamp, orderDirection: asc) {
        id
        timestamp
        gasPrice
    }
}
```

The following pseudocode shows how to fetch the gas price at each hour, Algorithm 1:

4.1.2 Types of Orders and Execution

There are numerous order types that the backtesting supports due to measures to enusre a positive balance and avoid liquidation. The types of orders are; `BUY ETH`, `WITHDRAW`,

Algorithm 1 Retrieval of hourly gas prices where min_time & max_time are arguments

$rows\_set \leftarrow \{\}$
for $timestamp$ from min_time to $max\_time + (60 \times 60), step\_size = (60 \times 60)$ **do**
  $found\_result \leftarrow False$
  $window\_size\_in\_minutes \leftarrow 5$
  while *not* $found\_result$ **do**
    $transaction\_data \leftarrow$ GraphQL Query with arguments: $\{$"min_time" : $timestamp - (60 * window\_size\_in\_minutes)$, "$max\_time$" : $timestamp + (60 * window\_size\_in\_minutes)\}$
    if $transaction\_data! = 0$ **then**
      $transaction\_data\_sorted = sorted(transaction\_data, key = lambda\ x : abs(timestamp - int(x['timestamp'])))$
      $rows\_set.update(timestamp : (timestamp, transaction\_data\_sorted[0]['gasPrice']))$
      $found\_result \leftarrow True$
    else
      $window\_size\_in\_minutes \leftarrow window\_size\_in\_minutes + 5$
    end if
  end while
  insert $rows\_set[timestamp]$ into `gas_prices` table
end for

`DEPOSIT`, `SWAP`, `OPEN BUY`, `CLOSE BUY`, `OPEN SELL`, `CLOSE SELL`. The precedures of each are outlined below.

`BUY ETH`

The `BUY ETH` order is used to swap a percentage of the accounts balance from WETH to ETH. The arguments is an float between 0 and 1. The logic of the execution, excluding gas fees, of the order is outlined below.

```
amount_to_swap = self.account['WETH'] * order[1]
self.account['WETH'] = self.account['WETH'] - amount_to_swap
self.account['ETH'] = self.account['ETH'] + amount_to_swap
```

`SWAP`

The `SWAP` order is to execute a simple swap for either token0 or token1 of the liquidity pool. This order is only used as a precautionary order, if the balance of the WETH balance gets too low. It takes 2 arguments, the first indicating whether it is swapping for token0 or token1 and the second parameter is a list of swaps, tuples containing with pool and the quantity, that would like to be swapped. As can be seen in the code excerpt below, the amount received in the account is multiplied by `(1 - swap_fees[swap_token])`, this is because Uniswap charges a percentage of the swap specified by `swap_fees[swap_token]`.

```
if is_for_token0:
    for swap_for_token0 in swaps:
        swap_token, swap_volume = swap_for_token0
        self.account['WETH'] = self.account['WETH'] - (swap_volume * prices[f'P{swap_token[1]}'])
        self.account[swap_token] = self.account[swap_token] + (swap_volume * (1 - swap_fees[swap_token]))
else:
    for swap_for_token1 in swaps:
        swap_token, swap_volume = swap_for_token1
```

```
        self.account[swap_token] = self.account[swap_token] - (swap_volume / prices[f'P{swap_token[1]}'])
        self.account['WETH'] = self.account['WETH'] + (swap_volume * (1 - swap_fees[swap_token]))
```

OPEN BUY

Similar to the SWAP order, OPEN BUY opens a buy position. As mentioned above opening a buy order is simply swapping token1 for token0. The parameters of buying are the target token and the volume.

```
self.account['WETH'] = self.account['WETH'] - (volume * buy_price)
self.account[token] = self.account[token] + (volume * (1 - swap_fees[token]))
```

CLOSE BUY

Closing a buy position is similar to opening a buy position, however, the swap is in the other direction, i.e. from token to WETH. The only argument is the id of the buy order. To account for Uniswap's fees, the volume that was received, after the initial swap, is calculated then this volume is swapped back to WETH.

```
buy_token, bought_price, buy_volume, buy_timestamp = self.open_positions['BUY'][buy_id]
volume_bought = buy_volume * (1 - swap_fees[buy_token])
self.account[buy_token] = self.account[buy_token] - volume_bought
self.account['WETH'] = self.account['WETH'] + (prices[f'P{buy_token[1]}'] * (volume_bought * (1 - swap_fees[buy_token])))
```

OPEN SELL

Opening a sell position involves borrowing a token and then swapping the token. The parameters of selling are the target token and the volume. It also consists of depositing the required amount to borrow the volume ordered, calculated by the value to loan ratio.

```
# Borrow token and Deposit collateral
amount_to_move_to_collateral_WETH = ((volume * sell_price) / ltv_eth)
self.account[token] = self.account[token] + volume
self.account['WETH'] = self.account['WETH'] - amount_to_move_to_collateral_WETH
self.account['collateral_WETH'] = self.account['collateral_WETH'] + amount_to_move_to_collateral_WETH

# Swap borrowed tokens to WETH
self.account[token] = self.account[token] - volume
self.account['WETH'] = self.account['WETH'] + (volume * (1 - swap_fees[token]) * sell_price)
```

CLOSE SELL

Closing a sell position is more complicated that as in requires to swap back from the token to WETH, repay the loan with interest and finally return collateral. Swapping back to the required amount of tokens is the first step. For this the variable Annual Yield Rates between the timestamp of the opening of the position and closing positon is used to calculate the amount of interest required to be paid as this is cumulated every second. Once the volume of tokens required to be returned is calculated, the equivelent amount of

WETH plus accounting for Uniswap fees is swapped to obtain these tokens. Finally, the tokens are repayed and the collateral is returned.

```
sell_token, sold_price, sell_volume, sell_timestamp = self.open_positions['SELL'][sell_id]

# Swap WETH back to Token
volume_required_to_return = sell_volume
previous_timestamp = sell_timestamp

for apy_idx in apy[sell_token].index:
    local_apy = apy[sell_token].loc[apy_idx]['borrow_rate']
    number_of_seconds = apy[sell_token].loc[apy_idx]['timestamp'] - previous_timestamp
    secondly_yield = (1 + local_apy)**(1 / (365*24*60*60))
    volume_required_to_return *= secondly_yield ** number_of_seconds
    previous_timestamp = apy[sell_token].loc[apy_idx]['timestamp']

self.account['WETH'] = self.account['WETH'] - (sold_price * volume_required_to_return / (1 - swap_fees[sell_token]))
self.account[sell_token] = self.account[sell_token] + volume_required_to_return

# Return Borrowed Tokens and Collatoral
self.account[sell_token] = self.account[sell_token] - volume_required_to_return
self.account['WETH'] = self.account['WETH'] + self.account['collateral_WETH']
self.account['collateral_WETH'] = 0
```

`WITHDRAW`

The `WITHDRAW` order's function is to withdraw some collateral that is stored in Aave. It is not used in the strategies however is implemented in case of developing further strategies that may require this functionality. The only parameter is the amount that would like to be withdrawn.

```
self.account['WETH'] = self.account['WETH'] + withdraw_amount
self.account['collateral_WETH'] = self.account['collateral_WETH'] - withdraw_amount
```

`DEPOSIT`

The `DEPOSIT` order's function is to deposit some additional collateral that is stored in Aave. This order is used when the collateral value not properly covering the loan value, hence avoiding liquidation. The only parameter is the amount that would like to be depositted.

```
self.account['WETH'] = self.account['WETH'] - deposit_amount
self.account['collateral_WETH'] = self.account['collateral_WETH'] + deposit_amount
```

4.1.3 Gas Fees

The remaining fee that is yet to be discussed is gas fees of each order. To best approximate the gas fee it is paramount to have an approximation to how much gas is used in each. Therefore, the smart contract for trading was written and run on a forked Ethereum mainnet, to be able to accurately simulate the contracts behaviour, see Subsection 4.2.1 to see the implementation of the smart contract. Using Hardhat, the gas usage of each

function was tested and Figure 4.3 shows the results. It can be seen that some functions have a tighter spread, between the maximum and minimum, compared to others therefore to better understand this, plots of using various parameters on the different functions provided a deeper insight into the cause of this, Figures 4.4 and 4.5.

Solc version: 0.7.6		Optimizer enabled: true		Runs: 200	Block limit: 30000000 gas	
Methods						
Contract	Method	Min	Max	Avg	# calls	eur (avg)
Swaps	borrowToken	423221	429540	423777	92	-
Swaps	closeBuySellPositions	501246	511946	502811	182	-
Swaps	depositCollateral	211596	251105	214353	54	-
Swaps	openBuySellPositions	523792	566537	526749	184	-
Swaps	repayBorrowedToken	400754	401069	401056	90	-
Swaps	swapEthForWeth	50776	64456	58136	342	-
Swaps	swapExactUsingPool	93702	181561	126108	54	-
Swaps	swapExactUsingRouter	128093	178762	152002	54	-
Swaps	swapWethForEth	61476	61485	61482	54	-
Swaps	withdrawCollateral	181067	198163	181733	52	-
Deployments				% of limit		
Swaps		-	-	1581910	5.3 %	-

Figure 4.3: Results from running smart contract functions

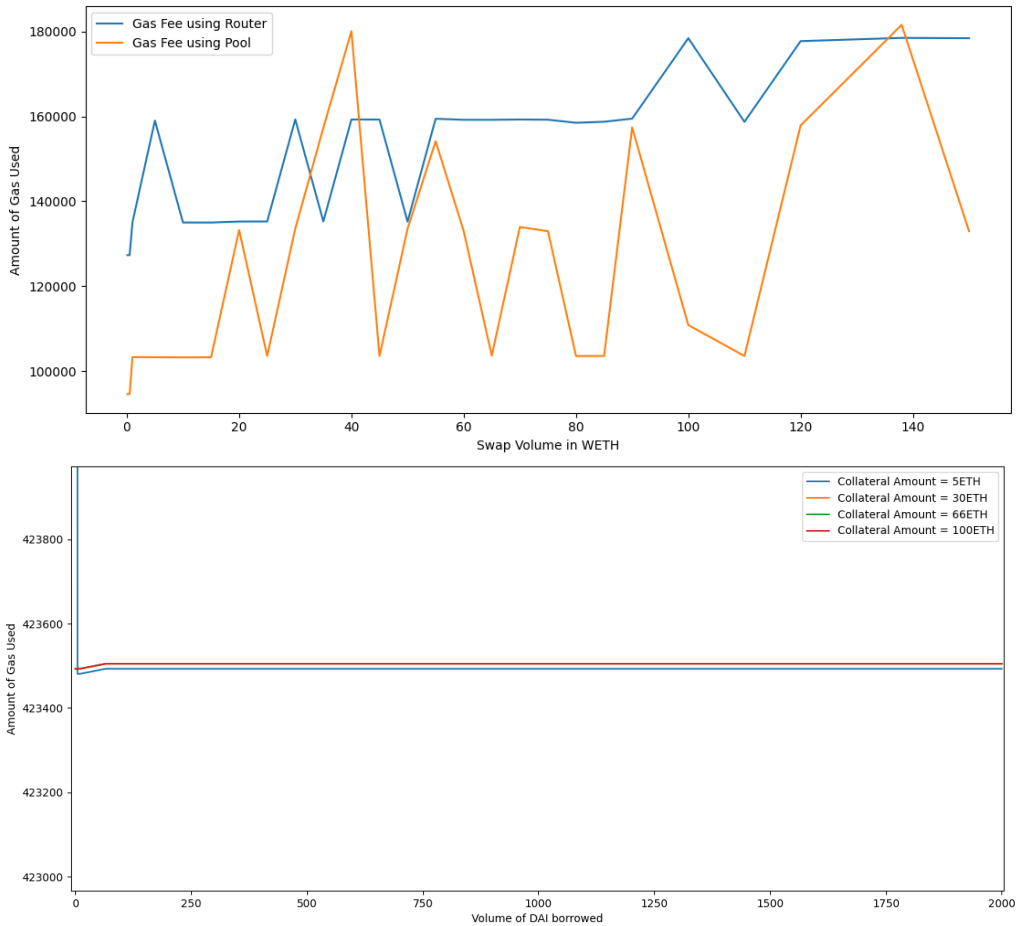

Figure 4.4: Gas Usage Plots of Gas Used by Swapping (Top) and Borrowing (Bottom)

Upon analyzing these plots, it becomes apparent that the majority of the functions exhibit a high level of consistency. However, it is noteworthy that for the smallest quantity tested,

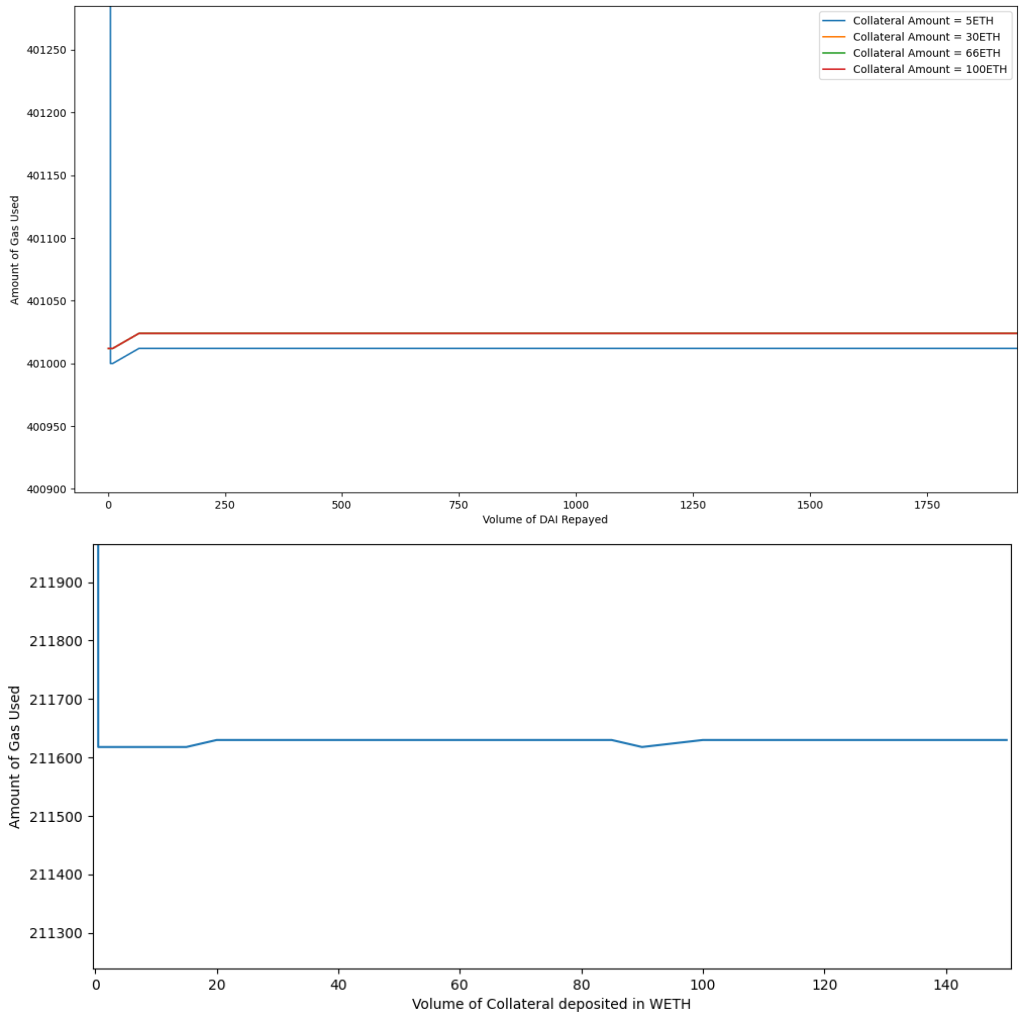

Figure 4.5: Gas Usage Plots of Gas Used by Repaying Borrowed Tokens (Top) and Depositting additional tokens (Bottom)

the gas consumption is significantly higher compared to that of larger quantities. Nevertheless, once this threshold is surpassed, the gas usage remains consistent. Therefore the average of the gas usage of each operation is used in the backtesting system as it provides a representative estimate of the gas consumption across different quantities.

Furthermore, another feature that is interesting is the gas usage by swaps and the different methods. The initial method utilizes Uniswap's Router, which scans all available liquidity pools to identify the optimal price, whereas the second method involves direct swapping within a specific liquidity pool. Figure 4.4 illustrates that, on average, the router-based approach exhibits higher gas usage compared to direct swapping using a liquidity pool. However, directly swapping using a liquidity pool exhibits greater volatility in terms of gas consumption but also consistently returns to values above 100,000, because of this the average gas usage of swapping using a liquidity pool is used.

It can also be seen that combining the operations to open/close buy and sell positions consumes less gas compared to if were to be executed seperately. This also confirms the claims in Wang's paper, that batching the operations together is more effective than executing them seperately [52].

To implement this in the backtesting system, each order deducts some ETH where this quantity is calculated by how much gas is used by the order type multiplied by the gas

price. For example, in the case of buying more ETH:

```
amount_to_swap = self.account['WETH'] * order[1]
self.account['WETH'] = self.account['WETH'] - amount_to_swap
self.account['ETH'] = self.account['ETH'] + amount_to_swap - (
    GAS_USED_BY_BUYING_ETH * gas_price_in_eth)
```

However, deducting ETH for gas fees for open and close orders is done differently to better understand the impact of combining the operations on the returns of the strategies. Thus, instead of being deducted immediatly at each order, it is deducted after iterating through the signal:

```
if len([order[0] for order in signal if order[0] == 'OPEN']) == 2:
    if strategy.should_batch_trade:
        self.account['ETH'] = self.account['ETH'] - (
    GAS_USED_BY_OPEN_BUY_AND_SELL_POSITION * gas_price_in_eth)
    else:
        self.account['ETH'] = self.account['ETH'] - ((GAS_USED_BY_SWAP +
    GAS_USED_BY_SWAP + GAS_USED_BY_BORROW) * gas_price_in_eth)

if len([order[0] for order in signal if order[0] == 'CLOSE']) == 2:
    if strategy.should_batch_trade:
        self.account['ETH'] = self.account['ETH'] - (
    GAS_USED_BY_CLOSE_BUY_AND_SELL_POSITION * gas_price_in_eth)
    else:
        self.account['ETH'] = self.account['ETH'] - ((GAS_USED_BY_SWAP +
    GAS_USED_BY_SWAP + GAS_USED_BY_REPAY) * gas_price_in_eth)
```

Therefore, by passing a boolean to the strategy that indicates if the open or close positions should be batched, the necessary amount of fees is deducted.

4.1.4 Validating Balance Health

To maintain the integrity and effectiveness of the trading strategy, it is crucial to incorporate various checks and safeguards after each trade and at each timestep. One key aspect involves monitoring the balance of each token to ensure that it remains positive. After every order execution, the balances of the tokens involved are checked, and if any of them fall below zero, an exception is triggered.

```
if self.account['T1'] < negative_threshold:
    raise Exception('Account balace goes below 0 - T1')

if self.account['T2'] < negative_threshold:
    raise Exception('Account balace goes below 0 - T2')

if self.account['WETH'] < negative_threshold:
    raise Exception('Account balace goes below 0 - WETH')

if self.account['ETH'] < negative_threshold:
    raise Exception('Account balace goes below 0 - ETH')
```

Furthermore, to simulate the potential liquidation of a sell position's loan, the loan-to-value ratio is calculated. This ratio serves as an indicator of the position's health and risk level. If the loan-to-value ratio breaches the predefined liquidation threshold, indicating that the position is approaching an unsustainable state, an exception is thrown.

```
sell_token, sold_price, sell_volume, _ = sell_trade
current_token_price = prices[f'P{sell_token[1]}']
curr_value_of_loan_pct = (sell_volume * current_token_price) / self.account
    ['collateral_WETH']
if round(curr_value_of_loan_pct, 4) > liquidation_threshold:
    raise Exception(f'Short position liquidated')
```

4.2 Live Trading

To run the strategies, a similar approach is required however the executions system and balance tracking is altered. For the system there are multiple components that are required for live trading to occur. The first is the ability to interact with the blockchain to execute the orders, this is done by smart contracts, the second is to manage the state of the strategy, accounts and the open positions for the stratgies to use to generate signals and finally a method to convert the signal into smart contract invokations to execute the orders. The workflow of trading on the live executiom system.

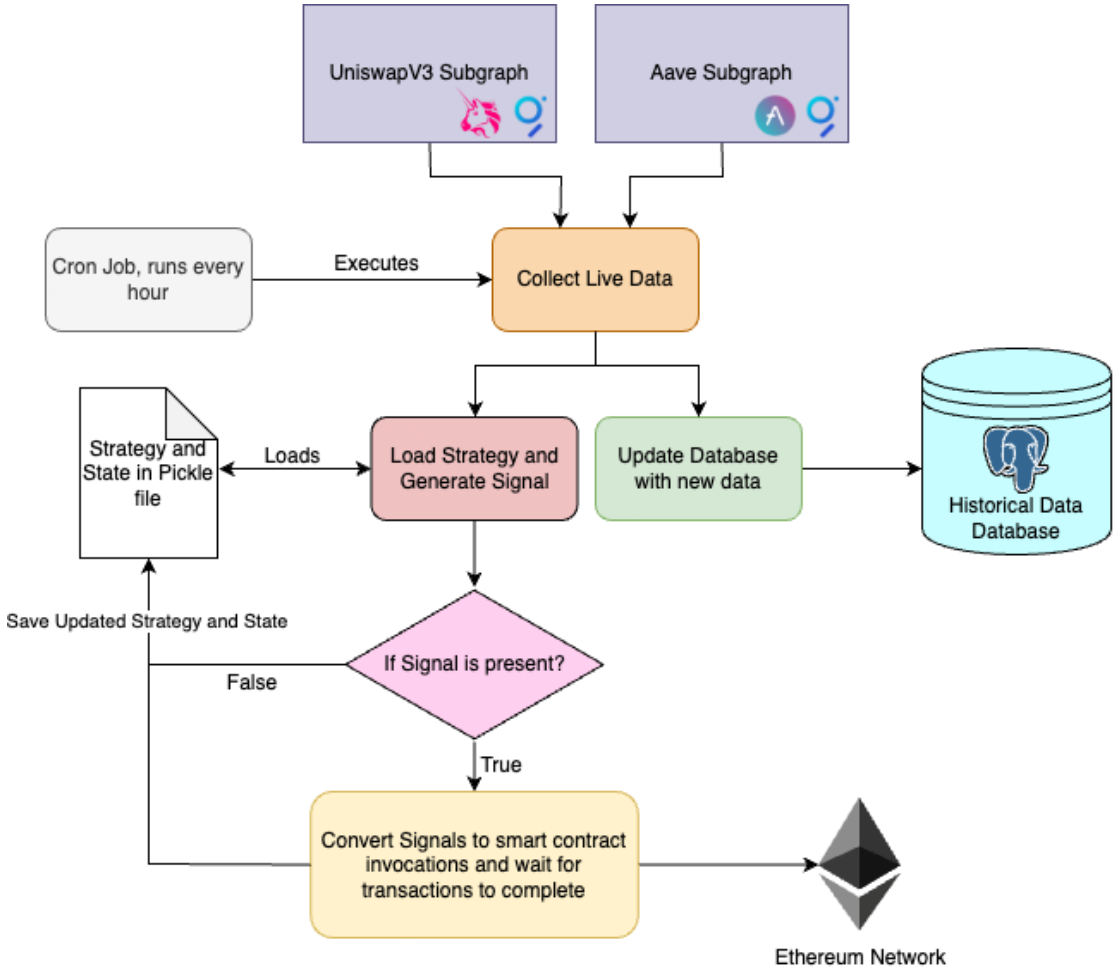

Figure 4.6: Flowchart of trading a strategy live

4.2.1 Smart Contracts

In order to be able to interact with uniswap and aave over ethereum, smart contracts are written, which are then deployed on the blockchain network. Functions that the smart contract posseses can then be executed with anyone that knows the address of the contract on the network. Smart contracts are programmable protocols that are written in Solidity, a high-level programming language designed for creating and executing smart contracts

on Ethereum. For the purpose of trading using the aforementioned strategies, a few the following functionalities are required to be implemented; wrapping ETH for WETH, unwrapping WETH, swapping token A for token B using a given liquidity pool and finally borrowing and repaying some of token A using WETH as collateral. The first 2 functions are required if the trader has only ETH in their account, thus by wrapping the ETH, they would be able to trade ETH as a ERC-20 token.

Wrapping of ETH and Unwrapping WETH

This functionality are required as if the trader has only ETH in their account, thus by wrapping the ETH, they would be able to trade ETH in it's ERC-20 token, WETH. ERC-20 is a widely adopted technical standard utilized for the creation of interchangeable tokens on the Ethereum blockchain. It provides developers with the framework to design tokens that are compatible with various applications and services operating within the Ethereum ecosystem. These tokens are designed to represent a variety of assets, most importantly cryptocurrencies. The functions of these functionalities can be seen below, where `swapEthForWeth` wraps ETH and `swapWethForEth` unwraps WETH. The `swapEthForWeth` function first uses the `IWETH` interface, which is itself an ERC20 interface, to allow functions such as tranfer of ownership of the token, to deposit an amount of ETH in exchange for WETH, which is then transfered back to the address of the caller. The `swapWethForEth` function unwraps the WETH by first sending the specified volume of WETH from the caller to the smart contract, withdrawing the specified amount from the IERC20, which calls the `receive` function to enable recieving of ETH and once the ETH is withdrawn into the smart contract's account, it is transfered back to the caller.

```
interface IWETH is IERC20 {
  function deposit() external payable;

  function withdraw(uint amount) external;
}

address public immutable wethAddress = 0
    xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2;

function swapEthForWeth() external payable {
    IWETH weth = IWETH(wethAddress);
    weth.deposit{ value: msg.value }();
    weth.transfer(msg.sender, msg.value);
}

function swapWethForEth(uint256 amount) external payable {
    IWETH(wethAddress).transferFrom(msg.sender, address(this), amount);
    IWETH(wethAddress).withdraw(amount);
    msg.sender.transfer(address(this).balance);
}

receive() external payable {}
```

Swapping using a Uniswap Liquidity Pool

Swapping using a Uniswap liquidity pool is also another key functionality. This is done by using the `IUniswapV3Pool` interface to expose the liquidity pool's functions and calling swap on it using the necessary parameters. The parameters, in their respective order, are the address of the recipient of the swapped tokens, a boolean on whether the swap direction is from token0 to token1 or vice versa, the volume of the source token that would like to be swapped, an integer (sqrtPriceLimitX96) that manages how much slippage is acceptable, and finally abi encoded data that is used in the callback function. When calling `swap` function, the sqrtPriceLimitX96 parameter is set to ignore the effects of slippage and proceed with the swap. Finally, once the swap has taken place, Uniswap calls the callback

function `uniswapV3SwapCallback` with transfers the swapped tokens to the caller of the function.

```
function swapExactUsingPool(address poolAddress, bool zeroForOne, int256 amountIn) public returns (int256, int256) {
    IUniswapV3Pool pool = IUniswapV3Pool(poolAddress);
    return
        pool.swap(
            msg.sender,
            zeroForOne,
            amountIn,
            zeroForOne ? TickMath.MIN_SQRT_RATIO + 1 : TickMath.MAX_SQRT_RATIO - 1,
            abi.encode(poolAddress, pool.token0(), pool.token1(), msg.sender)
        );
}

function uniswapV3SwapCallback(int256 amount0Delta, int256 amount1Delta, bytes calldata data) external override {
    (address poolAddress, address token0, address token1, address userAddress) = abi.decode(data, (address, address, address, address));
    require(msg.sender == address(poolAddress));
    if (amount0Delta > 0) {
        IERC20(token0).transferFrom(
            userAddress,
            msg.sender,
            uint256(amount0Delta)
        );
    }
    if (amount1Delta > 0) {
        IERC20(token1).transferFrom(
            userAddress,
            msg.sender,
            uint256(amount1Delta)
        );
    }
}
```

Borrowing and Repaying Tokens using Aave

To allow for short selling, borrowing and repaying of tokens are required, therefore interacting with Aave in the smart contract is essential. The first step is to instantiate the lending pool using Aave's `PoolAddressesProvider`. When borrowing, some collateral is set aside in order to be able to borrow any number of tokens, thus the first step is to deposit this collateral into the lending pool and allow it to be used as collateral (Lines 1-8). Secondly, the specified token is borrowed from the lending pool using the variable interest rate, specified by the third argument in the `borrow` function, once received, the tokens are sent to the caller's address. The opposite methodology is true for the repayment of a loan, the tokens are transfered from the callers address to the smart contract's address which the sent back to the lending pool by the `repay` function and the collateral is withdrawn and automatically sent to the caller's address.

```
IPool public immutable lendingPool = IPool(IPoolAddressesProvider(0x2f39d218133AFaB8F2B819B1066c7E434Ad94E9e).getPool());

function borrowToken(address tokenAddress, uint256 borrowAmount, uint256 collatoralAmount) public {
    // Deposit Collatoral
    IERC20(wethAddress).transferFrom(msg.sender, address(this), collatoralAmount);
    IERC20(wethAddress).approve(address(lendingPool), collatoralAmount);
    lendingPool.deposit(wethAddress, collatoralAmount, address(this), 0);
    lendingPool.setUserUseReserveAsCollateral(wethAddress, true);
```

```
9
10      // Borrow token
11      lendingPool.borrow(tokenAddress, borrowAmount, 2, 0, address(this));
12      IERC20(tokenAddress).transferFrom(address(this), msg.sender,
        borrowAmount);
13  }
14
15  function repayBorrowedToken(address tokenAddress, uint256 repayAmount,
        uint256 collateralWithdrawAmount) public {
16      IERC20(tokenAddress).transferFrom(msg.sender, address(this),
        repayAmount);
17      IERC20(tokenAddress).approve(address(lendingPool), repayAmount);
18      lendingPool.repay(tokenAddress, repayAmount, 2, address(this));
19      lendingPool.withdraw(wethAddress, collateralWithdrawAmount, msg.sender)
        ;
20  }
```

Opening and Closing of Trading Positions

In addition to these functions, it is know that batching transactions together is more efficient than having to execute them separately, therefore as the strategies being explored for this project require both a buying and selling trade each time, the functions `openBuySellPositions` and `closeBuySellPositions` have been implemented such that it follows the method mentioned in Subsection 3.2.

```
1  function openBuySellPositions(
2      address buyPoolAddress,
3      bool buyZeroForOne,
4      int256 buyAmount,
5      address sellTokenAddress,
6      uint256 sellAmount,
7      uint256 collatoralAmount,
8      address sellPoolAddress,
9      bool sellZeroForOne
10     ) external {
11         swapExactUsingPool(buyPoolAddress, buyZeroForOne, buyAmount);
12         borrowToken(sellTokenAddress, sellAmount, collatoralAmount);
13         swapExactUsingPool(sellPoolAddress, sellZeroForOne, int256(
       sellAmount));
14 }
15
16 function closeBuySellPositions(
17     address buyPoolAddress,
18     bool buyZeroForOne,
19     int256 buyAmount,
20     address sellTokenAddress,
21     uint256 sellAmount,
22     uint256 collatoralAmount,
23     address sellPoolAddress,
24     bool sellZeroForOne
25     ) external {
26         swapExactUsingPool(buyPoolAddress, buyZeroForOne, buyAmount);
27         swapExactUsingPool(sellPoolAddress, sellZeroForOne, int256(
       sellAmount));
28         repayBorrowedToken(sellTokenAddress, sellAmount, collatoralAmount);
29 }
```

4.2.2 State

Storing the state of the strategy is important as the strategies need to store and maintain it's history and vairables to ensure consistency when trading, therefore the state is stored as a dictionary in a pickle file. The State contains the liquidity pool pair that is being traded, the strategy instance, a dictionary of the open positions (initially empty) and the

account balance. The state is updated after every exectution and is stored back into the pickle file.

4.2.3 Retreival of Data and Signal Generation

The frequency at which signals and data is collected is easily variable, however to ensure consistency, the current design has a frequency of every hour. This frequency is set in a Cron job. The Cron job runs a script is to collect the current price data and generate a signal. The data collected is retrieved from the Uniswap GraphQL endpoint:

Fetching Liquidity Pool Price data

```
query ($id: ID!) {
    pool(id: $id) {
        token0 {
            symbol
        }
        token1 {
            symbol
        }
        token0Price
        token1Price
        liquidity
    }
}
```

Fetching Gas Price data

```
query {
    transactions(first:1, orderBy:
    timestamp, orderDirection:desc) {
        gasPrice
        timestamp
    }
}
```

The price data queries the liquidity pool returning it's current prices whereas the gas price data is retrieved from the most recent transaction on Uniswap, regardless of type. These data points are inserted into their corresponding tables in the database, and is also sent to the strategy to generate a signal, which is sent to the execution system for exectution.

4.2.4 Trade Execution

Once the signal has been received, it is sent to the trade exectution system. Once the signal is required to be actualized, the execution system breaks the signal into the different types of orders and executes them. For each type of order, its parameters are extracted, e.g. volume to buy, volume to sell, the target token and the prices in the case of opening a position. These parameters are then used to calculate the remaining paramters that are required to call the corresponding smart contract function, e.g. the swap direction's zeroForOne. Finally, once all of the necessary function parameters has been defined, the order is executed on the blockchain by performing 4 steps. The first is calling the smart contract function, signing the transaction, sending the transaction and waiting for the transaction to complete.

```
# Call the order's corresponding function
call_function = contract.functions.<Smart Contract Function Name>(<Function
     Paramters>).buildTransaction({"chainId": Chain_id, "from": caller, "
    nonce": nonce})
# Sign transaction
signed_tx = web3.eth.account.sign_transaction(call_function, private_key=
    private_key)
# Send transaction
send_tx = web3.eth.send_raw_transaction(signed_tx.rawTransaction)
# Wait for transaction receipt
tx_receipt = web3.eth.wait_for_transaction_receipt(send_tx)
```

After all of the orders have been executed, the new balances of each token along with the updated `open_positions` is returned to then be stored as the new state.

Chapter 5

Trading Strategies

As previously mentioned, the strategy I employ is the mean reversion trading strategy. The basic concept of a mean reversion strategy identifying two closely related assets or securities, typically referred to as a pair, and taking advantage of deviations from their historical price relationship. This works by initially identifying a pair of assets that have a historically stable relationship as I have done in Section 3.5. Then as prices change you calculate the spread, which is the difference between their prices. Look for deviations from the historical mean spread, upon a deviation one buys the undervalued asset and sells the other. The positions are then closed once the spread reverts back to its historical mean.

5.1 Abstract Strategy

The implementation of the generalization of these strategies is intricate and designed to be scalable and highly customizable with various parameters. Additionally, since the trading strategies being investigated are based on mean reversion, an abstract strategy is created to allow for quick exploration and research into different approaches and the impact of hedge ratio calculations on returns. This abstract strategy encompasses the core logic of generating trading signals, determining optimal trade timing and volumes, and transmitting them to the live or backtesting system. The strategy-specific classes contain operations that calculate the hedge ratio and establish the thresholds that trigger the generation of trading signals. Below shows the functions that the abstract strategy possesses.

```
1  class Abstract_Strategy():
2      def __init__(self, number_of_sds_from_mean, window_size_in_seconds,
       percent_to_invest, strategy_name, gas_price_threshold,
       rebalance_threshold_as_percent_of_initial_investment):
3          ...
4
5      def initialise_historical_data(self, history_p1, history_p2):
6          ...
7
8      def recalculate_thresholds(self, has_trade=False):
9          raise NotImplementedError("recalculate_thresholds not implemented")
10
11     def new_tick(self, price_of_pair1, price_of_pair2, has_trade):
12         ...
13
14     def generate_signal(self, ctx, prices):
15         ...
```

The functions `__init__`, `initialise_historical_data`, `new_tick` and `generate_signal` are all inheritted by each strategy and `recalculate_thresholds` is implemented in each instance of the strategy. `__init__`, `initialise_historical_data` are self explanatory. `new_tick` is called in `generate_signal` to update the strategy's knowledge of historical

prices, this function also triggers a call to `recalculate_thresholds` which re-calculates the hedge ratio and determines the thresholds at which an arbitrage opportunity becomes apparent. `generate_signal` is the function that is called by the trading system at each price update, it first invokes `new_tick` which in turn updates the thresholds, finally the updated hedge ratio and thresholds are used in the process of generating a signal. The steps of `generate_signal` are outlined below:

1. The first step is to call `new_tick` which updates the thresholds
2. Second, is to check whether a position is already held;
   (a) If a position IS held, the first is to check if the spread is between the thresholds and if the gas price is below that the specified limit, the positions are closed. However, before this, a precautionary check is made to ensure the balance of any token does not go below 0, if it does, a swap order is placed before the closing of the positions such than the balance of each token remains positive. Otherwise, the loan to value health factor is calculated, if it may cause liquidation, additional collateral is deposited in Aave.
   (b) However, if a position is NOT held and the gas price is below that the specified limit, there are 3 cases.
      i. $spread > upper\_threshold$ - Buy pair 2 and sell pair 1
      ii. $spread > lower\_threshold$ - Buy pair 1 and sell pair 2
      iii. No trade

      Furthermore, if a trade is ordered, it is also checked, if the WETH balance falls below the rebalance threshold, the additional tokens are converted back to WETH.

   In addition, prior to opening or closing a position, a preliminary check is conducted to ensure that the ETH balance remains positive. If the anticipated balance after executing the orders indicates a potential shortfall, an order to `BUY ETH` is automatically initiated. This precautionary measure ensures that the strategy maintains a positive ETH balance and avoids entering into positions that could lead to negative balances.

5.1.1 Volume of Trades

A major part of the strategy is at which volumes to trade, therefore in order to be able to trade the maximum amount a minimization problem must be solved. However, before delving into thie problem it is important to highlight the influence of the hedge ratio of the trading volumes. If the hedge ratio is positive, for every unit of pair0 traded, `hedge_ratio` units of pair1 should be traded. However, the inverse is true if the hedge ratio is negative, for every `-hedge_ratio` units of pair0 traded, 1 unit of pair1 should be traded.

Given that Z is amount of WETH in the account, p_0, p_1 are the prices of pairs 0 and 1 respectively, LTV is the loan-to-value ratio and x, y are the volumes to be traded on pair0 and pair1 respectively. Additionally, to generalize the problem for both positive and negative hedge ratios, let $n : k$ represent the ratio of volume traded for pair0:pair1. Assuming we are buying pair0 and selling pair1. The following problem states finds the

maximum volumes that can be traded to leave the least amount of WETH remaining.

$$\begin{aligned} \min_{x,y} \quad & Z - xp_0 - \frac{yp_1}{LTV} \\ \text{s.t.} \quad & x = \frac{k}{n}y \\ & x, y \geq 0 \\ & p_0, p_1 \geq 0 \\ & Z \geq 0 \\ & 0 \leq LTV \leq 1 \end{aligned} \tag{5.1}$$

Therefore, as all terms in the objective function are positive therefore, the minimal objective value in theory must be 0:

$$\begin{aligned} Z - xp_0 - \frac{yp_1}{LTV} &= 0 \\ Z - \frac{k}{n}yp_0 - \frac{yp_1}{LTV} &= 0 \\ Z - (\frac{k}{n}p_0 + \frac{p_1}{LTV})y &= 0 \\ y &= \frac{Z}{\frac{k}{n}p_0 + \frac{p_1}{LTV}} \\ x &= \frac{k}{n}\frac{Z}{\frac{k}{n}p_0 + \frac{p_1}{LTV}} \end{aligned}$$

Thus the resulting values of x and y represent the solution that minimizes the problem while maximizing the tradable volume. The case above shows that x units of pair0 can be bought and thus y units of pair1 must be sold. Similarly, if pair0 is sold and pair1 is bought, the maximum volumes are determined by $x = \frac{Z}{\frac{p_0}{LTV} + \frac{n}{k}p_1}$ and $y = \frac{n}{k}\frac{Z}{\frac{p_0}{LTV} + \frac{n}{k}p_1}$.

5.2 Constant Hedge Ratio Strategy

In the most simplistic mean reversion strategy, the approach relies on a given historical dataset, assuming that the hedge ratio between the paired assets remains consistent over the long term. To determine this hedge ratio, the Ordinary Least Squares (OLS) regression method is employed.

```
model = sm.OLS(history_p1, sm.add_constant(history_p2))
results = model.fit()
# Gradient of the OLS i.e. X = results.params[0] + results.params[1] * '
    p2_token1_price'
hedge_ratio = params[1]
```

In Figure 5.1, the observed trend reveals that the gradient, representing the rate of change, exhibits a proximity to the value of 1. This indicates a relatively balanced relationship between the prices of each pair. However, it is important to note that as time progresses, any fluctuations or deviations from the exact value of 1 are minimal and have a negligible impact on the overall trend. The stability of the gradient over time suggests a consistent and relatively stable relationship between the liquidity pools, reinforcing the notion that their interdependence remains relatively constant.

5.3 Sliding Window OLS Strategy

As mentioned earlier, the hedge ratio plays a crucial role in minimizing risk and achieving a market-neutral position in trading strategies. However, employing a static hedge ratio

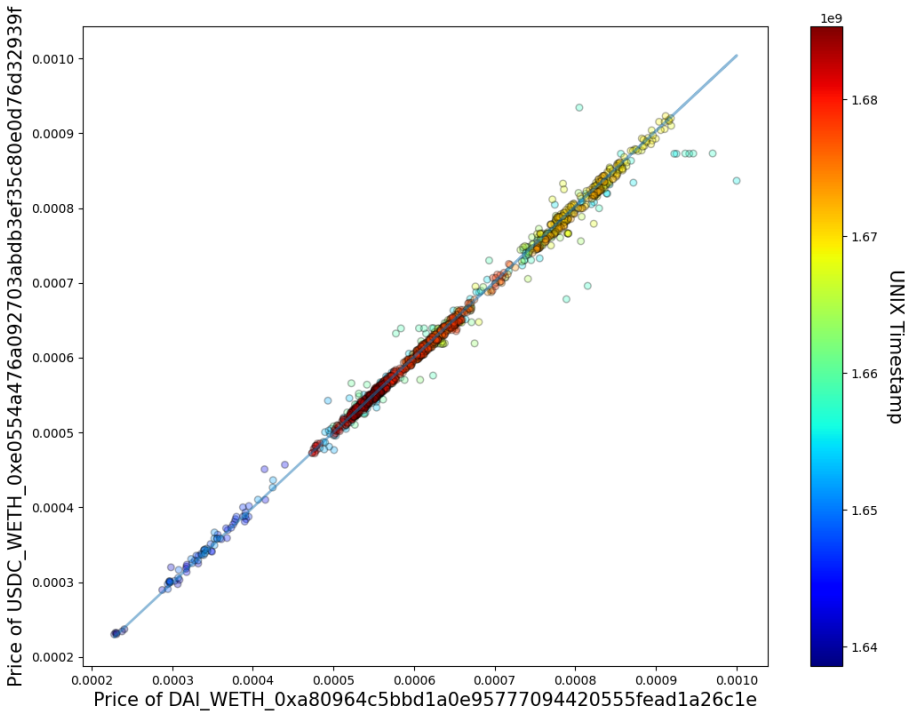

Figure 5.1: Conducting OLS to obtain the Hedge Ratio

may prove suboptimal since market conditions and the correlation between paired assets can evolve over time. To tackle this challenge, a dynamic approach is adopted using a sliding window approach. In this approach, the hedge ratio is continuously updated at each tick by considering the most recent data. The length of the data window used for the calculation is determined by the value specified in the `window_size` argument. By incorporating the sliding window mechanism, the hedge ratio remains adaptive to the changing market dynamics, providing a more accurate and responsive estimation for effective risk management and trading decisions. Therefore, the following function is ran every price tick,

```
def update_hedge_ratio(self):
    p1, p2 = self.history_p1[-window_size:], self.history_p2[-window_size:]
    model = sm.OLS(p2, sm.add_constant(p1))
    results = model.fit()
    self.hedge_ratio = results.params[1]
```

5.4 Lagged OLS Strategy

A lagged strategy was also a implemented as an extension of the sliding window strategy. In the context of the sliding window strategy, the hedge ratio is updated using the most recent data within a specified window size. However, due to potential delays or lags in the synchronization of liquidity pools, the instantaneous prices of the assets may not fully reflect their actual relationship. Therefore, this approach aims to account for potential lags in the synchronization of liquidity pools. The lagged strategy extends the existing sliding window approach by using `sm.tsa.lagmat` function to perform the lag, which is then used to perform OLS.

```
def update_hedge_ratio(self):
    maxlag = min(self.lag, min(self.window_size_in_hours, len(self.
    history_p1))-1)
    p1, p2 = self.history_p1[-(self.window_size_in_hours+maxlag):], self.
    history_p2[-self.window_size_in_hours:]
    lagged_p1 = sm.tsa.lagmat(p1, maxlag=maxlag)[-self.window_size_in_hours
    :]
```

```
    model = sm.OLS(p2, sm.add_constant(lagged_p1))
    results = model.fit()
    self.hedge_ratio = results.params[1]
```

5.5 Unrestricted Model from the Granger Causality Test

Similarly, to account for lag and causality, a strategy that was employed was using the Granger Causality Test. The Granger Causality Test is a statistical hypothesis test that assesses the predictive value of one time series in forecasting another. Part of the test the results provide insights into the relationship between the variables and help determine the presence of Granger causality in the form of an OLS regression for both the restricted and unrestricted models. The restricted model is a regression model that includes only the lagged values of the dependent variable whereas the unrestricted model includes both the lagged values of the dependent variable and the potentially causal variable. The restricted model's regression parameters are used to calculate the hedge ratio.

```
def update_hedge_ratio(self):
    p1, p2 = self.history_p1, self.history_p2
    data = pd.DataFrame({'Asset1': p1, 'Asset2': p2})
    granger_results = statsmodels.tsa.stattools.grangercausalitytests(data,
     maxlag=[1], verbose=False)
    self.hedge_ratio = granger_results[1][1][1].params[0]
```

5.6 Kalman Filter Strategy

Another method that is often used to ensure market-neutral positions is the use of Kalman Filters as a dynamic tool to update and adapt the hedge ratio, taking into account the changing market conditions and the evolving relationship between the paired assets. This allows for a more responsive and adaptive approach to maintaining a market-neutral position. The Kalman Filter takes into consideration not only the current price data but also the historical information and the underlying dynamics of the assets. It provides a more robust and accurate estimation of the optimal hedge ratio, which aligns with the mean reversion strategy's objective of capturing potential price divergences and profiting from their eventual convergence.

The Kalman Filter works by iteratively updating and refining its estimate of the system state using a prediction step and an update step. For this an initial 'guess' of the parameters that I use is the using the Ordinary Least Squares from the historical information provided. The remainder of the parameters that are selected can be seen below:

```
model = sm.OLS(price_history_2, sm.add_constant(price_history_1))
initial_state = model.fit().params[::-1]

kf = KalmanFilter(
    n_dim_state=2,
    initial_state_mean=initial_state,
    transition_matrices=np.eye(2),
    observation_matrices=obs_mat,
    transition_covariance=1e-5 * np.eye(2)
    )
```

Below describes the reasoning behind the choices of parameters:

- `n_dim_state` - This parameter to the number of elements in the state. In our scenario, the state consists of two components: the y-intercept and the gradient.

- `initial_state_mean` - As mentioned earlier, the Kalman Filter utilizes past estimates to iteratively estimate the true states. Therefore, I employ the OLS method to calculate the initial state of the price history's regression, which returns both the y-intercept and gradient.

- `observation_matrices` - The observation matrix defines the relationship between the observed measurements and the hidden state variables. Thus takes the historical data that is provided zipped along with 1 as the observed measurements directly correspond to the hedge ratio. This can be represented as $\begin{bmatrix} y_1 & 1 \\ y_1 & 1 \\ \vdots & 1 \\ y_n & 1 \end{bmatrix}$.

- `transition_matrices` - In the context of our strategy, we anticipate a consistent long-term hedge ratio. Therefore, we set the `transition_matrices` parameter to $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$.

- `transition_covariance` - Finally the transition_covariance specifies the covariance matrix of the process noise. It represents the uncertainty or variability in the state transition, hence it is set to be very small, $\begin{bmatrix} 10^{-5} & 0 \\ 0 & 10^{-5} \end{bmatrix}$.

Figures 5.2 and 5.3 provide visual representations of the hedge ratio's evolution over time. These figures illustrate that the hedge ratio, while exhibiting subtle variations, does indeed change over the observed period. One notable event is the significant drop that occurs at the timestamp 1.655e9 or June 2022. This particular shift in the hedge ratio is likely attributed to the migration from the Proof of Work (PoW) consensus mechanism to the Proof of Stake (PoS) consensus mechanism on the Ethereum network.

During the transition from PoW to PoS, there are fundamental changes in how the Ethereum network reaches consensus and validates transactions. This change in consensus mechanism can impact various aspects of the network, including the dynamics of the hedge ratio. It is plausible that the observed drop in the hedge ratio during the migration period reflects the adjustments and reconfiguration of the underlying mechanisms in response to the shift to PoS.

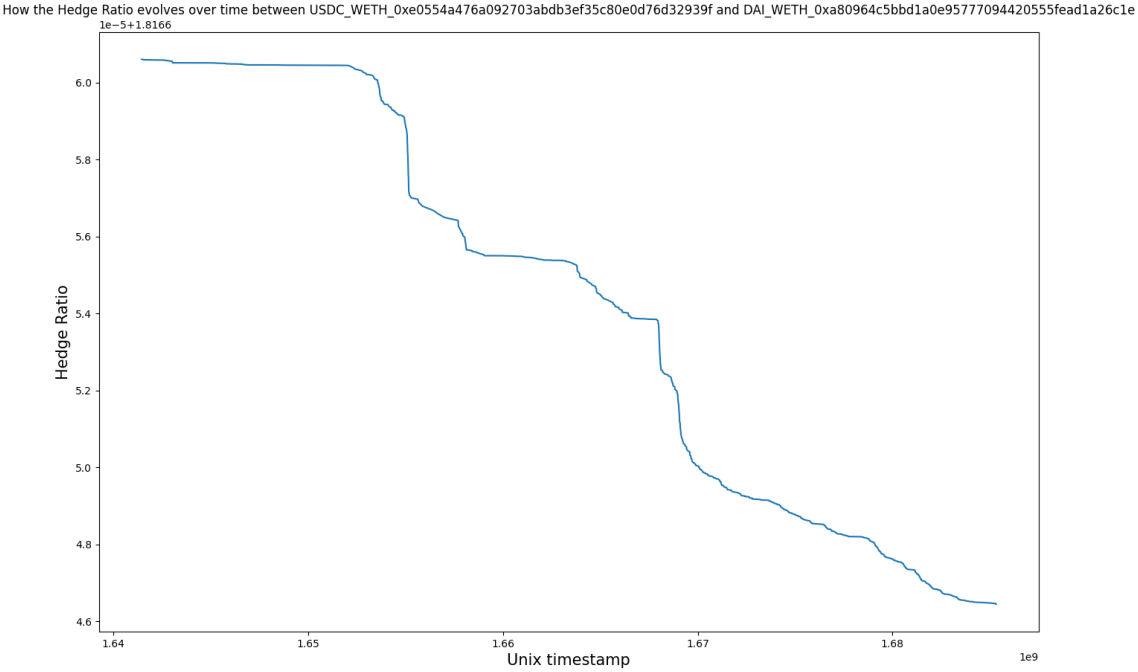

Figure 5.2: Plot of how the Hedge Ratio using the Kalman Filter

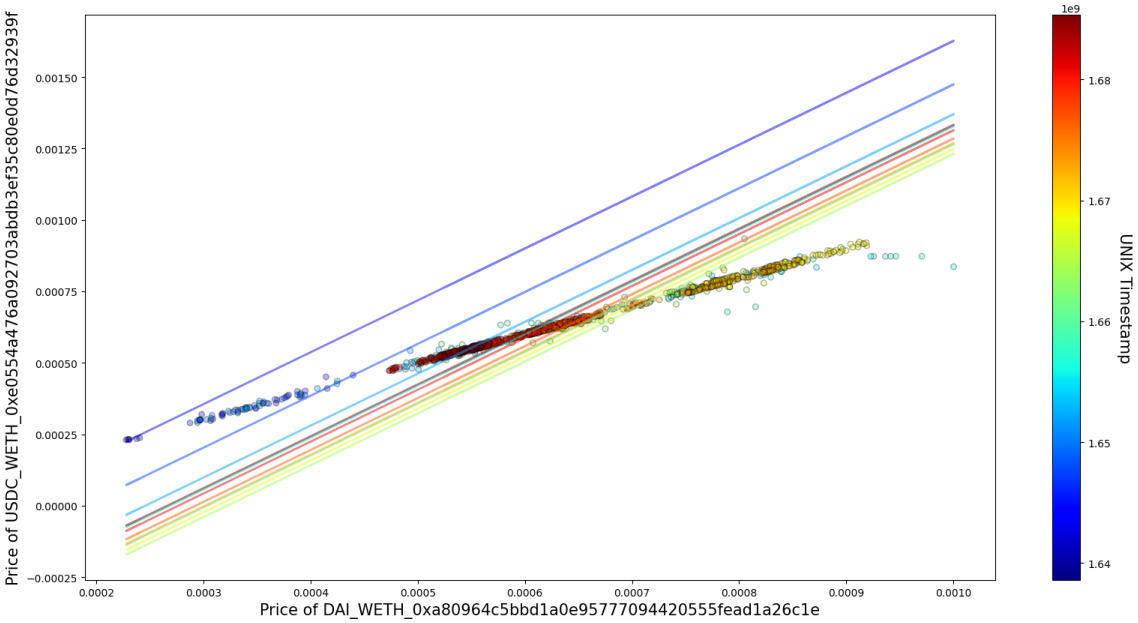

Figure 5.3: The plot shows the Kalman Filter estimating the regression line over time,

Chapter 6

Evaluation

The design of each strategy and the backtesting system allow for extensive parameterization, providing multiple axes for analysis. Parameters that can be adjusted are at what number of standard deviations away from the mean spread should a threshold set and a signal to be generated, how the trades are executed, limits on how much gas price should be when executing a trade, and also the volume of the initial investment. These are all factors that can effect the returns of the trading strategy. In the following sections, we will examine the performance of the strategy by varying these parameters, providing valuable insights into their influence on trading outcomes. For the purpose of data visualisation we denote the liquidity pool pairs by their index:

Pool1	Pool2	Label
USDC_WETH_0x88e6a0c2ddd26feeb64f...	USDC_WETH_0xe0554a476a092703abd...	0
USDC_WETH_0x8ad599c3a0ff1de08201...	USDC_WETH_0xe0554a476a092703abd...	1
WETH_USDT_0x11b815efb8f581194ae7...	USDC_WETH_0xe0554a476a092703abd...	2
WETH_USDT_0x4e68ccd3e89f51c3074c...	USDC_WETH_0xe0554a476a092703abd...	3
DAI_WETH_0x60594a405d53811d3bc47...	USDC_WETH_0xe0554a476a092703abd...	4
DAI_WETH_0xc2e9f25be6257c210d7adf...	USDC_WETH_0xe0554a476a092703abd...	5
USDC_WETH_0xe0554a476a092703abd...	WETH_USDT_0xc5af84701f98fa483ece7...	6

Table 6.1: Liquidity pool pairs and their corresponding labels

6.1 Trading Systems

6.2 Strategy Parameters

6.2.1 Number of Standard Deviations away from the Mean

Varying the standard deviation in a mean reversion pairs trading strategy can have a significant impact on the strategy's returns. When the number of standard deviations is set too high, it means that the spread between the pair's prices must deviate significantly from the mean before a signal is generated. In this case, the strategy may generate fewer trades, resulting in longer holding periods for positions, however this tends to have a greater return per trade. On the other hand, setting a lower number of standard deviations makes the strategy more sensitive to smaller deviations from the mean. This increases the frequency of trade signals and potentially leads to more active trading. However, it also exposes the strategy to a higher risk of false signals or trading on noise in the price data. It can result in higher transaction costs and potentially lower returns due to increased trading activity, potentially resulting in losses. For the purpose of simplicity, $\#_\sigma$ denotes this parameter.

Table 6.2 displays the returns as the standard deviations away from the mean the threshold is set. For the purpose of this experiment the other parameters were; 100 ETH worth of

initial investment, the window size set to 30 days, a gas price threshold of 1 ETH (i.e. not having a threshold), the interactions with the blockchain to be seperately executed and finally the lag that the lagged strategy uses is set to 1 hour. The period that trading occurs is from 18th December 2021 to 9th June 2023.

We can see that as $\#_\sigma \to 0$, the number of trades increases however causes the returns to be more negative (and in some cases terminating early with 100% loss) than positive, likely due to trades being opened when gas fees are substantially larger than the profit. However, if $\#_\sigma$ is larger, although the profits from each trade increases, the number of opportunities that were actionable were limited. Another interesting feature that can easily be spotted is that the Constant Hedge Ratio Strategy makes little to no profit in all cases whereas the other strategies perform better as $\#_\sigma \to\sim 1.75$. Furthermore, all of the strategies return a profit on pool pair 6 with the exception of the Kalman Filter Strategy, albeit with a higher threshold.

Consistent profit among all of the pool pairs, excluding pool pair 6, is also achieved at different number of standard deviations. We can see in Figure 6.1 that the average return across all liquidity pool pairs that each strategy generates at each standard deviation. The Kalman Filter becomes largely profitable at $\#_\sigma = 0.5$, whereas the lagged and sliding window strategies become profitable at $\#_\sigma = 2$. The reason for this is is highly due to the fact that the Kalman Filter and the Granger Causality tests are better able to find underlying trends thus being better at selecting the hedge ratio, which in turn effect the volumes being traded and hence the effect the returns.

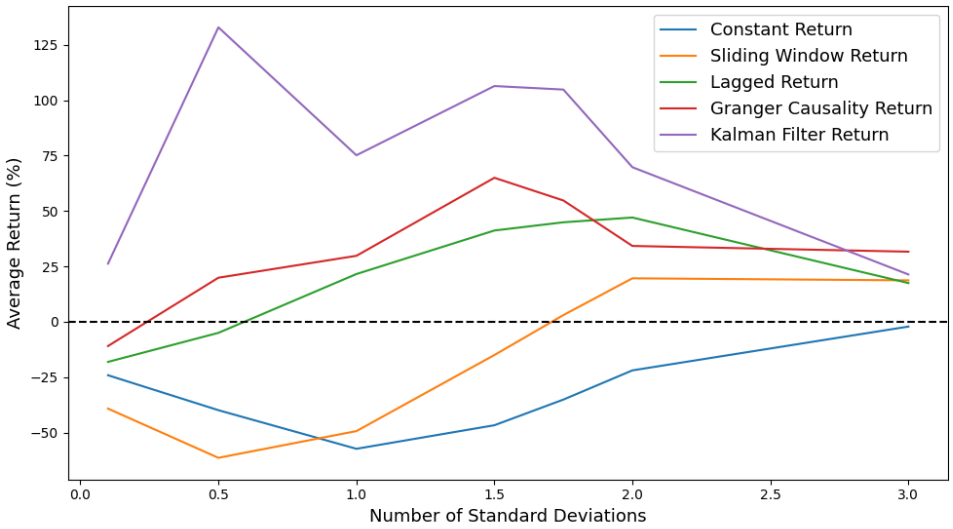

Figure 6.1: Average Return across all pool pairs using various standard deviations

6.2.2 Gas Fees

Another aspect that could be changed is fees; the first is how the transactions are executed on the blockchain and also the maximum gas price if we want to open a position. As previously mentioned, the gas fee is calculated by a combination of the gas price and the gas used. Therefore, these 2 parameters allow the strategy control both the gas used and gas price.

Batched Transactions vs Seperately Executed Transactions

The method in which the transactions are executed effects how much gas is used. When transactions are executed separately, each transaction incurs its own gas cost. This means that for every individual transaction, there will be gas consumed for various operations such as contract interaction, data storage, and computational tasks. As a result, executing a large number of separate transactions can lead to a significant cumulative gas cost. On the other hand, when transactions are batched together, they are consolidated into a single transaction. This consolidation reduces the overhead associated with executing multiple transactions individually. By combining the operations of multiple transactions into a single execution, redundant computations and storage operations can be eliminated, resulting in more efficient gas usage. It can immediately be seen in Figure 4.3 that the process of batching the transactions consumes less gas than the combined gas of the transactions required to exectue the openning and closing of the positions.

Furthermore, the effects of using seperate versus batched transactions can be seen in Table 6.3. The parameters used for this investigation were the same as above with the selection eg $\sigma = 1.75$, 100 ETH worth of initial investment, the window size set to 30 days, a gas price threshold of 1 ETH (i.e. not having a threshold) and the testing period remained from 18th December 2021 to 9th June 2023.

	Pool Pair	Strategy's Annual Percentage Rate (APR) - Trading from 18th December 2021 to 9th June 2023									
		Constant		Sliding Window		Lagged		Granger Causality		Kalman Filter	
		Return %	# of Trades	Return %	# of Trades	Return %	# of Trades	Return %	# of Trades	Return %	# of Trades
Seperate	0	-7.14	288	24.57	342	81.97	353	61.58	192	98.76	153
	1	-52.98	358	-20.9	293	-5.29	323	12.99	213	45.66	178
	2	-1.71	225	21.43	277	68.39	270	68.62	180	95.11	133
	3	-45.93	309	-14.98	268	-1.01	294	22.19	211	53.56	145
	4	-2.54	241	26.46	294	55.77	291	65.92	201	84.62	152
	5	-41.71	287	-9.73	225	9.79	250	20.48	200	61.97	152
	6	-35.28	76	-19.16	67	-29.96	76	-23.36	37	-16.02	74
Batched	0	-2.2	288	30.8	342	87.79	353	66.76	192	100.13	153
	1	-47.23	358	-16.28	293	0.91	323	16.58	213	46.78	178
	2	2.63	225	28.29	277	74.07	270	75.52	180	97.43	133
	3	-42.06	309	-9.95	268	4.71	294	25.33	211	52.93	145
	4	1.72	241	29.97	294	62.84	291	69.21	201	86.18	152
	5	-37.04	287	-4.83	225	14.37	250	23.51	200	62.31	152
	6	-33.66	76	-17.93	67	-27.78	76	-21.93	37	-14.77	74

Table 6.3: Returns of each strategy when using seperate and batched transactions

.

When comparing the use of batched transactions to executing transactions separately, it is evident that utilizing batched transactions yields a higher return, as expected. The observed increase in return ranges from approximately 1% to 8% compared to executing the transactions individually. The higher return achieved through batched transactions can be attributed to several factors. First, as mentioned earlier, batching transactions reduces the gas costs associated with executing multiple transactions. By consolidating operations into a single transaction, redundant computations and storage operations are eliminated, resulting in more efficient gas usage. The reduction in gas costs directly contributes to the overall profitability of the trading strategy.

Gas Price Threshold

A threshold on the gas price is also set to ensure positions that have a high transaction cost associated with it are avoided. As the gas price is variable, we can see in Figure 6.2 the distribution of the gas prices over time is similar to a geometric distribution therefore by thresholding high gas prices, the strategies would avoid making a trade where the trade's profit would be overshadowed by the cost of trading.

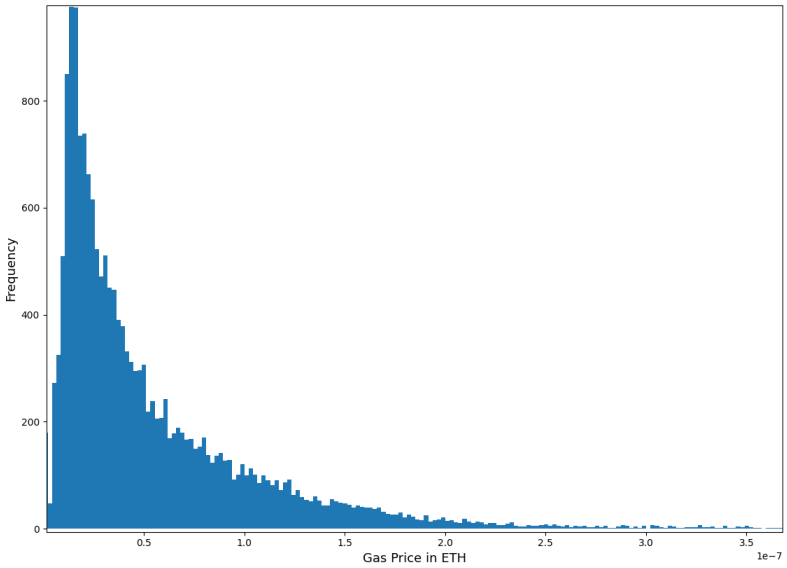

Figure 6.2: Histogram Plot of Gas Prices in ETH

Therefore, our investigation consists of varying this threshold to see its effects on the returns of each strategy. Using batched transactions and the same parameters used in the aboce experiment, the thresholds are set to the $60^{th}, 70^{th}, 80^{th}, 90^{th}, 100^{th}$ quantiles. Table 6.4 displays the results obtained by varying these thresholds. It can be seen that by filtering more costly opportunities, expectedly reduces the number of actionable, however the higher thresholds also show that these opportunities are still just as if not more profitable. In addition, to this Figure 6.3, shows that a low threshold benefits constant hedge ratio strategy most as with the low threshold, it has a light profit and as the threshold increases the return using the strategy decreases. Whereas, the returns are hindered with a lower threshold as many profitable opportunities are missed due to the overly low threshold, resulting in a reduction in the overall profitability of the trading strategy. The figure also shows how setting the threshold after the 90^{th} percentile does little to no effect in the return, giving a slight positive impact on the Lagged strategy and a slight negative impact on the the remaining. This is likely due to there only a small amounts of arbitrage opportunities to be present in such times when the gas price is high, thus having little impact. It is also interesting to see that the Lagged and the Kalman Filter strategies perform better by setting the threshold to the 90^{th} percentile in comparison to the 80^{th} percentile, whereas the other strategies perform worse.

Overall, the analysis of varying gas price thresholds reveals that adjusting the threshold level has a direct impact on the number of actionable opportunities and the resulting returns. Higher thresholds filter out less favorable opportunities, while still maintaining profitability. However, it is crucial to strike a balance and avoid setting the threshold too low, as this may lead to missed profitable opportunities. Hence, to get the best balance the threshold is set to the 90^{th} percentile.

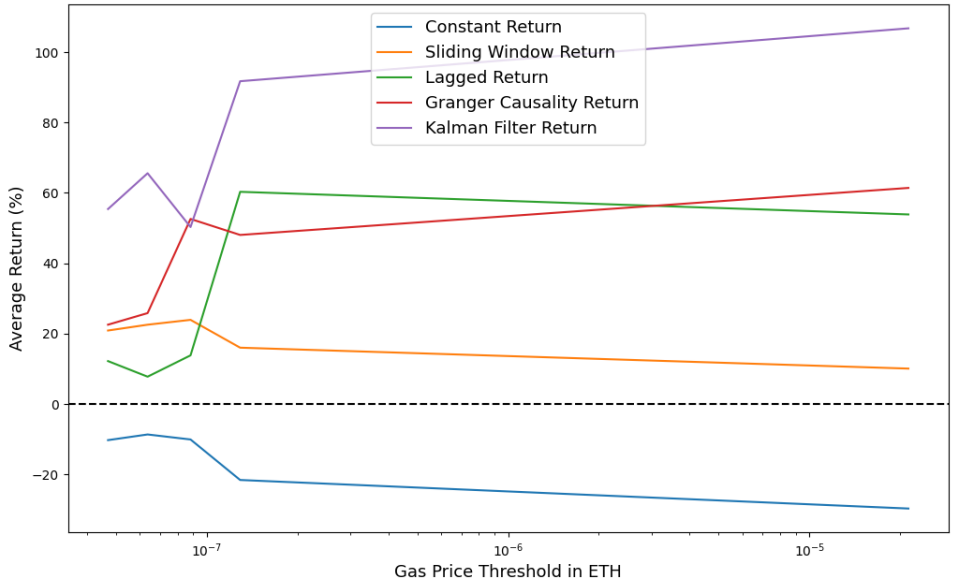

Figure 6.3: Average Returns of each strategy when using different gas price thresholds

6.2.3 Initial Investment Volume

Another import factor that contributes to the profit is the initial investment therefore Table 6.5 and Figure 6.4 displays the returns obtained by various initial investments. It can immediately be seen that the higher the investment, the greater the return. However, the smallest investments fall to a loss immediately for all strategies, this is because the fee that are incurred by gas fees are fixed regardless of the volume whereas the costs incurred by Aave and Uniswap are proportional to the volume being traded. This fixed fee means that a greater volume is required to cover the cost of the gas fee in order to generate profit, which is of course proportional to the volume being traded. Hence, as the investment increases the return increases however, begins to platau after 25ETH as the fixed fee becomes negligible in comparison to the investment.

Upon comparing the different strategies, the Kalman Filter is able to generate a profit with least amount of investment, followed by the Granger Causality strategy, then, the Lagged and Sliding Window strategies and finally the Constant strategy fails to make any money with any investment. Interestingly, although the Granger Causality test strategy is able to churn a better return at lower initial investments, when initial investment ≤ 20, the Lagged strategy overtakes the returns of the Granger Causality strategy once returns hit 0%. The Lagged strategy plataus to around 60% whereas the Granger Causality strategy plataus to 50%, indicating that although both of these methods assesses causality of the price dynamic between the two liquidity pairs, the lagged strategy estimates the hedge ratio better than the unrestricted model generated from the Granger Causality test.

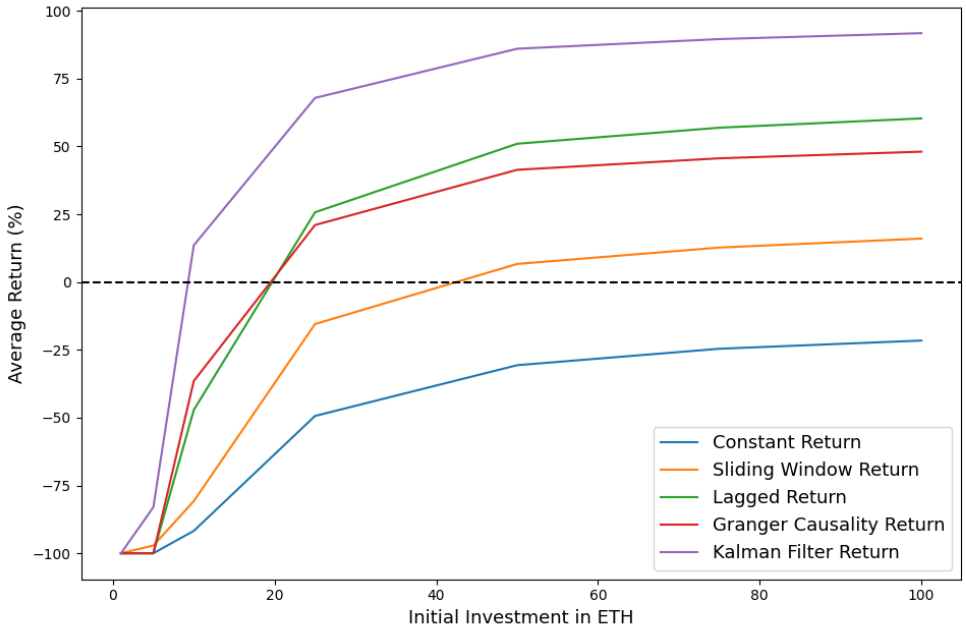

Figure 6.4: Average Returns of each strategy using different initial investment volumes

6.2.4 Window Size

The final parameter that also has an effect on the returns is the window size. This window size is used when calculating the mean and standard deviation that are used to calculate the thresholds. As we can see in the excerpt below, the greater the window size, the more data the strategy would use to calculate the thresholds.

```
spread = self.history_p1[-self.window_size_in_hours:] - self.hedge_ratio *
    self.history_p2[-self.window_size_in_hours:]
spread_mean = spread.mean()
spread_std = spread.std()
self.upper_threshold = spread_mean + self.number_of_sds_from_mean *
    spread_std
self.lower_threshold = spread_mean - self.number_of_sds_from_mean *
    spread_std
```

To evaluate how the window size effects the returns, backtesting is conducted over the period 9th June 2022 to the 9th June 2023, exactly 1 year. The results can be found in Table 6.6 and Figure 6.5 where we can see that, as the window sizes increase, the returns increase however for the smaller window sizes, the return are more eratic and volatile. This volatility decreases as the window size is increases platauing to each strategies respective returns.

It is also interesting to see that in this period of trading the Kalman Filter performs the worst which is surprising as in has outperformed each of the strategies in the experiment that have been previously mentioned.

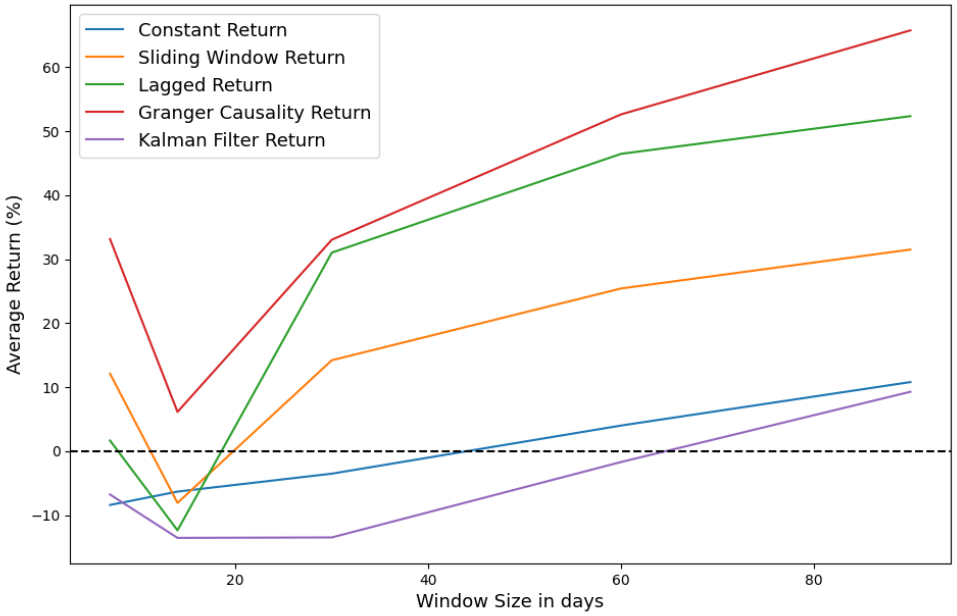

Figure 6.5: Average Returns of each strategy using different Window Sizes

6.3 Evaluation of Strategies

Overall, given the experiments conducted the optimal paramters for each strategy are different and thus to evaluate each strategy, Table 6.7 displays the optimal parameters used for each strategy.

Strategy	Number of Standard Deviations from the mean, $\#_\sigma$	Gas Price Threshold in ETH	Window Size in Days	Type of Execution, Seperate or Batched
Constant	3	6.36e-08	60	Batched
Sliding Window	2	8.83e-08	60	Batched
Lagged	2	1.29e-07	60	Batched
Granger Causality	1.5	2.13e-05	60	Batched
Kalman Filter	0.5	2.13e-05	60	Batched

Table 6.7: Optimal Parameters for each strategy

By running the backtesting system from 14th January 2022 to 9th June 2023 with an initial investment of 100ETH, we can see in Table 6.8 the Kalman Filter outperforms all of the other strategies in all liquidity pool pairs except pair 6. The Granger Causality strategy has also performed well, whereas the Lagged strategy returned mediocre a return, with the Constant and the Sliding Window strategies strategy performing similarly, both resulting in around 20% APY.

In addition to this, Figure 6.6 shows the account value over time of each of the strategies, it is easy the see that the account values seem to follow a trend however, the gradients of the return are less drastic in some strategies i.e. the Kalman Filter. It is also notable

that the number of profitable trades and decreased since the beginning of 2023, with the Kalman Filter strategy taking a large loss in January of this year, 2023.

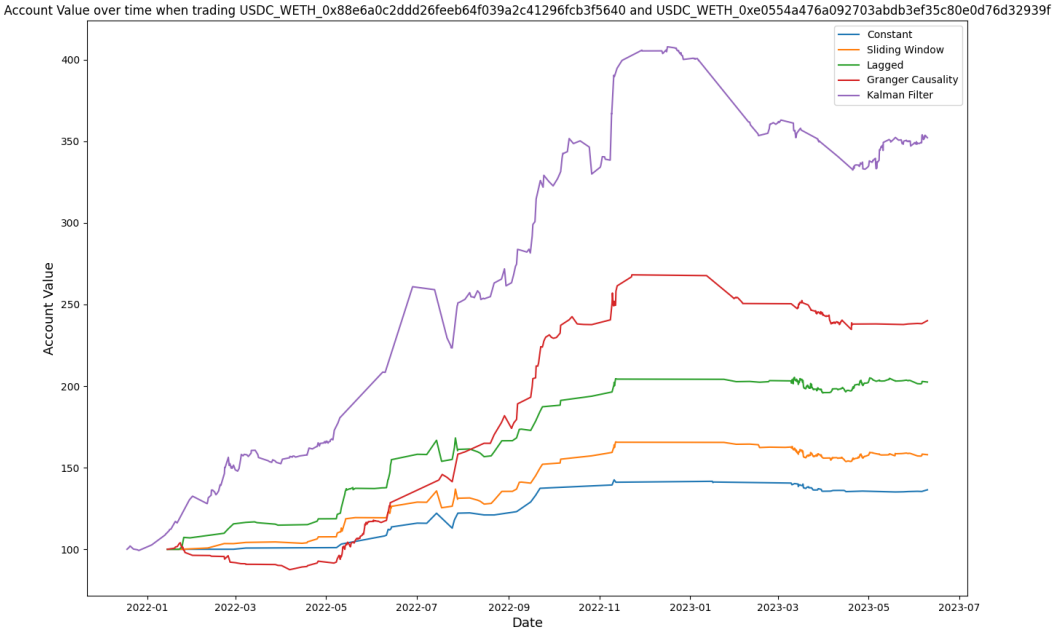

Figure 6.6: Account Value over Time from backtesting Pair 0

Another valuable insight can be seen in Figure 6.7 where the evolution of the hedge ratio is plotted for each strategy. It can be seen that the sliding window and the granger causality strategies' hedge ratio is fairly consistent in comparison to the lagged strategy's hedge ratio. However, the Kalman Filter calculates the Hedge Ratio slightly lower and also more constant with only minor deviations, as can be seen in Figure 5.2. This indicates a more persistent relationship between the paired assets. The minor deviations observed in the hedge ratio can be attributed to temporary fluctuations in market dynamics or noise in the data.

Figure 6.7: The Hedge Ratio over time for Pair 0

6.3.1 Beta

An important metric when evaluating trading strategies or investments that is used in industry is Beta, β. It measures how correlated the strategy's return is with the market return rate. It is often used as a risk-reward measure, allowing investor to make better decisions when investing, balancing the risk and reward. β is defined as $\beta = \frac{Cov(R_M, R_S)}{Var(R_M)}$, where R_M is the market's returns and R_S is the strategy's or stock's returns, the higher the β value the more correlated the strategy's return is with the market whereas a lower one, i.e. $\to -\infty$, the strategy swings the opposing direction to the market. Therefore, in risk neutral strategies, a low absolute value is ideal as it minimizes the risk.

Therefore, to analyse the β of each strategy, the conversion from WETH to USD is used as the strategies use WETH as their base currency. As we can see in Table 6.9, the βs are very close to 0 indicating that the strategy's dependence on the market returns is very low. The low β values imply that the trading strategy may be designed to generate returns based on factors other than general market movements. It suggests that the strategy relies more on specific signals, indicators, or market inefficiencies rather than the broader market conditions. This can be advantageous in certain situations, as it may allow the strategy to perform well even when the overall market is experiencing volatility or downturns.

The comparison of account values and the evolution of ETH price over time are depicted in Figure 6.8. The figure illustrates that there is a limited correlation between the returns of the trading strategy and the price of ETH. While the returns of the strategy do not closely follow the exact movements in the ETH price, it is evident that significant price fluctuations in ETH do have an impact on the strategy's performance. This observation suggests that the trading strategy's returns are influenced, to some extent, by the general trends and movements in the price of ETH. However, it is important to note that the relationship is not entirely linear or directly proportional. Analyzing the correlation between the strategy's returns and the price of ETH provides valuable insights into the dynamics of the trading strategy and its sensitivity to market movements. By understanding the impact of ETH price changes on the strategy's performance, traders and investors can make informed

decisions and potentially adjust their strategies to optimize their returns and manage risks effectively. It can also be seen that the Constant, Sliding Window and Lagged strategies are less effected by the market compared to the Granger Causality and Kalman Filter strategies.

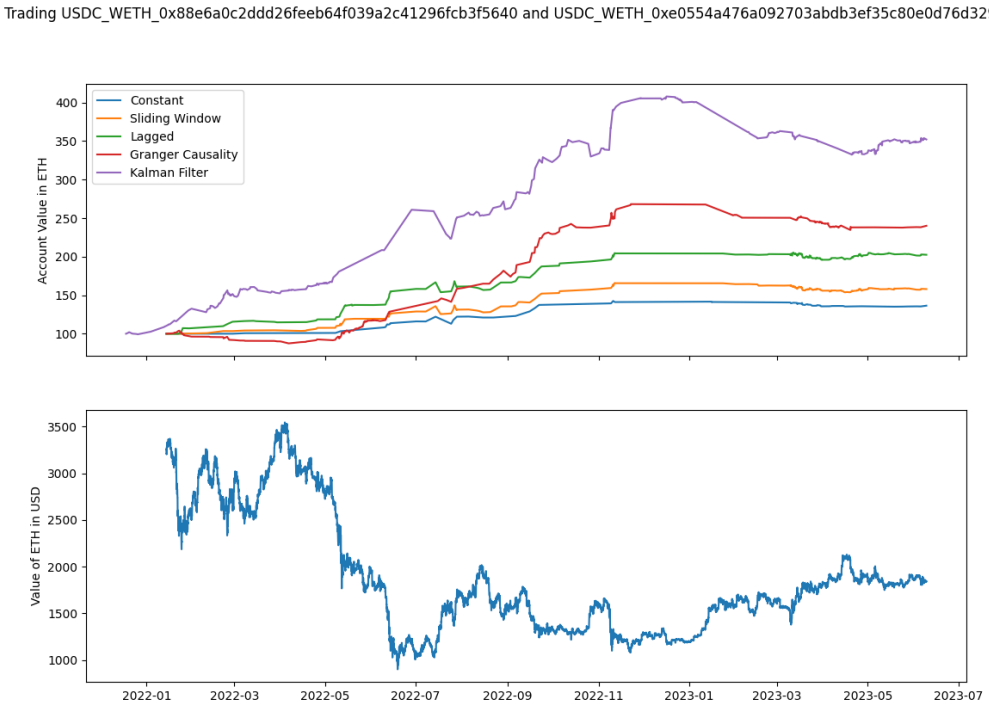

Figure 6.8: Account Value of time with the price of ETH

6.3.2 Volatility & Sharpe Ratio

Another metric that is commonly used in finance is the Sharpe Ratio, the reason for this is that it is important to evaluate how risky the strategies are compared to the risk free rate. The metric is given by the formula below:

$$\text{Sharpe Ratio} = \frac{R_p - R_f}{\sigma_p}$$

Where R_p is the return of a portfolio, R_f is the risk-free rate and σ_p is the standard deviation of the portfolio. A higher Sharpe ratio indicates a better risk-adjusted return, as it signifies a higher return relative to the amount of risk taken. It implies that the investment has achieved a greater excess return compared to the risk-free rate, considering the level of volatility. Conversely, a lower Sharpe ratio suggests a lower risk-adjusted return, indicating a relatively higher level of risk for the given return. The Sharpe ratio also helps investors compare different investments or strategies based on their risk adversity performance. It provides a way to assess whether the return generated by an investment adequately compensates for the level of risk taken.

To calculate each of the strategies Sharpe Ratios, the Bank of England's interest rate of 4.5% [53]. Table 6.10 displays the sharpe ratios of each strategy, we see that the Constant Strategy exhibits the highest hedge ratio, followed by the Lagged and Granger Causality strategies, while the Sliding Window and Kalman Filter strategies have the lowest Sharpe ratios. This discrepancy can be attributed to the Kalman Filter strategy having a higher variance, indicating a greater level of risk compared to the Sliding Window, Lagged, and Granger Causality strategies. According to Forbes, a Sharpe ratio > 1 is considerer to

be 'good' as it suggests that the investment or strategy has generated a return that is more than one unit of risk. Overall, all strategies yield desirable Sharpe ratios, indicating favourable risk-adjusted returns.

6.3.3 Comparison with Previous results

Overall, the strategies seem to be a desirable investment for those with enough capital however how does these invests stand against other techniques used. There has not been any research into statistical arbitrage on decentralized exchanges, with the few that have looked into trading in DEXes evaluating pure arbitrage, searching for miss-pricing and executing them. One of which found there to be little to no profitable arbitrage opportunities on various decentralised exchanges, including Uniswap, regardless of volume size [54]. This happens as the pair prices as automatically adjusted based on the supply and demand, making the opportunities limited, short lived and costly. In comparing, the results from the strategies implemented in this paper do generate a profit in comparison to the cyclic arbitrage approach.

Other research into statistical approaches to arbitrage have been done on more mainstream assets such as ETFs however, they have not been experimented on cryptocurrencies. One paper used the Kalman Filter on ETFs and ETNs and found there to be an average of 25.17% returns in 1 year in insample backtesting, however failed to generate any profit in its out of sample results [39]. The implemented Lagged, Kalman Filter and Granger Causality trading strategies yeild a greater APR compared to this. In addition to this, another peice of research used a combination of machine learning and the Kalman Filter on assets on the BM&FBovespa Exchange, yeilding a return of 26.13% in the out of sample results [40]. In comparison, the Lagged, Kalman Filter and the Granger Causality strategies implemented in this paper have a greater return, however the strategy implemented by Oliveira and Nóbrega, possesses a greater Sharpe ratio meaning the risk-adjusted returns are more favourable. To summarize, the two of the five strategies implemented out perform the current research out there on both purer forms of arbitrage, i.e. cyclic arbitrage, and also the use of Kalman Filter on other types of assets, i.e. ETFs and ETNs.

Std	Pool Pair	Strategy's Annual Percentage Rate (APR) - Trading from 18th December 2021 to 9th June 2023									
		Constant		Sliding Window		Lagged		Granger Causality		Kalman Filter	
		Return %	# of Trades	Return %	# of Trades	Return %	# of Trades	Return %	# of Trades	Return %	# of Trades
$\#_\sigma = 0.1$	0	3.66	360	19.02	661	41.18	687	17.8	275	50.66	228
	1	-41.23	397	-70.78	660	-63.39	653	-40.2	388	-10.43	307
	2	3.92	454	-1.43	634	27.81	619	24.64	300	54.06	236
	3	-53.35	535	-56.31	572	-59.1	604	-33.24	375	-9.76	301
	4	7.42	375	6.72	759	36.73	669	16.18	296	44.35	227
	5	-40.45	396	-61.12	587	-54.1	593	-37.0	337	-17.72	328
	6	-11.0	72	-68.8	213	-62.28	198	-13.58	68	-4.2	84
$\#_\sigma = 0.5$	0	7.84	723	-11.01	1223	62.43	1185	66.93	550	130.2	378
	1	-100.0	832	-100.0	920	-100.0	936	-56.1	650	39.26	456
	2	6.14	688	-0.59	1105	73.22	1025	71.64	630	153.85	396
	3	-84.34	794	-100.0	944	-82.5	1063	-51.1	691	24.18	425
	4	16.97	826	-11.24	1243	66.69	1213	47.5	538	131.36	391
	5	-86.92	837	-100.0	854	-100.0	905	-24.3	503	54.86	388
	6	-27.3	124	-90.13	359	-71.37	301	-3.33	56	-29.04	150
$\#_\sigma = 1$	0	-15.08	675	-1.08	864	75.76	865	60.3	586	93.32	442
	1	-100.0	791	-100.0	814	-85.22	1022	-40.6	641	-7.2	505
	2	-14.94	535	4.72	777	86.24	721	75.65	497	103.11	404
	3	-100.0	497	-100.0	657	-64.55	872	-26.11	550	1.42	436
	4	-3.6	689	10.29	799	89.18	822	74.11	558	105.91	460
	5	-100.0	533	-100.0	612	-58.88	785	-25.29	500	26.89	408
	6	-40.18	108	-52.01	169	-34.85	125	-21.08	70	-30.59	158
$\#_\sigma = 1.5$	0	3.83	365	30.85	447	83.24	476	79.52	272	100.22	179
	1	-100.0	408	-53.49	509	-23.97	504	7.18	293	48.04	221
	2	-4.17	302	14.67	402	68.45	391	79.98	250	96.97	147
	3	-57.3	389	-44.96	453	-19.47	449	5.83	290	48.86	166
	4	-8.38	324	24.77	428	75.32	426	85.94	273	99.13	201
	5	-100.0	312	-42.04	402	-2.13	379	23.05	259	60.87	175
	6	-31.39	77	-29.45	87	-29.0	89	-13.84	35	-12.98	102
$\#_\sigma = 1.75$	0	-7.14	288	24.57	342	81.97	353	61.58	192	98.76	153
	1	-52.98	358	-20.9	293	-5.29	323	12.99	213	45.66	178
	2	-1.71	225	21.43	277	68.39	270	68.62	180	95.11	133
	3	-45.93	309	-14.98	268	-1.01	294	22.19	211	53.56	145
	4	-2.54	241	26.46	294	55.77	291	65.92	201	84.62	152
	5	-41.71	287	-9.73	225	9.79	250	20.48	200	61.97	152
	6	-35.28	76	-19.16	67	-29.96	76	-23.36	37	-16.02	74
$\#_\sigma = 2$	0	1.43	226	28.09	271	54.88	268	35.91	141	64.38	116
	1	-30.57	245	5.14	165	19.55	184	4.65	144	35.95	125
	2	-5.94	191	28.81	190	54.98	202	49.3	144	58.95	106
	3	-36.5	228	3.9	160	16.28	178	21.79	154	36.32	121
	4	5.85	206	29.2	190	51.79	201	46.95	149	58.28	125
	5	-19.82	192	7.19	138	21.28	147	12.38	138	42.87	125
	6	-27.18	65	-15.52	49	-17.7	54	-24.53	31	0.67	53
$\#_\sigma = 3$	0	1.75	54	18.68	102	17.31	83	27.63	51	17.88	51
	1	-3.91	64	13.87	49	12.0	46	20.85	60	15.28	60
	2	1.32	61	13.41	56	14.79	64	23.26	52	14.02	52
	3	-7.05	73	8.19	46	7.33	48	17.52	55	8.29	58
	4	3.13	67	20.95	57	18.84	64	26.98	60	17.25	57
	5	-4.45	53	13.73	39	12.17	39	23.27	51	17.39	57
	6	-1.17	13	-3.24	20	-2.18	25	3.19	5	8.01	16

Table 6.2: Returns of various standard deviation thresholds of each strategy

.

Gas Price Threshold	Pool Pair	Strategy's Annual Percentage Rate (APR) - Trading from 18th December 2021 to 9th June 2023									
		Constant		Sliding Window		Lagged		Granger Causality		Kalman Filter	
		Return %	# of Trades	Return %	# of Trades	Return %	# of Trades	Return %	# of Trades	Return %	# of Trades
60^{th} percentile = 4.7e-08	0	7.97	193	26.18	221	30.36	242	30.83	124	51.35	105
	1	-23.52	253	-0.69	178	-9.49	205	7.4	139	26.12	124
	2	8.71	134	30.05	171	26.33	169	30.82	103	51.46	77
	3	-22.44	214	0.58	165	-8.34	188	7.65	134	31.41	90
	4	14.69	150	34.45	178	24.69	184	31.4	123	47.09	90
	5	-12.54	191	6.98	128	3.85	150	15.33	124	38.32	96
	6	-27.3	53	-4.83	30	-15.56	46	-25.36	26	-6.35	50
70^{th} percentile = 6.36e-08	0	12.85	216	33.49	255	33.74	277	33.2	146	58.28	120
	1	-24.53	283	-2.14	209	-17.44	243	4.14	161	29.83	140
	2	12.4	156	33.28	199	25.97	200	36.35	123	61.2	94
	3	-23.07	240	0.35	193	-14.67	219	4.35	162	35.85	107
	4	14.6	175	33.64	215	23.48	219	38.4	152	59.88	111
	5	-13.43	220	6.63	152	-3.77	179	13.97	148	42.8	116
	6	-26.12	59	-6.02	37	-17.86	56	-17.84	30	-8.94	58
80^{th} percentile = 8.83e-08	0	12.79	239	37.89	282	42.17	298	55.47	159	48.36	127
	1	-27.28	310	-3.55	235	-17.09	267	17.17	178	19.48	148
	2	12.72	182	37.34	224	34.2	222	63.04	141	50.22	102
	3	-27.36	266	-1.32	220	-14.55	247	14.51	177	24.88	116
	4	14.38	196	37.34	241	30.37	240	63.65	164	50.73	122
	5	-14.74	239	8.61	177	-3.93	201	25.26	162	32.34	123
	6	-25.48	60	-12.8	46	-16.09	59	-16.77	32	-8.39	59
90^{th} percentile = 1.29e-07	0	7.19	261	35.32	307	89.98	323	52.91	176	82.2	146
	1	-40.49	336	-9.77	264	5.2	294	10.93	197	40.46	167
	2	5.12	201	31.92	249	73.17	244	63.31	161	80.48	122
	3	-34.87	287	-6.13	243	7.72	272	15.23	195	46.48	133
	4	8.38	217	32.28	267	66.24	264	58.31	187	82.52	139
	5	-29.62	265	-0.58	201	22.09	226	20.87	183	55.01	144
	6	-30.43	69	-15.39	56	-18.0	70	-18.27	34	-9.18	67
100^{th} percentile = 2.13e-05	0	-2.2	288	30.8	342	87.79	353	66.76	192	100.13	153
	1	-47.23	358	-16.28	293	0.91	323	16.58	213	46.78	178
	2	2.63	225	28.29	277	74.07	270	75.52	180	97.43	133
	3	-42.06	309	-9.95	268	4.71	294	25.33	211	52.93	145
	4	1.72	241	29.97	294	62.84	291	69.21	201	86.18	152
	5	-37.04	287	-4.83	225	14.37	250	23.51	200	62.31	152
	6	-33.66	76	-17.93	67	-27.78	76	-21.93	37	-14.77	74

Table 6.4: Returns of each strategy when using different gas price thresholds

.

Initial Investment	Pool Pair	Strategy's Annual Percentage Rate (APR) - Trading from 18th December 2021 to 9th June 2023									
		Constant		Sliding Window		Lagged		Granger Causality		Kalman Filter	
		Return %	# of Trades	Return %	# of Trades	Return %	# of Trades	Return %	# of Trades	Return %	# of Trades
$1ETH = \$3,946.26$	0	-100.0	3	-100.0	4	-100.0	2	-100.0	2	-100.0	3
	1	-100.0	3	-100.0	4	-100.0	2	-100.0	2	-100.0	3
	2	-100.0	3	-100.0	4	-100.0	2	-100.0	3	-100.0	3
	3	-100.0	3	-100.0	4	-100.0	2	-100.0	3	-100.0	3
	4	-100.0	3	-100.0	4	-100.0	2	-100.0	4	-100.0	3
	5	-100.0	3	-100.0	5	-100.0	2	-100.0	2	-100.0	3
	6	-100.0	4	-100.0	3	-100.0	2	-100.0	2	-100.0	2
$5ETH = \$19,731.30$	0	-100.0	72	-100.0	133	-100.0	226	-100.0	144	-92.99	146
	1	-100.0	67	-100.0	104	-100.0	162	-100.0	30	-100.0	160
	2	-100.0	79	-100.0	138	-100.0	199	-100.0	132	-87.15	122
	3	-100.0	71	-100.0	135	-100.0	165	-100.0	112	-100.0	121
	4	-100.0	63	-100.0	135	-100.0	174	-100.0	129	-88.55	139
	5	-100.0	67	-100.0	114	-100.0	147	-100.0	100	-97.5	144
	6	-100.0	37	-96.31	56	-100.0	70	-100.0	8	-100.0	35
$10ETH = \$39,462.60$	0	-100.0	241	-73.29	307	-5.49	323	-9.44	176	28.26	146
	1	-100.0	216	-100.0	227	-100.0	291	-56.99	197	-7.98	167
	2	-72.85	201	-55.37	249	-3.76	244	-1.41	161	28.45	122
	3	-100.0	231	-100.0	240	-73.99	272	-50.8	195	1.99	133
	4	-80.04	217	-64.22	267	-14.27	264	-11.29	187	30.22	139
	5	-100.0	221	-75.2	201	-48.32	226	-42.44	183	6.61	144
	6	-52.11	69	-37.11	56	-42.18	70	-35.21	34	-30.05	67
$25ETH = \$98,656.50$	0	-18.93	261	7.87	307	62.82	323	33.34	176	67.25	146
	1	-71.42	336	-33.85	264	-17.26	294	-5.83	197	28.16	167
	2	-15.96	201	7.51	249	49.6	244	45.58	161	66.77	122
	3	-58.44	287	-26.86	243	-12.37	272	-3.11	195	32.29	133
	4	-15.09	217	6.05	267	42.03	264	37.49	187	65.93	139
	5	-53.22	265	-19.49	201	2.9	226	4.43	183	40.5	144
	6	-38.19	69	-22.01	56	-26.43	70	-24.02	34	-15.81	67
$50ETH = \$197,313.00$	0	-0.76	261	27.61	307	82.45	323	48.52	176	78.51	146
	1	-49.33	336	-16.59	264	-1.54	294	6.43	197	37.68	167
	2	-1.27	201	25.19	249	67.31	244	59.25	161	77.32	122
	3	-41.96	287	-11.83	243	3.3	272	10.75	195	43.43	133
	4	0.85	217	25.11	267	59.95	264	53.16	187	78.63	139
	5	-37.12	265	-6.58	201	17.6	226	17.37	183	51.84	144
	6	-33.3	69	-17.81	56	-20.8	70	-20.18	34	-11.58	67
$75ETH = \$295,969.50$	0	4.54	261	32.75	307	87.19	323	51.42	176	80.97	146
	1	-43.77	336	-12.39	264	2.51	294	9.31	197	39.03	167
	2	2.97	201	29.4	249	70.56	244	61.79	161	79.32	122
	3	-37.01	287	-8.16	243	6.7	272	13.5	195	45.19	133
	4	6.18	217	29.66	267	64.16	264	56.71	187	81.31	139
	5	-32.1	265	-2.36	201	20.4	226	19.21	183	53.72	144
	6	-31.37	69	-16.19	56	-18.93	70	-18.9	34	-9.98	67
$100ETH = \$394,626.00$	0	7.19	261	35.32	307	89.98	323	52.91	176	82.2	146
	1	-40.49	336	-9.77	264	5.2	294	10.93	197	40.46	167
	2	5.12	201	31.92	249	73.17	244	63.31	161	80.48	122
	3	-34.87	287	-6.13	243	7.72	272	15.23	195	46.48	133
	4	8.38	217	32.28	267	66.24	264	58.31	187	82.52	139
	5	-29.62	265	-0.58	201	22.09	226	20.87	183	55.01	144
	6	-30.43	69	-15.39	56	-18.0	70	-18.27	34	-9.18	67

Table 6.5: Returns of each strategy when using different initial investment volumes (Note: Ethereum to USD conversion rate is from 18th December 2021)

Window Size	Pool Pair	Strategy's Annual Percentage Rate (APR) - Trading from 9th June 2022 to the 9th June 2023									
		Constant		Sliding Window		Lagged		Granger Causality		Kalman Filter	
		Return %	# of Trades	Return %	# of Trades	Return %	# of Trades	Return %	# of Trades	Return %	# of Trades
7 days	0	22.15	404	51.04	329	35.35	357	76.7	238	24.5	358
	1	-36.93	328	-11.1	273	-28.35	304	-1.24	316	-26.28	293
	2	21.97	365	46.2	336	36.53	332	81.15	234	15.32	362
	3	-37.98	323	-18.41	278	-27.55	293	-8.08	315	-35.33	306
	4	26.56	377	44.91	351	40.59	344	83.15	255	23.02	334
	5	-26.07	272	-6.36	231	-21.73	249	15.63	268	-21.63	239
	6	-28.77	63	-21.75	67	-23.42	65	-15.56	31	-27.23	67
14 days	0	20.66	383	11.37	293	15.97	352	40.32	182	10.84	353
	1	-31.35	318	-24.58	249	-38.7	287	-15.16	246	-34.38	277
	2	17.73	327	10.61	283	5.99	311	38.21	186	6.87	311
	3	-29.1	299	-35.08	268	-40.28	297	-20.64	256	-43.54	312
	4	20.21	346	11.44	278	5.44	308	47.6	212	12.56	305
	5	-18.27	251	-13.4	192	-29.71	242	-8.71	200	-28.11	231
	6	-24.32	42	-17.15	45	-5.46	38	-38.92	32	-19.27	45
30 days	0	21.69	289	39.05	260	75.61	270	55.53	104	10.56	323
	1	-32.04	318	-9.96	219	-4.39	238	21.46	133	-42.09	287
	2	19.44	208	39.02	203	66.67	193	57.53	85	3.43	232
	3	-26.53	274	-7.43	200	0.83	219	24.28	117	-39.38	241
	4	15.11	245	36.06	217	52.06	206	60.94	104	8.39	257
	5	-12.89	214	2.11	153	14.63	168	35.09	105	-30.14	218
	6	-9.55	24	0.45	26	11.62	16	-23.8	21	-5.29	24
60 days	0	15.48	314	48.26	275	68.23	277	81.41	108	21.07	351
	1	-28.11	303	-0.49	180	20.55	172	42.31	135	-34.98	259
	2	22.46	100	40.3	92	67.59	83	77.2	50	13.71	147
	3	-6.78	179	13.53	112	39.52	96	56.05	68	-22.18	154
	4	25.22	170	47.69	139	67.06	120	83.31	106	13.68	209
	5	7.16	141	25.9	80	48.05	76	48.54	135	-0.08	97
	6	-7.56	23	2.66	25	14.11	15	-20.72	17	-3.22	23
90 days	0	16.47	238	47.24	289	64.28	295	87.26	71	30.47	287
	1	-17.53	208	7.99	149	26.87	144	63.9	75	-15.38	176
	2	28.07	56	42.6	42	71.06	33	92.17	36	14.67	46
	3	10.76	78	30.95	45	57.68	35	67.29	53	0.61	59
	4	32.23	137	55.23	106	76.85	103	93.66	73	30.24	126
	5	12.98	91	33.68	54	55.49	56	76.41	36	7.44	61
	6	-7.56	23	2.66	25	14.11	15	-20.72	17	-3.22	23

Table 6.6: Returns of various Window Sizes

.

Pool Pair	Strategy's Annual Percentage Rate (APR) - Trading from 14th January 2022 to 9th June 2023									
	Constant		Sliding Window		Lagged		Granger Causality		Kalman Filter	
	Return %	# of Trades	Return %	# of Trades	Return %	# of Trades	Return %	# of Trades	Return %	# of Trades
0	24.84	83	38.61	250	65.5	276	86.91	310	145.74	378
1	20.7	37	14.38	127	29.54	151	20.37	269	43.94	456
2	22.12	46	30.74	103	49.75	112	92.19	218	164.55	396
3	18.14	31	14.26	86	33.67	98	28.51	231	33.36	425
4	27.65	45	38.44	122	66.23	140	92.71	290	152.26	391
5	23.11	25	25.76	77	45.57	98	31.8	299	57.81	388
6	-2.54	13	-10.28	30	-19.31	49	-10.56	23	-29.71	150
Average	19.14	40	21.70	113.6	38.70	132	48.85	234.3	81.14	369.1

Table 6.8: Returns of each strategy with their optimal parameters

.

Pool Pair	Strategies' Beta, β - Trading from 14th January 2022 to 9th June 2023				
	Constant	Sliding Window	Lagged	Granger Causality	Kalman Filter
0	-1.57001e-11	-2.37822e-11	-2.34107e-11	-1.17100e-11	-3.36017e-11
1	-1.33728e-11	-2.55276e-11	-2.34361e-11	-1.48089e-11	-2.80161e-11
2	-1.44463e-11	-2.51260e-11	-2.48695e-11	-1.08178e-11	-3.28074e-11
3	-1.30815e-11	-2.34876e-11	-2.48109e-11	-1.12766e-11	-3.32465e-11
4	-1.38961e-11	-2.59124e-11	-2.78297e-11	-1.86356e-11	-3.65904e-11
5	-1.43195e-11	-3.08987e-11	-2.79315e-11	-5.46023e-12	-3.38610e-11
6	3.42127e-12	-4.57712e-11	-3.56508e-11	-9.80665e-11	-1.43624e-10
Average	-1.162784e-11	-2.864367e-11	-2.684845e-11	-2.439650e-11	-4.882103e-11

Table 6.9: βs of each strategy

.

Strategy	Volatility	Average Return	Sharpe Ratio
Constant	9.28	19.14	1.46
Sliding Window	16.02	21.70	1.10
Lagged	27.04	38.70	1.29
Granger Causality	38.35	48.85	1.20
Kalman Filter	68.34	81.14	1.17

Table 6.10: Volatility and Sharpe Ratios of each strategy

.

Chapter 7

Conclusion - TODO

Appendix A

Additional Background

A.1 Pure Arbitrage

As mentioned in Chapter 2.3, research into this topic is still in its infancy thus which means a very thin slice of exploration on the subject matter. The majority of the research has been into the arbitrage on centralized exchanges [55, 56, 57]. Cristian Pauna investigates and implements an arbitrage strategy in [57]. The paper details the technical details of arbitrage trading from the data and the system architecture used. Pauna finds complications such as requesting data from multiple exchanges, converting the data such that it is homogeneous and also managing server load. Pauna presents the architecture such that the servers request data from the necessary exchanges, aggregating prices in a relational database which then triggers a server that is used to generate trading signals.

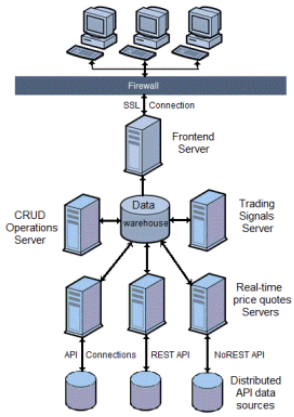

Figure A.1: Arbitrage system architecture [57]

Triangular and cyclic arbitrage is one of the most used and purest forms of arbitrage to implement and analyse, [54] explores triangular arbitrage on decentralised exchanges. Algorithm 2 is the algorithm used to find the most profitable arbitrage route on a particular platform, once this is calculated, it is compared with other routes on other platforms. Initially, the system converts the base token into another token and converts it back into the base token, using only one token is used as a middle route, then using the algorithm below, increases the number of middle tokens.

On evaluating the performance of the strategy on differing platforms depended on three main features of each exchange:

Algorithm 2 Maximum Profit Route Searching (R)

Input: T (token list), P (price graph), n (current route)

```
for i = 1, ..., T do
    r = get_profit(n + i)
    for j = 1, ..., P[i] do
        p = max(r, R(T, P, n_j))
    end for
end for
return p
```

1. Portion size - Depending on how much the "trader" invested revenues differed and with the larger portion size, the revenue decreases as the token pair prices are adjusted based on supply/demand.

2. Transaction fees - Each exchange has its own transaction fee.

3. Other considerations such as price slippage - Exchanges have different liquidity levels which depend on the usage and liquidity providers that the exchange employs.

Figure A.2 displays the revenues obtained by same trading token route, ETH → MKR → OMG → USDT → ETH. As we can see upon applying the strategy on multiple exchanges; Uniswap, 1inch, Kyberswap and Bancor, 1 inch was the only exchange that generated a profit whereas the others lose money [54].

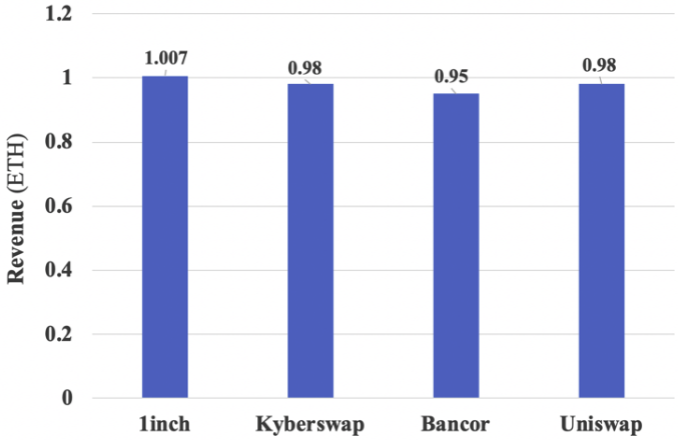

Figure A.2: Trading profits same token routes within different exchanges [54]

Another paper that implemented and evaluated a cyclic arbitrage opportunity is [52]. The research consists of proposing a theoretical arbitrage model and further evaluation of real transactional data. The arbitrage model used is simple to understand, as it searches for a cyclic transaction between n tokens, $A_1, A_2, ..., A_n$ is a sequence of n trades:

$$\textit{Trade 1:}\ \text{Exchange } \delta_1 \text{ of } A_1 \text{ to } \delta_2 \text{ of } A_2$$

$$\textit{Trade 2:}\ \text{Exchange } \delta_2 \text{ of } A_2 \text{ to } \delta_3 \text{ of } A_3$$

$$\dots$$

$$\textit{Trade n:}\ \text{Exchange } \delta_n \text{ of } A_n \text{ to } \delta_1' \text{ of } A_1$$

It is important to note that $\delta_i = \delta_{i+1}$, i.e. the output of trade is equivalent to the input of the next. The revenues within a cycle are defined as $\delta_{i+1} - \delta_i$, and the overall profit

is $\delta_1' - \delta_1$. This is not as simple as the revenues depend on how liquid the exchange is, thus the liquidity pools of each possible trading pair are hugely important. Therefore, the paper proposes a theorem, below:

Theorem 1 *For a given cycle $A_1 \to A_2 \to \cdots \to A_n \to A_1$ with n tokens, there exists an arbitrage opportunity for the cyclic transaction if the product of exchange rates $\frac{a_{2,1}a_{3,2}\cdots a_{1,n}}{a_{1,2}a_{2,3}\cdots a_{n,1}} > \frac{1}{r_1^n r_2^n}$ where $a_{i,j}$ denotes the liquidity of token A_i in the liquidity pool with token A_j. [52]*

In addition to the theorem, to obtain an optimal strategy we need to compute the optimal trading volume of a cycle, $A_1 \to A_2 \to \cdots \to A_n \to A_1$. The paper proposes the optimal trading volume to be $\delta_a^{op} = \frac{\sqrt{r_1 r_2 a' a} - a}{r_1}$ where $a = \frac{a'_{1,n} a_{n,1}}{a_{n,1} + r_1 r_2 a'_{n,1}}$ and $a' = \frac{r_1 r_2 a'_{1,n} a_{n,1}}{a_{n,1} + r_1 r_2 a'_{n,1}}$. Thus to calculate such arbitrage opportunities knowing the liquidity of tokens in other tokens' liquidity pools, algorithm 3 infers the direction and volumes to trade to get the optimal revenue.

Algorithm 3 Computing the equivalent liquidity of the cycle

$a'_{1,n} \leftarrow a_{1,2}$
$a'_{n,1} \leftarrow a_{2,1}$
for i from 2 to $n-1$ **do**
  $a'_{1,n} \leftarrow \frac{a'_{1,n} a_{i,i+1}}{a_{i,i+1} + r_1 r_2 a'_{n,1}}$
  $a'_{n,1} \leftarrow \frac{r_1 r_2 a'_{1,n} a_{i+1,i}}{a_{i,i+1} + r_1 r_2 a'_{n,1}}$
end for

After analyzing Ethereum block data and applying this strategy to identify the number of arbitrage opportunities, it was found that between May 4, 2020, and April 15, 2021, there were numerous exploitable and profitable arbitrage opportunities. These opportunities grew consistently to reach 1,750 in 11 months, as depicted in Figure A.3. Only cycles with length 3 were experimented with and only cycles including ETH as 80% of the liquidity pools on Uniswap include ETH and another cryptocurrency [58]. Furthermore, it is found that 287,241 of the 292,606 arbitrages executed started with ETH, and 85% of the arbitrages used a cycle of length 3. The total revenue of the cyclic arbitrage was 34,429 ETH. However, gas fees account for 24.6% of the total revenue leaving an approximate 25,971 ETH profit.

The paper then delves into the implementation of the smart contract, and explores how both *sequential* and *atomic* implementations would affect the revenue and execution of the contracts. It was found that 52.3% of the arbitrages that were executed sequentially generated a loss, likely due to the fact that, when one submits n orders, the n blockchain transactions are executed sequentially, meaning some external transactions can be inserted between these transactions. Thus using atomic transactions avoids this issue of external transactions does not affect the market price that may affect the outcome of the arbitrage.

Furthermore, the authors of the paper also investigated the performance differences between using private smart contracts and public contracts. Deploying a smart contract that calls Uniswap functions, i.e. a private smart contract, is intuitively better and achieves a higher success rate of a lower bound of 52% and a higher bound of 90% in comparison to calling a public Uniswap smart contract which has a success rate of 27.3%. Overall the paper provides an insightful look into cyclic arbitrage in DEXes and highlights important decisions made such as liquidity calculations and smart contracts while comparing the performance of different options available.

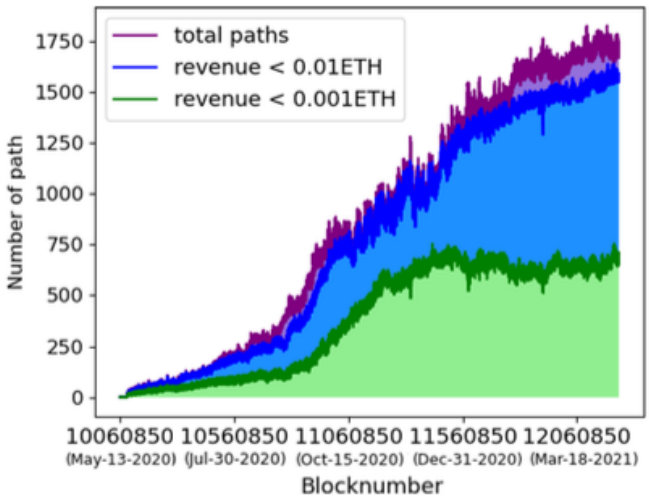

Figure A.3: Number of exploitable opportunities in Uniswap V2 over time. The purple line represents the number of cycles that provide revenue higher than 0.0001 ETH. The green represents the number of cycles whose revenue is under 0.001 ETH. The blue line represents the number of cycles whose revenue is under 0.01 ETH. [52]

A.2 Optimal Portfolio Design for Mean Reversion

There has been further research into optimizing mean reversion, one of which was to use the successive convex approximation method on the mean reverting portfolio design [59]. The paper initially proposes the mean reversion portfolio:

- For each asset, the price at time t is denoted as p_t and its correspondeing log-price $y_t \triangleq log(p_t)$, its vector form of M assets $\mathbf{y}_t \triangleq \left[y_{1,t}, \ldots, y_{M,t}\right]^T$.
- The log-price spread is given by $y_t \triangleq \beta^T \mathbf{y}_t$, where $\beta \triangleq \left[\beta_1, \ldots, \beta_M\right]^T$ denotes the hedge ratios.
- The cointegration space with N relations is defined by $\mathbf{B} \triangleq \left[\beta_1, \ldots, \beta_N\right]$, thus the N spreads are $s_t \triangleq \mathbf{B}^T \mathbf{y}_t$.
- For these N spreads, the portfolio weight matrix is denoted as $\mathbf{w} \triangleq \left[w_1, \ldots, w_N\right]^T$.
- The auto-covariance matrix for the spreads s_t is defined as $M_i \triangleq Cov(s_t, s_{t+i}) = \mathbb{E}\left[\left(s_t - \mathbb{E}\left[s_t\right]\right)\left(s_{t+i} - \mathbb{E}\left[s_{t+i}\right]\right)^T\right]$

Now that we have defined everything required, we can now formalize the problem. The general problem of mean reversion portfolio design problem is formalized by:

$$\underset{\mathbf{w}}{\text{minimize}} \quad F(\mathbf{w}) \triangleq U(\mathbf{w}) + \mu V(\mathbf{w}) + \gamma S(\mathbf{w})$$

$$\text{subject to} \quad \mathbf{w} \in \left\{\mathbf{w} \mid \left\|\mathbf{B}\mathbf{w}\right\|_0 \leq L\right\}, \qquad \text{where } L \text{ is the total leveraged investment}$$

- μ defines the trade-off between the mean reversion measure and the variance preference.
- γ defines the regularization parameter of how sparse we would like the cointegration space to be.

Where the Mean Reversion term:

$$U(\mathbf{w}) \triangleq \xi \frac{\mathbf{w}^T \mathbf{H} \mathbf{w}}{\mathbf{w}^T \mathbf{M}_0 \mathbf{w}} + \zeta \left(\frac{\mathbf{w}^T \mathbf{M}_1 \mathbf{w}}{\mathbf{w}^T \mathbf{M}_0 \mathbf{w}}\right)^2 + \eta \sum_{i=2}^{p} \left(\frac{\mathbf{w}^T \mathbf{M}_i \mathbf{w}}{\mathbf{w}^T \mathbf{M}_0 \mathbf{w}}\right)^2$$

And the variance term:

$$V(\mathbf{w}) \triangleq \begin{cases} 1/\mathbf{w}^T\mathbf{M}_0\mathbf{w} & \text{VarInv}(\mathbf{w}) \\ 1/\sqrt{\mathbf{w}^T\mathbf{M}_0\mathbf{w}} & \text{StdInv}(\mathbf{w}) \\ -\mathbf{w}^T\mathbf{M}_0\mathbf{w} & \text{VarNeg}(\mathbf{w}) \\ -\sqrt{\mathbf{w}^T\mathbf{M}_0\mathbf{w}} & \text{StdNeg}(\mathbf{w}) \end{cases}$$

The variance term can be represented in any of the four forms.

And the asset selection term:

$$S(\mathbf{w}) \triangleq \left\|\mathbf{B}\mathbf{w}\right\|_0 = \sum_{m=1}^{M} sgn(|\ \left[\mathbf{B}\mathbf{w}\right]_m\ |)$$

This asset selection criterion is not necessary however as trading incurs a cost, selecting all of the assets is costly, thus selecting a subset of assets to trade is more profitable. To formalize this goal, we would like to minimize the cointegration space thus we use the ℓ_0 norm.

The paper then goes on to solve the optimization problem using the successive convex approximation (SCA) method [60]. The SCA method takes an optimization problem in the form of:

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & \mathbf{x} \in \mathcal{X} \end{aligned}$$

Where $\mathcal{X} \subseteq \mathbb{R}^N$ is convex and $f(\mathbf{x})$ is non-convex. The SCA method involves starting at an initial point $\mathbf{x}^{(0)}$ and solving a series of subproblems of surrogate funtions $\tilde{f}(\mathbf{x}; \mathbf{x}^{(k)})$ over the set $\mathcal{X}$. The sequence $\left\{\mathbf{x}^{(k)}\right\}$ is generated by:

$$\begin{cases} \hat{\mathbf{x}}^{(k+1)} = \underset{\mathbf{x}\in\mathcal{X}}{argmin}\, \tilde{f}(\mathbf{x}; \mathbf{x}^{(k)}) \\ \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \gamma^{(k)}(\hat{\mathbf{x}}^{(k+1)} - \mathbf{x}^{(k)}) \end{cases}$$

The first step is to generate a descent direction and then update the variable with a step size of $\gamma^{(k)}$. After applying this method to the MRP problem and further analysis of the paper, the following algorithm is proposed and used to solve the MRP design problem:

Algorithm 4 SCA-Based Algorithm for The Optimal MRP Design Problem

Require: $\mathbf{H}, \mathbf{M}_i, \mu, \gamma, \mathbf{B}, L$ and τ

1: Set $k = 0, \gamma^{(0)}$ and $\mathbf{w}^{(0)}$
2: **repeat**
3: Compute $\mathbf{A}^{(k)}$ and $\mathbf{b}^{(k)}$
4: $\hat{\mathbf{w}}^{(k+1)} = \underset{\mathbf{w}\in\mathcal{W}}{argmin}\, \mathbf{w}^T\mathbf{A}^{(k)}\mathbf{w} + \mathbf{b}^{(k)T}\mathbf{w}$
5: $\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \gamma^{(k)}(\hat{\mathbf{w}}^{(k+1)} - \mathbf{w}^{(k)})$
6: $k \leftarrow k + 1$
7: **until** convergence

However, 4 line is a convex problem and has no closed-form solution thus to solve this subproblem using the ADMM method, this is done by introducing an auxiliary variable $\mathbf{z} = \mathbf{B}\mathbf{w}$.

$$\begin{aligned} \underset{\mathbf{x},\mathbf{z}}{\text{minimize}} \quad & \mathbf{w}^T\mathbf{A}\mathbf{w} + \mathbf{b}^T\mathbf{w} \\ \text{subject to} \quad & \left\|\mathbf{z}\right\|_1 \leq B, \mathbf{B}\mathbf{w} - \mathbf{z} = \mathbf{0} \end{aligned}$$

This is then summarized into Algorithm 5:

Algorithm 5 An ADMM-Based Algorithm for Problem on line 4 in Algorithm 4

Require: $\mathbf{A}, \mathbf{b}, \mathbf{B}, B, \rho$

1: Set $\mathbf{w}^{(0)}, \mathbf{z}^{(0)}, \mathbf{u}^{(0)}$ and $k = 0$
2: **repeat**
3: $\quad \mathbf{w}^{(k+1)} = -(2\mathbf{A} + \rho\mathbf{B}^T\mathbf{B})^{-1}(\mathbf{b} + \rho\mathbf{B}^T(\mathbf{u}^{(k)} - \mathbf{z}^{(k)}))$
4: $\quad \mathbf{h}^{(k)} = \mathbf{B}\mathbf{w}^{(k+1)} + \mathbf{u}^{(k)}$
5: $\quad \mathbf{z}^{(k+1)} = \Pi_{\mathcal{C}}(\mathbf{h}^{(k)})$
6: $\quad \mathbf{u}^{(k+1)} = \mathbf{u}^{(k)} + \mathbf{B}\mathbf{w}^{(k+1)} - \mathbf{z}^{(k+1)}$
7: $\quad k \leftarrow k + 1$
8: **until** convergence

After all of this analysis, the authors of the paper, [61, 59], ran simulations on real data comparing underlying spread. It found that it resulted in consistent profits as shown in Figure A.4.

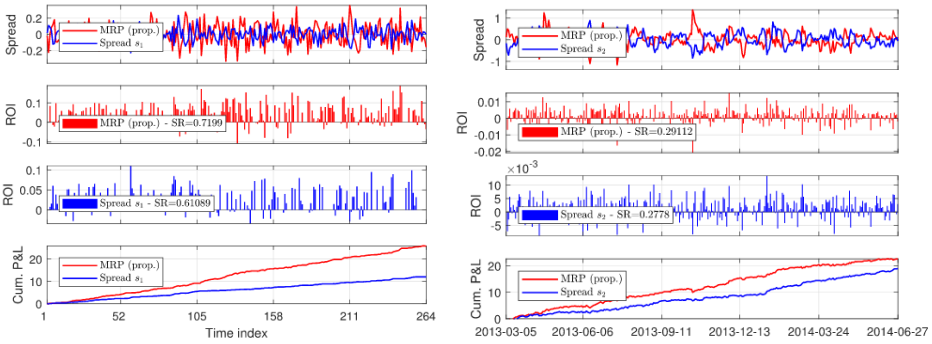

Figure A.4: A mean-reversion trading based on real data [59]

Overall, this research successfully formulizes, solves the optimization problem mathematically, and goes further to implement the algorithms to solve the problem programmatically. In addition, the author compares the implementation with other benchmark algorithms, showing that it results in a greater P&L and Sharpe ratio.

Bibliography

[1] "'I lost millions through cryptocurrency trading addiction'," *BBC News*, May 2021. [Online]. Available: https://www.bbc.com/news/uk-scotland-57268024

[2] C. Gondek, "What Big Companies Are Investing In Cryptocurrency?" [Online]. Available: https://originstamp.com/blog/what-big-companies-are-investing-in-cryptocurrency/

[3] "History of bitcoin exchanges and trading - Bit2Me Academy." [Online]. Available: https://academy.bit2me.com/en/history-exchanges-bitcoin-trading/

[4] "DEX to CEX Spot Trade Volume." [Online]. Available: https://www.theblock.co/data/decentralized-finance/dex-non-custodial/dex-to-cex-spot-trade-volume

[5] M. A. Egiyi and G. N. Ofoegbu, "Cryptocurrency and climate change: An overview," *International Journal of Mechanical Engineering and Technology (IJMET)*, vol. 11, no. 3, pp. 15–22, 2020.

[6] M. Nofer, P. Gomber, O. Hinz, and D. Schiereck, "Blockchain," *Business & Information Systems Engineering*, vol. 59, no. 3, pp. 183–187, 2017.

[7] "Ethereum Whitepaper." [Online]. Available: https://ethereum.org

[8] Y.-C. Liang, *Blockchain for Dynamic Spectrum Management*, 01 2020, pp. 121–146.

[9] "Merkle Tree in Blockchain: What is it and How does it work | Simplilearn." [Online]. Available: https://www.simplilearn.com/tutorials/blockchain-tutorial/merkle-tree-in-blockchain

[10] "Consensus mechanisms." [Online]. Available: https://ethereum.org

[11] V. Buterin *et al.*, "A next-generation smart contract and decentralized application platform," *white paper*, vol. 3, no. 37, pp. 2–1, 2014.

[12] "What are smart contracts on blockchain? | IBM." [Online]. Available: https://www.ibm.com/topics/smart-contracts

[13] "Introduction to smart contracts." [Online]. Available: https://ethereum.org/en/developers/docs/smart-contracts/

[14] "Smart Contracts - Industrial IoT Use Case Profile | IoT ONE Digital Transformation Advisors." [Online]. Available: https://www.iotone.com/usecase/smart-contracts/u88

[15] "Ethereum Virtual Machine (EVM)." [Online]. Available: https://ethereum.org/en/developers/docs/evm/

[16] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger." [Online]. Available: https://ethereum.github.io/yellowpaper/paper.pdf

[17] "Components of Blockchain Network," Apr. 2021. [Online]. Available: https://www.geeksforgeeks.org/components-of-blockchain-network/

[18] "Gas and fees." [Online]. Available: https://ethereum.org/en/developers/docs/gas/

[19] "What Is a DEX (Decentralized Exchange)? | Chainlink." [Online]. Available: https://chain.link/education-hub/what-is-decentralized-exchange-dex

[20] "Decentralized Exchange Platforms in Crypto Trading." [Online]. Available: https://www.gemini.com/cryptopedia/decentralized-exchange-crypto-dex

[21] "What is arbitrage?" Jul 2021. [Online]. Available: https://online.hbs.edu/blog/post/what-is-arbitrage

[22] G. Poitras, "Origins of arbitrage," *Financial History Review*, vol. 28, no. 1, p. 96–123, 2021.

[23] "Law of One Price: Definition, Example, Assumptions." [Online]. Available: https://www.investopedia.com/terms/l/law-one-price.asp

[24] P. Isard, "How far can we push the "law of one price"?" *The American Economic Review*, vol. 67, no. 5, pp. 942–948, 1977. [Online]. Available: http://www.jstor.org/stable/1828075

[25] J. Richardson, "Some empirical evidence on commodity arbitrage and the law of one price," *Journal of International Economics*, vol. 8, no. 2, pp. 341–351, 1978. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0022199678900272

[26] I. Eyal, "The miner's dilemma," in *2015 IEEE Symposium on Security and Privacy*. IEEE, 2015, pp. 89–103.

[27] Z. Avarikioti, L. Heimbach, Y. Wang, and R. Wattenhofer, "Ride the lightning: The game theory of payment channels," in *International Conference on Financial Cryptography and Data Security*. Springer, 2020, pp. 264–283.

[28] G. Huberman, J. D. Leshno, and C. Moallemi, "Monopoly without a monopolist: An economic analysis of the bitcoin payment system," *The Review of Economic Studies*, vol. 88, no. 6, pp. 3011–3040, 2021.

[29] S. Athey, I. Parashkevov, V. Sarukkai, and J. Xia, "Bitcoin pricing, adoption, and usage: Theory and evidence," 2016.

[30] D. Easley, M. O'Hara, and S. Basu, "From mining to markets: The evolution of bitcoin transaction fees," *Journal of Financial Economics*, vol. 134, no. 1, pp. 91–109, 2019.

[31] C. R. Harvey, "Cryptofinance," *Available at SSRN 2438299*, 2016.

[32] E. Pagnotta and A. Buraschi, "An equilibrium valuation of bitcoin and decentralized network assets," *Available at SSRN 3142022*, 2018.

[33] C. Krauss, "Statistical arbitrage pairs trading strategies: Review and outlook," *Journal of Economic Surveys*, vol. 31, no. 2, pp. 513–545, 2017. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/joes.12153

[34] H. Rad, R. K. Y. Low, and R. Faff, "The profitability of pairs trading strategies: distance, cointegration and copula methods," *Quantitative Finance*, vol. 16, no. 10, pp. 1541–1558, 2016. [Online]. Available: https://doi.org/10.1080/14697688.2016.1164337

[35] Y.-x. Lin, M. Michael, and G. Chandra, "Loss protection in pairs trading through minimum profit bounds: A cointegration approach," *Journal of Applied Mathematics and Decision Sciences*, vol. 2006, 08 2006.

[36] B. Alsadik, "Chapter 10 - kalman filter," in *Adjustment Models in 3D Geomatics and Computational Geophysics*, ser. Computational Geophysics, B. Alsadik, Ed. Elsevier, 2019, vol. 4, pp. 299–326. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128175880000106

[37] G. Vidyamurthy, *Pairs Trading: quantitative methods and analysis*. John Wiley & Sons, 2004, vol. 217.

[38] "How a Kalman filter works, in pictures." [Online]. Available: https://www.bzarg.com/p/how-a-kalman-filter-works-in-pictures/

[39] H. E. Dempsey, "Market Inefficiency: Pairs Trading with the Kalman Filter," Dec. 2017. [Online]. Available: http://www.hedempsey.com/papers/Pairs%20Trading%20with%20a%20Kalman%20Filter.pdf

[40] J. P. Nóbrega and A. L. I. Oliveira, "A combination forecasting model using machine learning and kalman filter for statistical arbitrage," in *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2014, pp. 1294–1299.

[41] C. Krauss, X. A. Do, and N. Huck, "Deep neural networks, gradient-boosted trees, random forests: Statistical arbitrage on the s&p 500," *European Journal of Operational Research*, vol. 259, no. 2, pp. 689–702, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221716308657

[42] S. Moraes Sarmento, *A Machine Learning based Pairs Trading Investment Strategy*, 1st ed., ser. SpringerBriefs in Computational Intelligence. Cham: Springer International Publishing, 2021.

[43] T. G. Fischer, C. Krauss, and A. Deinert, "Statistical arbitrage in cryptocurrency markets," *Journal of Risk and Financial Management*, vol. 12, no. 1, 2019. [Online]. Available: https://www.mdpi.com/1911-8074/12/1/31

[44] D. Ju-Long, "Control problems of grey systems," *Systems & Control Letters*, vol. 1, no. 5, pp. 288–294, 1982. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016769118280025X

[45] R. Engle, "Dynamic conditional correlation: A simple class of multivariate generalized autoregressive conditional heteroskedasticity models," *Journal of Business & Economic Statistics*, vol. 20, no. 3, pp. 339–350, 2002.

[46] G. Figá-Talamanca, S. Focardi, and M. Patacca, "Common dynamic factors for cryptocurrencies and multiple pair-trading statistical arbitrages," *Decisions in economics and finance*, vol. 44, no. 2, pp. 863–882, 2021.

[47] B. M. Blau, T. Griffith, and R. J. Whitby, "Comovement in the cryptocurrency market," *Economics Bulletin*, vol. 40, no. 1, p. 1, 2020.

[48] F. Schär, "Decentralized finance: On blockchain-and smart contract-based financial markets," *FRB of St. Louis Review*, 2021.

[49] "How Uniswap works | Uniswap." [Online]. Available: https://docs.uniswap.org//contracts/v2/concepts/protocol-overview/how-uniswap-works

[50] "Risk Parameters." [Online]. Available: https://docs.aave.com/risk/asset-risk/risk-parameters

[51] "The Graph." [Online]. Available: https://thegraph.com

[52] Y. Wang, Y. Chen, H. Wu, L. Zhou, S. Deng, and R. Wattenhofer, "Cyclic Arbitrage in Decentralized Exchanges," Jan. 2022, arXiv:2105.02784 [cs, q-fin]. [Online]. Available: http://arxiv.org/abs/2105.02784

[53] "Interest rates and Bank Rate," May 2023. [Online]. Available: https://www.bankofengland.co.uk/monetary-policy/the-interest-rate-bank-rate

[54] N. Boonpeam, W. Werapun, and T. Karode, "The arbitrage system on decentralized exchanges," in *2021 18th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*. IEEE, 2021, pp. 768–771.

[55] I. Makarov and A. Schoar, "Trading and arbitrage in cryptocurrency markets," *Journal of financial economics*, vol. 135, no. 2, pp. 293–319, 2020.

[56] T. Crépellière, M. Pelster, and S. Zeisberger, "Arbitrage in the Market for Cryptocurrencies," Rochester, NY, Dec. 2022. [Online]. Available: https://papers.ssrn.com/abstract=3606053

[57] C. PAUNA, "Arbitrage trading systems for cryptocurrencies. design principles and server architecture," *Informatica economica*, vol. 22, no. 2/2018, pp. 35–42, 2018.

[58] L. Heimbach, Y. Wang, and R. Wattenhofer, "Behavior of liquidity providers in decentralized exchanges," *arXiv preprint arXiv:2105.13822*, 2021.

[59] Z. Zhao, R. Zhou, and D. P. Palomar, "Optimal mean-reverting portfolio with leverage constraint for statistical arbitrage in finance," *IEEE transactions on signal processing*, vol. 67, no. 7, pp. 1681–1695, 2019.

[60] Q. Tran-Dinh, W. Michiels, and M. Diehl, "An inner convex approximation algorithm for bmi optimization and applications in control," *Proceedings of the IEEE Conference on Decision and Control*, 02 2012.

[61] Z. Zhao, R. Zhou, Z. Wang, and D. P. Palomar, "Optimal portfolio design for statistical arbitrage in finance," in *2018 IEEE Statistical Signal Processing Workshop (SSP)*, 2018, pp. 801–805.