KU LEUVEN

Pairs Trading with Expected Signatures and Reinforcement Learning

Ir. Laurent Balesse

Thesis submitted for the degree of
Master of Science in Artificial
Intelligence, option Engineering and
Computer Science

Supervisors:
Prof. Johannes De Smedt
Emiel Lemahieu

Assessors:
Yanyi Zhang
Jan De Spiegeleer

Academic year 2023 – 2024

Preface

I would like to use this preface to express my gratitude to the people who contributed to this work and supported me throughout writing this master's thesis. This work would not have been the same without the support from those I am to thank.

First of all, I wish to express my sincerest gratitude to Emiel Lemahieu for embarking on a second thesis adventure together, for investing so much time in me and the problems I encountered, for the hours long late night sparring sessions, for the long theoretical lessons, and for not only being there as a supervisor, but also as a friend.

Secondly, I would like to thank Prof. Johannes De Smedt for allowing me to pursue my own thesis topic and for the helpful and professional feedback sessions and guidance.

Finally, a word of thanks goes out to my friends, family, and colleagues for their continued support, for providing tips and tricks, and for always letting me talk on and on about my thesis.

Ir. Laurent Balesse

Contents

Abstract

This thesis researches a new methodology for pairs trading. The methodology is a two step process involving the selection of optimal pairs through a developed expected signature model based on the relation between signatures and future spreads and the relation between signatures and potential pairs trading PnL and optimal trading of the selected spreads with deep reinforcement learning. The established signature relations are evaluated and the expected signature model is benchmarked against the classical cointegration selection model. Furthermore, the training of the reinforcement learner is analysed and compared with a random agent. The trained model is benchmarked against a random agent and a classic execution model based on standard deviation. The results indicate a strong relation between signatures of a spread and the potential pairs trading PnL. However, the expected signature model struggles with the prediction of future spread values, mostly due to the lack of data. On the execution side the deep Q-learning agent manages to outperform the random agent and the benchmark when comparing cumulative rewards.

List of Figures and Tables

List of Figures

List of Tables

Chapter 1

Introduction

Arbitrage is an established financial strategy that involves buying and selling assets where the trader does not take an active position in the market. The trader has no conviction on where financial markets will evolve to as he aims to profit from temporary market inefficiencies that cause price discrepancies. Different arbitrage strategies exists, but one of the most well known strategies is the concept of pairs trading. Pairs trading is a relative-value arbitrage strategy and is still actively used today by traders. The concept of pairs trading is straightforward: find two assets who have moved together historically and when the spread between the prices widens, buy the losing asset and short the winning asset. If history repeats itself, the asset prices will converge again and the previously losing asset will win while the previously winning asset will lose. Pairs trading is classified under the statistical arbitrage trading methods given that it relies on statistical information, the fact that two assets are mean-reverting, in order to apply arbitrage. Arbitrage here assumes that the buy and the sell is dollar neutral, meaning the assets one holds have the same monetary value and even each other out, lowering the risk towards the market.

Pairs trading typically consists of two consecutive processes. The first process is the selection of optimal pairs based on historical prices and price moves. The ideal pair has a long standing mean-reverting relationship with a highly volatile spread creating many trading opportunities. The second process is the actual trading of the spread between the two selected assets. Timing is key here as the trader wants to take position when the price spread is at its maximum and wants to exit the position when the price spread is at its minimum, before increasing again and creating a new trading opportunity.

Recent developments in the area of machine learning have provided promising frameworks to increase profitability of trading in financial markets. One of these developments is reinforcement learning, a third branch inside the machine learning universe next to supervised and unsupervised learning. In reinforcement learning, an agent interacts with the environment. From this interaction comes a reward or a punishment for the agent, and based on the accumulation of rewards/punishments

the agent is capable of learning a specific task. In theory, it should then be possible to build an agent that learns how to execute a pairs trading strategy on a provided universe of assets. One of the major drawbacks of this method, however, is the amount of data that is required for the agent to learn. The more complex the task we expect the agent to learn, the more data is required.

In our research we intend to use reinforcement learning to create a profitable pairs trading strategy. In order to overcome the issues with the required data we simplify the task for the reinforcement learner substantially by making an optimal selection of pairs before feeding the selected pairs to the reinforcement learner, which then learns to optimally trade the price spread between the assets in a dollar-neutral way.

While there exists classical methods for selection of pairs for pairs trading, like minimal price distance strategies and co-integration strategies, these strategies focus more on finding historical mean-reverting pairs and less on potential profitability for a trading strategy. In our research we present a data-driven methodology based on truncated signatures to select the optimal pairs with as goal maximizing the potential profitability for pairs trading. Signatures are a mathematical technique coming forth from rough path theory, where they are used to describe the interactions of complex oscillatory systems. These signatures allow us to use high-dimensional information in time series while working in a linear setting, maximizing the time series information use while keeping the model complexity low.

Chapter 2

Literature Review

With at least a 40-year history on Wall Street, there is an abundance of research on different approaches to pairs trading. Hence, the first section of the literature review describes the concept of pairs trading and discusses several classic and more modern approaches. The second section looks at the concept of signatures and their characteristics and discusses how these signatures can be a helpful tool when identifying pairs for pairs trading. The third and last section provides insight in reinforcement learning, its required building blocks, and the different approaches available to design a reinforcement learning agent.

2.1 Pairs trading

Pairs trading belongs to a group of statistical arbitrage strategies given it tries to find and profit from arbitrage opportunities based on statistical properties of different assets in the asset universe. The pairs trading approach is a relative-value arbitrage strategy that aims to create a positive return from selling one asset while buying another in a dollar neutral fashion (buying and selling for equal dollar amounts). The ideal asset pair for pairs trading is a pair of assets that is mean-reverting over the long run, but has a highly volatile spread. If such a pair can be identified, the pairs trader shorts the better performer and goes long on the under performer given it is the expectation that the better performer will loose over the short term and the under performer will win again over the short term. In this case a profit would be made on both the short position and the long position. However, should the market go through a general increase(decrease) over this time period, the loss of the short(long) position will be offset by the double winnings on the long(short) position. Because of this return behaviour we can call pairs trading a relative-value arbitrage strategy. The return only relies on the relative value between the chosen assets and is mostly independent from the general market movements. Figure 2.1 provides a schematic overview of what a pairs trading strategy looks like. In the figure, the position of the trader on the spread is shown next to the normalized asset prices. The trader opens a position when the spread between the normalized prices is substantially large and closes the position when the spread is substantially close to

zero. Important to note is that it is irrelevant for the spread which asset is winning and which asset is losing. It is clear that when looking for optimal pairs a volatile spread creates trading opportunities, while the mean-reverting character of the assets keeps the spread stable over time.

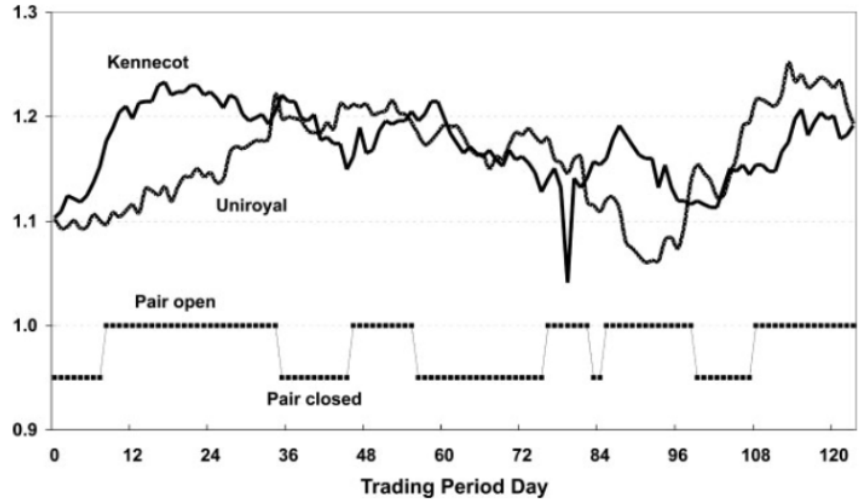

Figure 1
Daily normalized prices: Kennecott and Uniroyal (pair 5)
Trading period August 1963–January 1964.

Figure 2.1: Pairs trading example on a single pair of assets Kennecot and Uniroyal, which shows the daily normalized asset prices of both assets, and the positions taken over a trading period of 6 months [Gatev et al., 2006].

Pairs trading is said to be one of the products of a group of Wall Street quants under Nunzio Tartaglia with backgrounds in physics, mathematics, and computer science. This group was active during the mid-1980s at Morgan Stanley. One of Tartaglia's proteges was David Shaw, the now billionaire hedge fund manager and founder of D. E. Shaw. While already in use for multiple years in the industry, [Gatev et al., 2006] is one of the first works that puts pairs trading in the academic spotlight. The first version of this paper goes back to 1998. Shortly after, [Vidyamurthy, 2004] was published which is the first widespread book describing a quantitative approach to pairs trading. [Elliott et al., 2005], [Do and Faff,], [Engelberg et al.,], are other popular works from around the same period covering pairs trading. Although cited works all cover the subject of pairs trading, different techniques for identifying pairs and trading the spread are researched and discussed. [Krauss, 2017] provides a review and outlook on pairs trading and several popular approaches accompanied by representative studies on these approaches. In the classic literature on pairs trading, two and by extent three large groups of methods exist to identify optimal pairs. The first group is the group of distance approaches, the second group is the

group of cointegration approaches, and the third group concludes everything that can not be counted towards the previous two groups. It is interesting to note how [Do and Faff,] mentions that the classic approaches have become overcrowded and that their profitability is on a downward trend. In the following subsections we cover the three different approaches. After pair selection, different methods are applied to optimally trade the pairs. A final subsection covers some of these execution methods and their advantages and disadvantages.

2.1.1 Distance

This collection covers all methods that determine co-moving assets based on a distance metric. The distance metric is used to measure the difference in normalized asset prices at every point in time of the covered historical period. This data is then used to create a single metric that describes how close two assets move together. An example of this comes from [Gatev et al., 2006], where the sum of Euclidean squared distances (SSD) is used as the co-moving metric. The big advantage of this method is that it is straightforward to build and to execute. However, the method has some major drawbacks. A first drawback is that the use of this method is analytically less optimal given a pairs trader is looking for pairs that would maximize profit under a pairs trading strategy, while the most optimal pair of the SSD approach has a score of zero. This means that the most optimal pair according to this method would not provide any trading opportunities at all. Despite these drawbacks, Gatev's research still reports a positive return. [Do and Faff,] expanded on the work of Gatev, by refining the selection criteria to improve pairs identification by only allowing matching based on Fama-French industries and by expanding the historic data in scope. The top portfolios incorporating these industry restrictions also turned out to be profitable, even when including trading costs, and the study furthermore supports Gatev's findings, helping establishing pairs trading as a capital market anomaly. [Chen et al., 2019] also builds on the same data and time frame as Gatev. However, they opted for the Pearson correlation between historical prices to determine optimal pairs.

2.1.2 cointegration

Instead of looking at the spread as the distance between prices, this group of methods focuses on cointegration between time series. The idea of cointegration assumes that two separate variables that are non-stationary (like two asset prices) can be in equilibrium if there exists a combination of the variables that is stationary, otherwise any deviation from the equilibrium will not be temporary. This method assumes that co-moving asset prices are co-integrated. [Vidyamurthy, 2004] is one of the most cited works regarding pairs trading through cointegration. The research describes the selection of pairs as a two step process. The first step is making a pre-selection of pairs based on fundamental or statistical characteristics of the pairs. The second step

is testing the pre-selected pairs on cointegration by means of a cointegration test, which, for their research, is an adjusted version of the Engle-Granger test. However, there exist other cointegration tests. [Rad et al.,] lays out a step-by-step empirical application of a cointegration approach on the same data used in [Gatev et al., 2006]. The pre-selection consists of using the sum of Euclidean squared differences over a one year period. We're mostly interested in the pairs that have a low SSD value. From the SSD ranking, only the 20 best pairs are retained that are also co-integrated according to the Engle-Granger test. While the cointegration test is more statistically advanced compared to the distance method, there are several disadvantages. First, the pre-selection, which can be a manual exercise, makes it more difficult to work with a large asset universe. Second, a pre-selection method like the SSD pre-selection creates a clear bias. By looking at this ranking, it makes sense that much of the same pairs are selected when comparing to the distance method. To avoid this bias, more elaborated, end-to-end cointegration-based methods were introduced in [Huck and Afawubo, 2015], and [Caldeira and Moura,]. Next to the pre-selection bias, the cointegration approach has the same problem as the distance approach given the selection of pairs only focuses on the co-movement and not on the potential trading opportunities.

2.1.3 Other

Other than the distance approach and the cointegration approach, different approaches have been researched. [Xie et al., 2014] proposes a copula-based approach as a generalization for the distance method. The copula method is an effective tool to model the connection between individual marginal distributions and their joint distributions. According to [Xie et al., 2014], this method brings two main advantages. The first advantage is the separate estimation of marginal distributions of individual variables. The second advantage is that different dependency structures between variables can be modeled. Their research leverages these advantages to identify optimal pairs for pairs trading through the distribution of asset prices. [Han et al., 2023] and [Sarmento and Horta, 2020] provide a modern machine learning approach to pairs trading by leveraging unsupervised learning techniques like clustering in order to generate a pair selection. The clustering can be both fundamental (industry, financials, ..) and statistical (distance, distribution, ..). [Huck, 2009] also reverts to machine learning in order to forecast returns for each asset, using Elman neural networks. Based on the forecasts a ranking of pairs optimal for trading is generated and the top pairs are selected for trading. [Avellaneda and Lee,] proposes an approach based on principal component analysis (PCA) presenting positive returns.

2.1.4 Trading

Once the optimal pairs for pairs trading are identified, it is key to optimally trade the spreads to maximize the profit from the trading strategy. Different methods have been

applied for the strategies mentioned above. In [Gatev et al., 2006] and [Do and Faff,] trades are opened when the spread diverges by more than two historical standard deviations and closed upon mean-reversion or at the end of the trading period. Figure 2.1 shows the positions taken based on this strategy for a given pair of assets. [Jacobs et al., 2014] tested a variant of this strategy where they found that pairs opened on days where a high quantity of unexpected new information was provided to the market were more likely to converge and deliver profit. On timing [Huck, 2015] found that timing volatility does not improve performance of the trading strategy. The advantage of this strategy is that it is simple and easy to execute. However, there are multiple disadvantages. Firstly, there is a chance of missing out on potential profits if the spread continues to diverge before conversion start. Secondly, the spread between two assets might never fully converge before divergence takes place again. Hence, the position will stay opened and again potential profits are missed. [Vidyamurthy, 2004] proposes a more sophisticated approach through a multiple thresholds design, where with counting of cross-overs and regularization, multiple thresholds are found. However, this approach also considers a one technique fits all. [Elliott et al., 2005], [Cummins and Bucca, 2011], [Bertram, 2010], model a spread as a symmetric Ornstein-Uhlenbeck process and either determine an analytical solution (Bertram), or develop a threshold solution based on the time series structure of the spread.

2.2 Signatures

2.2.1 Rough Path and Signature

In order to capture the time series of asset prices in their full extent and to be able to extend the traditional framework of working with time series for pairs trading to a more advanced level, we rely on the theory of rough paths. The theory of rough paths is a mathematical framework described in [Lyons, 2014] that tries to capture and refine the interaction between highly oscillatory and non-linear systems. Often time series cannot be described by a smooth derivable function, but require a more advanced approach. From a mathematical perspective, the theory of rough paths tackles the challenge of describing a smooth but potentially highly oscillatory path. While first applications appeared in automated recognition of Chinese handwriting and extending Ito's theory of stochastic differential equations (SDEs), more recent applications can be found in finance related topics [Gyurkó et al., 2013], [Futter et al.,]. This makes sense given that assets prices and asset returns can be interpreted as a rough path. In order to be able to capture all required information from such a rough path we will rely on the rough path theory and more specifically on signatures, which is a tool developed to support the framework of the rough paths theory. The signature structure provides a hierarchical interpretation, where the low-order components of the signature capture broad path attributes and where high-order components potentially reveal intricate characteristics of the path. In short, one can say that a signature is an infinite vector that describes both the

temporal and geometric properties of a path.

More formally we can describe the signature transformation of a path $X : [s, T] \to \mathbb{R}^d$ between fixed time s and t with Equation 2.1. Let $\mathbb{T}((\mathbb{R}^d)) := \otimes_{k=0}^{\infty}(\mathbb{R}^d)^{\otimes k}$ represent a tensor algebra space, allowing a comprehensive representation of the signatures of $\mathbb{R}^d$-valued paths, then

$$\mathbb{X}_{s,t} = (1, \mathbb{X}^1_{s,t}, ..., \mathbb{X}^n_{s,t}, ...) \in \mathbb{T}((\mathbb{R}^d)) \tag{2.1}$$

where the n-th order of the signature is defined by Equation 2.2, called the n-th order iterated integral,

$$\mathbb{X}^n_{s,t} = \int_{s<u_1<...<u_n<t} \int dX_{u_1} \otimes ... \otimes dX_{u_n} \in (\mathbb{R}^d)^{\otimes n} \tag{2.2}$$

which results in signature $\mathbb{X}^{\leq\infty}_{0,T}$ looking like Equation 2.3.

$$\mathbb{X}^{<\infty}_{0,T} = \left(\underbrace{\mathbb{X}^{\emptyset}_{0,T}}_{=1}, \underbrace{\begin{pmatrix} \mathbb{X}^1_{0,T} \\ \vdots \\ \mathbb{X}^{\mathrm{d}}_{0,T} \end{pmatrix}}_{\mathbb{X}^1_{0,T}}, \underbrace{\begin{pmatrix} \mathbb{X}^{11}_{0,T} & \cdots & \mathbb{X}^{1\,\mathrm{d}}_{0,T} \\ \vdots & \ddots & \vdots \\ \mathbb{X}^{\mathrm{d}1}_{0,T} & \cdots & \mathbb{X}^{\mathrm{dd}}_{0,T} \end{pmatrix}}_{\mathbb{X}^2_{0,T}}, \dots \right) \tag{2.3}$$

The signature of a path defined in Equation 2.1 can be truncated at any finite order $N \in \mathbb{N}$. The truncated signature is given by Equation 2.4. The truncated signature preserves the first $\frac{d^{N+1}-1}{d-1}$ iterated integrals. The factorial decay of neglected iterated integrals ensures a minimal information loss due to truncation [Gregnanin et al., 2023].

$$\mathbb{X}_{s,t} = (1, \mathbb{X}^1_{s,t}, ..., \mathbb{X}^N_{s,t}) \tag{2.4}$$

.

Equation 2.4 corresponds to the monomials up to order N of the path X. Indeed, as we know that for scalars the polynomials, the power series X^n, allow us to approximate Lipschitz smooth functions by means of Taylor expansion [Patrinos, 2023], Equation 2.2 offers a non-commutative equivalent for tensors. This means that when X is not a scalar but a path, classical products in the exponential function are replaced by tensor products. The other way around, if X would be a one-step one-dimensional path from 0 to T assuming X_T, i.e. a scalar, the tensor products in Equation 2.2 could be replaced by simple scalar products and the n-th order signature would correspond to the n-folded integral on dX_t. The n-folded integral on dX_t would just be the area of the n-dimensional simplex, i.e. $X^n_T/n!$, which leads us to conclude that indeed expression 2.2 brings us to the classical Taylor series for the exponential function. By means of the Stone-Weierstrass theorem, one can then show that the universal non-linearity of signatures (applying linear functionals to

signatures to approximate non-linear functionals on paths) is nothing but the Taylor approximation equivalent for paths rather than scalars [Lemahieu et al.,].

2.2.2 Geometric interpretation

From the definition given above, we can illustrate how the signature is a geometric description of your path. To show this, we investigate the lower order signatures of a two dimensional path. The first order signature is calculated by Equation 2.5, which represents the trend of the time series.

$$\mathbb{X}^1_{s,T} = \int_{s<t<T} dX_t = X_T - X_s \tag{2.5}$$

Each element is the increment of the path on the corresponding axis over the time interval $[s, T]$. They denote the displacement of the given path either vertically or horizontally. Figure 2.2 is a representation of the first order signature of two different volume profiles. The figure shows how the first order signature only relies on the beginning and the end point of the path. This means that for different paths with the same coordinates for beginning and end point, the first order signature is the same.

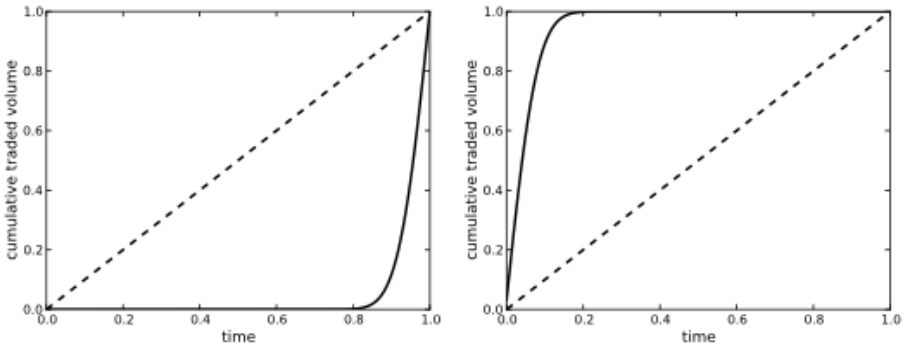

FIGURE 2.2: The figures shows the first order signature for a back-loaded(left) and front-loaded(right) volume profile. The first order signature for both volume profiles is the same given their start and end point is the same [Gyurkó et al., 2013].

The second order signature of a two dimensional path is the two-fold iterated integral of the path, which contains four elements described by Equation 2.6.

$$\begin{aligned} \mathbb{X}^{i,i}_{s,T} &= \int_{s<t_2\leq T} \int_{s<t_1\leq t_2} dX^i_{t_1} dX^i_{t_2} = \frac{1}{2!}\left(X^i_T - X^i_s\right)^2, \\ \mathbb{X}^{i,j}_{s,T} &= \int_{s<t\leq T} \int_{s<t_1\leq t_2} dX^i_{t_1} dX^j_{t_2}, \end{aligned} \tag{2.6}$$

Where i and j indicate the dimensions, either 1 or 2, The first of the two Equations from 2.6 is proportional to the square of the increment. The second equation can

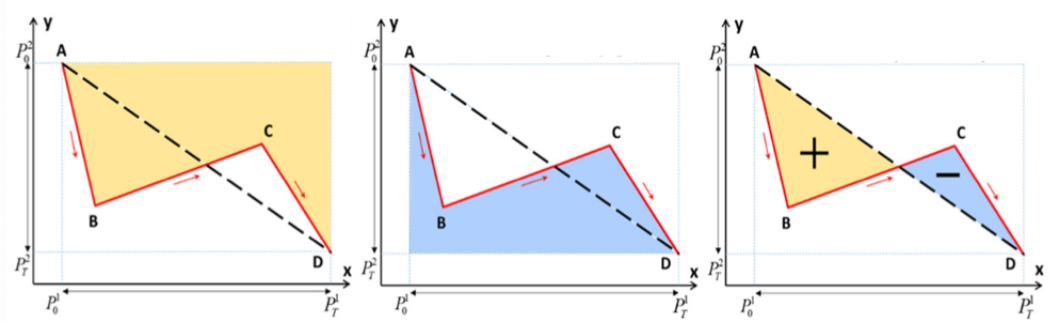

Figure 2.3: This figure shows the areas that are described by the 2-fold iterated integral of a 2D path where the dimensions are different. The third figure shows the difference between these integrals, which provides you with the Levi area [Gyurkó et al., 2013].

be explained by looking at Figure 2.3, which shows the surface for $\mathbb{X}_{s,T}^{i,j}$ and $\mathbb{X}_{s,T}^{j,i}$ starting from the left. The third graph in the figure shows the Levy area, which is the difference of the first and the second iterated integral.

For a two dimensional path, the interpretation of a k-fold integrated integral with k larger than 2 is not trivial anymore, and cannot be shown as a surface. However, one can extend the logic to a three dimensional path where the one, two, and three fold iterated integrals are units of displacement, area, and volume. This provides an intuition of how a signature can be used in order to describe a path. While the first and second order can be easily interpreted, the higher order signatures still carry information that can be useful when analysing the path for modeling purposes.

2.2.3 Signature properties

Given the intuition of a signature, we can elaborate on the attractive properties of signatures that can be used for modeling purposes. Signatures have some interesting characteristics like Chen's identity, invariance under reparametrisation, linearity, uniqueness, and universality. Uniqueness and universality are the two properties of main importance to our research. In particular, the uniqueness property highlights the relation between paths and their signatures; the truncated signature of a path is regarded as a projection of the path to a lower dimensional space. This translates an infinite dimensional time series into a finite dimensional vector describing the temporal and geometrical properties of the path. The universality property, allows us to prove that the lead-transform and the lag-transform both preserve the signature of data streams [Gyurkó et al., 2013]. Furthermore, the universality principle states that linear functionals on the signature are dense in the space of continuous functions on compact sets of paths, which allows to approximate any non-linear functional in the time series space as a linear functional in terms of the signatures of the time series

[Futter et al.,]. In other words, the uniqueness property indicates a unique relation from the path to the signature and the signature to the path for some underlying distribution that generated the path, while the universality property allows us to develop a closed form linear expression in the signature space describing a nonlinear relation between paths.

2.3 Reinforcement Learning

Optimal trade execution is widely studied in academia and certainly in industry given it determines for a large part the profitability of a trading strategy [Deng et al., 2017], [Zhang et al., 2019], [Liu et al., 2018], [Stefan Jansen, 2020]. Provided one is able to select an optimal pair for pairs trading, the actual profit is made by taking and exiting a position on the spread at the right time. Even though the problem in our research is reduced to trading only a single spread, there are still complex and unknown market dynamics that pose a significant challenge for the development and execution of an optimal trading strategy. These unknown dynamics in a live market make reinforcement learning (RL), with its interactive and model free nature, an attractive method for learning an optimal trading strategy [Karpe et al., 2020]. The first part of this section provides an introduction on reinforcement learning and its theoretical basis. The second part of the section focuses on methods to solve the RL problem.

2.3.1 Introduction

In reinforcement learning one models the problem at hand as a Markov Decision Process (MDP). A Markov Decision Process provides a mathematical framework for modeling decision making problems consisting of three main elements. The first element is a set of states $\mathbb{S}$, where a state is a description of the problem environment. The second element is a set of available actions $\mathbb{A}$, which are the possible actions that can be taken by the decision maker or agent. Every action will trigger a reaction from the environment in the shape of a reward R, which is the third element. Figure 2.4 provides a schematic overview of the interaction between the agent and the environment. The figure shows how action A_t triggers a reward R_{t+1} and potentially alters the environment to a new state S_{t+1}.

The goal of reinforcement learning is for the agent to learn an optimal policy $\pi^*(a|s)$ that indicates which action a to take in each state s in order to maximize the total reward during an episode. An episode is a period consisting of different time steps and can have a natural ending (a game of chess) or an artificial ending (100 trading days). This means that the action decided by the policy should not only take into account the direct reward of this action, but also how this action will influence future rewards. E.g., one can take a pawn in chess but because of that move and direct reward lose the game several moves later. In order to know which action to take at each state in order to improve the outcome, the agent requires

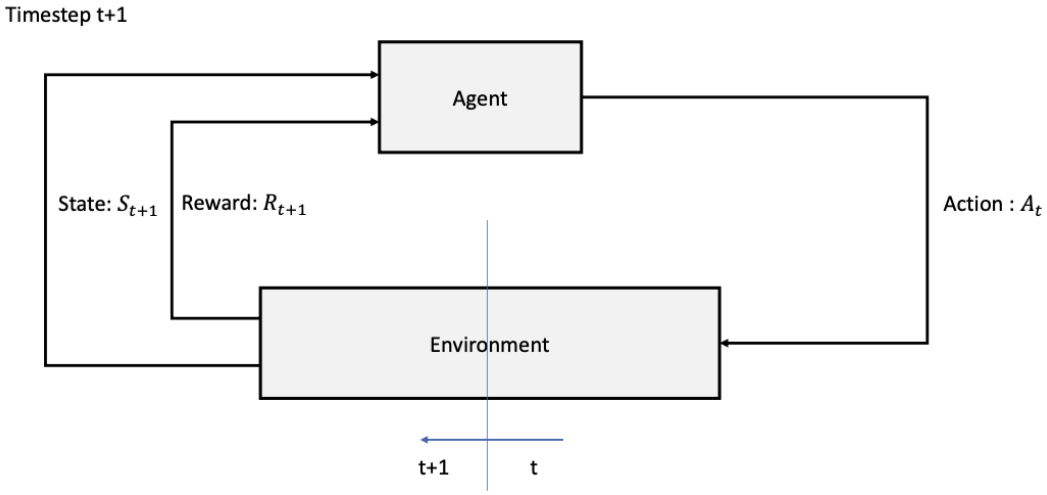

Figure 2.4: The concept of reinforcement learning for the step from time t to time t+1, where the agent interacts with the environment through an action, and the environment changes given the agent a reward and a description of its new state

knowledge about the state in the form of a value for the state or the state-action combination. These values are unknown to the agent at the start of the learning process and are updated throughout. Based on these updated values, the original policy of the agent is adjusted to make it more optimal and to maximize reward. This process of updating the knowledge about the states and updating the policy is called Generalized Policy Iteration (GPI) [Sutton and Barto, 2018]. Typically, the value of a state is defined by a value function $v(s)$, while the value of a state-action pair is defined by action-value function $q(s, a)$. Equation 2.7 and Equation 2.8, define these functions respectively. The value of a state s is the expected value of the sum of discounted future rewards G_t given the current state under current policy π, while the value of a state-action pair (s, a) is the expected value of the sum of discounted future rewards given the state and the action following a policy π. γ is the discount factor, has a value between zero and one, and is a measure for how much future rewards are values compared to the immediate reward of an action.

$$v_\pi(s) = E_\pi \left[G_t \mid S_t = s \right] = E_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s \right] \tag{2.7}$$

$$q_\pi(s, a) = E_\pi \left[G_t \mid S_t = s, A_t = a \right] = E_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a \right] \tag{2.8}$$

These equations can be rewritten as Equation 2.9 and Equation 2.10 respectively, which are known as the Bellman equations. s' represents the next state, r the direct reward, and the function $p(s', r|s, a)$ represents the probability of reward r and next state s' given the current state s and taking action a and is called the dynamics of

the environment. Knowing these values, the policy can be improved to be improved to better approximate the optimal policy.

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')] \tag{2.9}$$

$$q_\pi(s, a) = \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')] \tag{2.10}$$

2.3.2 Solution methods

There exists a closed form solution for the optimal policy and optimal value function through dynamic programming and the Bellman equations given the dynamics of the environment are fully known. In this case the state and state-action values can be calculated from start to end. However, in most real-world applications of reinforcement learning, these dynamics are unknown. One solution could be to approximate these dynamics and continue using dynamic programming, however, this turns out to be very difficult given all the different aspects that can be relevant for an environment. Two different solutions are provided by reinforcement learning literature that use the recursive relationship in the Bellman equations. The first method to determine the optimal action-value functions and the optimal policy is the Monte Carlo (MC) method. The Monte Carlo method only requires experience and does not require any knowledge of the environment's dynamics. In short, the MC method simulates entire episodes and uses the experience to update the action-value function for every state and every action. This method is very powerful given no dynamics are required, however, requires to simulate entire episodes to learn the action-value functions and to adjust the optimal policy. The second method available is Temporal-Difference (TD) learning. Temporal-difference learning is a combination of learning by experience of the Monte Carlo method and learning through the model of the Dynamic Programming method, by updating estimates based in part on other learned estimates, without having to wait for the final outcome of an episode. A first advantage of the Temporal-Difference method is that just like the Monte Carlo method there is no model required for the dynamics of the environment as the method learns from experience. A second advantage of the TD method is that the method can be naturally implemented in an online setting. Where for MC entire episodes were required, action-values are updated at every step. This is an attractive advantage when working in financial markets where the agent would need to trade in a live market. A final advantage of TD over MC is that Monte Carlo needs to ignore or discount some of the episodes where experimental actions were taken, which can slow down the learning of the agent. TD methods however, are less susceptible to these problems as they can learn from each state transition regardless of the actions that are taken later. Figure 2.5 provides a schematic overview on how value functions are updated based on the different methods [Sutton and Barto, 2018].

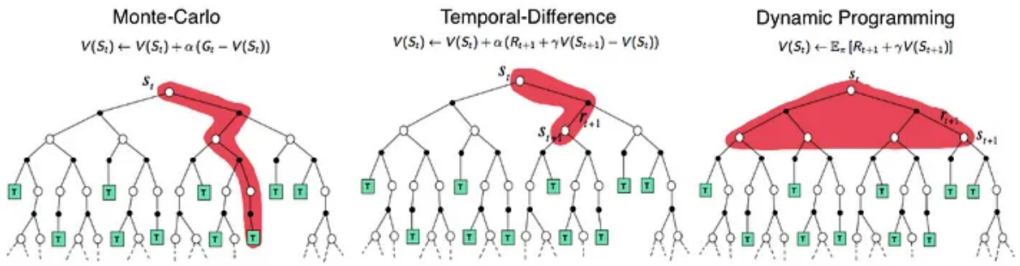

Figure 2.5: The figure illustrates on what basis the value of a state is updated based on three different methods. Left, the MC method, bases the update on one entire episode. Middle, the TD method, bases the update on one step of an episode. Right, the DP method, bases the update on knowing the environment dynamics and calculating the expected value [Lilian Weng, 2018].

A distinction has to be made between On-policy and Off-policy TD methods. The difference between On-policy and Off-policy is in the policy control, how the policy is adjusted with each adjustment of the state-action values. For On-policy TD, the same policy that is used to generate data of state transitions, is also the policy that is being adjusted into the direction of the optimal policy. For Off-policy TD, the policy control happens separately, meaning there is a policy to generate the state transition, and a policy that is being updated to approximate the optimal policy. Off-policy TD is better known as Q-learning. An advantage of Q-learning is the fact that the agent can execute more exploratory actions and even suboptimal decisions, from which it can learn valuable lessons, which would stay undiscovered in On-policy TD. Algorithm 5 provides a pseudo-algorithmic description of Q-learning.

Algorithm 1: Q-learning (off-policy TD control) [Sutton and Barto, 2018]

Initialize Q(s,a), for all $s \in \mathbb{S}^+$, $a \in \mathbb{A}(s)$, arbitrarily

for *Each episode* **do**
  Initialize S
  for *Each step of the episode* **do**
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Take action A, observe R,S'
    Update: Q(S,A) $\longleftarrow$ $Q(S,A)+\alpha$ [R + γmax_a Q(S',a) - Q(S,A)]
    S $\longleftarrow$ S'
    until S is terminal
  end
end

In words, the state-action values are arbitrarily chosen at first. Then for each step in every episode, an action is chosen according to a policy like the ϵ-greedy policy. For this action the transition is observed and based on the observed values, the state-action value for state S and action A is updated based on the optimal policy and the previous estimation of the state-action value.

2.3.3 Reinforcement learning for trading

The previous parts introduced reinforcement learning in a discrete setting where set of states is a closed set of discrete states. In such a case the Q(S,A) values can be tabled for all state-action pairs and updated for each occurrence in the episode. However, when in a live trading setting, the set of states is no longer a closed set of discrete states, but becomes continuous. This change requires a solution where state-action values can no longer be tabled. Instead of the tabled values, the continuous case requires a state-action function approximation based on a set of parameters. Instead of updating the tabular values while learning, the parameters of the function approximation are changed. The function approximation of the action-value function is a classic function approximation machine learning problem. Polynomials, Fourier based, and based on coarse coding or tile coding, are some feature construction methods for linear models that can be used for the action-value function approximation. Furthermore there exist memory-based or kernel-based function approximations [Sutton and Barto, 2018]. However, a more popular approach is function approximation with a nonlinear function and more specifically with Artificial Neural Networks. When function approximation is done with multilayered neural networks, it is generally called deep reinforcement learning. Deep reinforcement learning has caught the attention of both academics and industry for trading given its strong results [Zhang et al., 2019], [Karpe et al., 2020], and will also be the focus of our research.

This research intends to contribute to the literature on applications of signatures in finance, and more specifically contribute with pairs selection with signatures. Furthermore, using deep reinforcement learning, the thesis adds to the abundant literature on the application of deep reinforcement learning in trading, and more specifically for optimal execution.

Chapter 3

Data

In this chapter, the dataset is discussed that is used to create the asset universe and the data transformations required to generate the final spreads that are used for pair selection and trading. The code that is used to clean the data and transform the data to spreads is provided in the Python class `DataLayer` under Appendix A.1.

3.1 Asset universe

When using a dataset there are different parameters to take under consideration. The first consideration is the data availability. Preferably, for all assets in the dataset, there is plenty of data present to benefit training and testing. Regarding data availability, indices like S&P 500, Russel 3000, FTSE 100, DAX 50, groups of euro ETFs or US ETFs, and others are good candidates given they are well established indices with mostly large and known constituents. A second consideration is the company size in terms of market capitalization. In general the companies with a larger market cap provide a more stable stock price compared to their smaller peers. For this reason we drop indices like the Russel 3000. A third consideration is the potential to create a fundamental understanding of the output. Given a pair of two known assets, it allows to generate an idea of the fundamentals that drive the mean-reversion of these two assets. For ETFs, finding the fundamental drivers of the mean-reversion is much harder as you need to understand the composition of the different ETFs and how they interact with each other. For this reason we drop the groups of Euro ETFs and US ETFs. A fourth and final consideration for determining the dataset is the number of assets that are included. Ideally, there are enough datasets to generate many potential pairs, however given the number of pairs corresponds to $(n * (n-1))/2$, with n the number of different assets, the computational effort can increase fast. This consideration excludes the S&P 500 given 500 assets would provide 124'750 different pairs. Considering all of the above, the FTSE 100 is chosen as the dataset providing the universe of assets for our analysis.

3.2 Data cleaning

The initial data set contains stock price data for 288 assets starting from January 1st 2000, until November 29th 2023. Several steps are taken to filter the data. The goal of filtering the data is the filter for those assets for which there is a maximal data availability. The first step is the removal of each asset for which there are `NaN` values. These assets are mostly assets for which there is data missing over the given period either because they weren't listed yet or they defaulted at a point in time. The second step is making sure that the assets that are still included show a realistic price path. Some of the price paths that are being removed are penny stocks, which show very unreliable paths, and assets that got delisted at some point showing a flat price path. After applying filters for these assets, there are 193 assets left.

Given the research is mostly interested in the spread between asset prices, the next step consists of normalizing the price data. The price data is normalized by dividing the prices of every asset by its first available price value. This normalization eliminates the influence of the absolute value of the prices when calculating the spread. Once the normalized price values are calculated, the assets are ranked according to the sum of their normalized price values with the highest value first and the smallest value last. This allows to check and decide which price paths should be used in the analysis. Eventually, 100 assets are kept with the largest sum in prices, given they have a more realistic movement over the 20 years of data that are analysed. Figure 3.1 shows the normalized price path of 10 asset from the final asset price universe used in this work. The figure shows some of the major events that happened in the time interval being considering like the crash of 2008, COVID in 2020, and the post-COVID rally in 2021. These events will also be included when training the different models that are used and could present a challenge for the considered models.

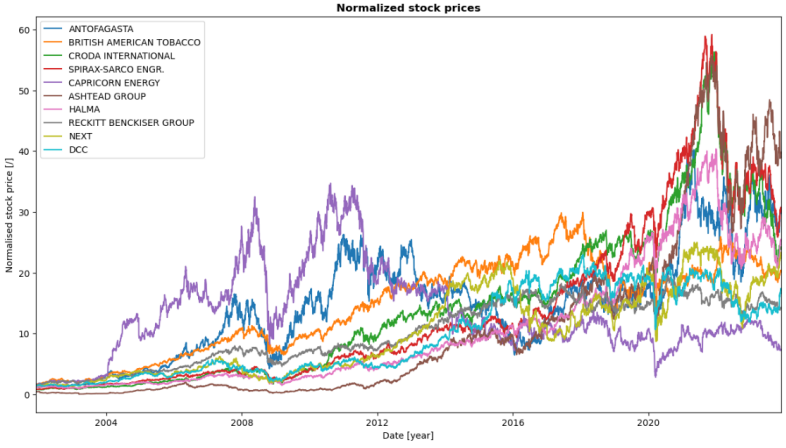

Figure 3.1: Normalized stock prices for 10 assets of the asset universe over the complete time interval.

3.3 Spreads

With the universe selected and the normalized price data calculated, one can calculate the spread values at every point in time as the absolute value of the difference between the normalized price values of two assets. The absolute value is used as for trading the spread it is less relevant which asset is the winner and which asset is the loser. Furthermore, it helps calculations when determining the profit and loss of a pairs trading strategy on a given spread, which is further explained in Chapter 4. Figure 3.2 shows the spread between Capricorn Energy and Pennon Group over the entire time interval. Given the research mostly focuses on trading over a specific time interval, it is more interesting to zoom in on the data for better understanding. Figure 3.3 provides the spread between Capricorn Energy and Pennon Group over the last 100 trading days on the left and the separate normalized asset prices of the respective assets on the right. Analyzing the entire data horizon, the spread is initially small. This is due to the normalization of asset prices and the general market growth over time. Otherwise the spread shows to be volatile. Analyzing 100 trading days, there seems to be a mean-reverting behaviour between the two assets providing potential trading opportunities when trading the spread.

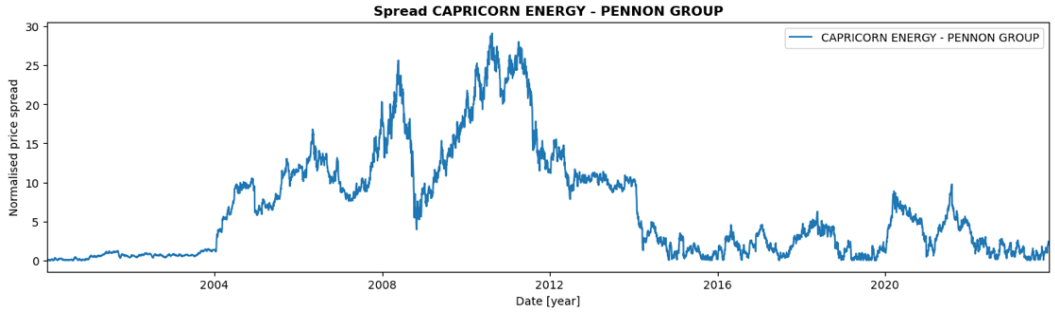

FIGURE 3.2: Spread between Capricorn Energy and Pennon Group over the entire data horizon.

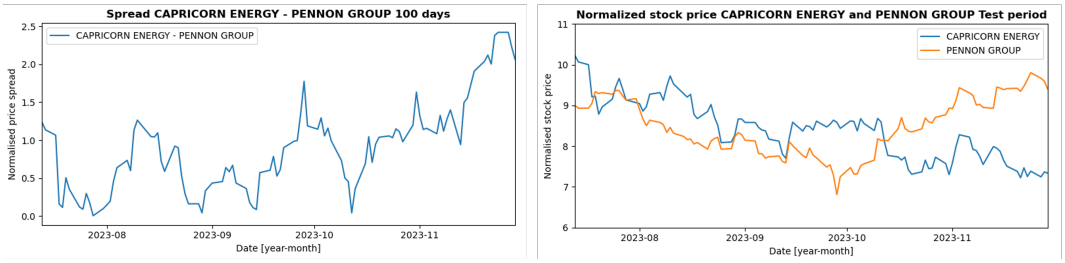

FIGURE 3.3: Left: spread between Capricorn Energy and Pennon Group. Right: Normalized stock prices for Capricorn Energy and Pennon Group.

Chapter 4

Methodology

The following chapter provides a detailed description of the methods applied in order to ascertain trading pairs and to determine an optimal trading strategy. The first section covers the calculation of the profitability of pairs trading on a specific spread. This method is used throughout the research as a proxy for the maximum profitability of pairs trading on a spread over a certain time period. Once the profitability is established, the method further describes how the profitability can be associated with the signature of a spread. The second section of the chapter describes the technique that is used to determine the profitability of pairs trading on a spread in the future, relying on the expected signatures and the modeled profitability from the first section. In the third section, the optimal execution algorithm is described. The main parts of the execution consist of the trading environment, the reward function, the design of the reinforcement learning agent, and how to train the agent. Lastly, a benchmark method is described in order to benchmark the pairs selection method with expected signatures against a classic pairs selection method and to benchmark the optimal execution method with reinforcement learning against a classic execution method.

4.1 Profit and Loss

The description of the data in Chapter 3 allows to build and explain a methodology to determine the profitability of a pairs trading strategy. In Chapter 3, the spread between normalized prices is calculated as the difference between two normalized prices at every time point. These spreads are all the information the trader receives when it needs to learn how to trade. Firstly, a method is described to calculate the maximal profitability of pairs trading on a spread. Secondly, based on the given spreads and the proxy for the profit and loss of the pairs trading strategy, a model is established that connects the signature of a spreads to the potential profit of the specific spread. This model is called the PnL signature model throughout the text. The code for calculating the PnL and creating the PnL signature model can be found in the Python class `PairSelection` under Appendix A.2.

4.1.1 Profit and Loss calculation

When trading a spread the goal is to maximize the profit by taking a position on the spread when the spread is at its maximum and relying on the mean-reverting character of the pair forming the spread such that the spread declines and the position can be exited at a minimum. Based on this logic, a proxy for the PnL is calculated by finding the local maxima and local minima ([Panos Patrinos, 2023]) and taking the sum of the differences of each maxima and the next minima. Algorithm 2 provides a structural overview of how the PnL of a spread over a specified time interval is calculated. The idea is that the trader takes position at the maximum and exits at the next minimum before taking position again at the following maximum. This method does not require the spread to fully revert to zero before exiting the position, which can be the case for many of the spreads where there might be a constant minimum distance during the whole training period. Furthermore, and most importantly, the calculated PnL provides an indication of the tradability of a specific spread, with a high PnL indicating a large volatility in the spread, and a low PnL indicating a low volatility in the spread. Figure 4.1 shows the positions taken and positions excited for a given spread following the PnL logic. Important to note is that in this approach transaction costs are being neglected as the only purpose is to find a PnL value that indicates tradability and profitability. When including transaction costs, some of the smaller transactions might not be profitable. This will be for the trading agent to learn. In the figure a spread over a period of 100 days in analysed, given this is the general length of trading periods found in literature. However, the code is build in a modular way such that one can adapt the trading window.

Algorithm 2: PnL calculation

Step 1: List local maxima and minima for a spread
for *each max and min* **do**
  Add to list of extrema sorted by time index
end

Step 2: Calculate profits
profit = 0
for *each max* **do**
  Calculate difference with the next minimum
  Add difference to profit
end

return profit

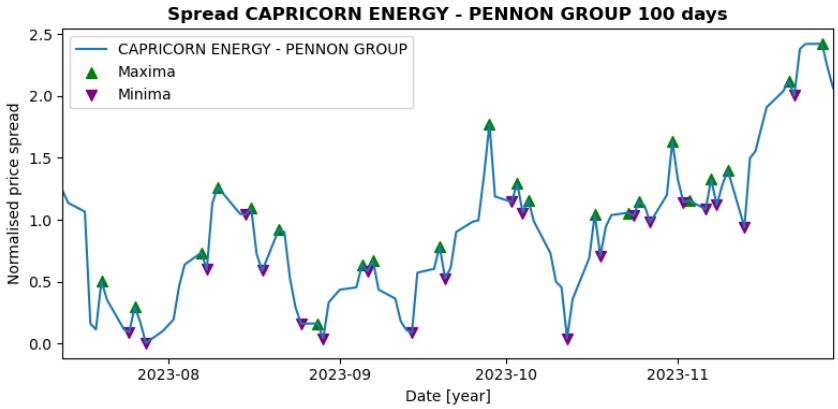

FIGURE 4.1: The local maxima and minima for the spread on Capricorn Energy vs. Pennon Group over a time interval of 100 days. Opening position at the maxima and closing the position at minima, neglecting transaction costs would result in a maximum profit from trading this spread. This approach delivers a total PnL value of 7.21 for this trading window.

4.1.2 Profit and Loss Signature

Given the characteristics of signatures, a signature can be understood as a noise filter, where only the relevant features of the respective path are translated into the signature elements. A higher truncation order of the signature will lead to a lower information to noise ratio of the additional signature elements. From this angle, the signature can be described as a filter and this filter can be used when estimating the PnL of the expected signature, which is discussed in the next section. In order to be able to calculate the expected PnL based on the expected signature, a model is required that links the signature of the path to the PnL proxy of that path. The proposed model in this research is a regression model that takes the signature elements of the path as features and the PnL proxy of that path as output value. The regression model is a Ridge regression model making use of a robust scaler for the features before fitting the model. In Subsection 4.2.2, the paper elaborates on the Ridge regression model and why a scaler is used as pre-treatment for the features. The model provides a direct link between the description of the geometry of a path through signatures and the potential trading PnL from trading the path. Algorithm 3 provides a structured description of how the model is established. With the goal to determine a relation between the path of a spread and its potential profit, the model is fitted over all pooled spreads and their PnL value and not for each spread separately. The goal is to find the driving features of a path behind the profit or loss on a spread.

In Algorithm 3, m represents the truncation order of the signature, $\mathbb{X}^m_{window}$ represents the m-order signature of the spread over a specified window, PnL_{window}, is the calculated PnL of the spread over a specified window, and α represents the

Algorithm 3: PnL signature regression

Step 1: Create regression DataFrame
for *Each trading window of every spread* **do**
  Add PnL_{window} ; $\mathbb{X}^m_{window}$ to DataFrame
end

Step 2: Fit regression model
X = signature element values
y = PnL values
X-scaled = scaled signature values using a robust scaler
Fit model RidgeRegression(X-scaled, y, α)

return model

regularization factor for the Ridge regression.

4.2 Expected Signatures

Once the above described model for calculating the profit and loss on the signature of a spread over a specified window is fitted, the potential profit of spread over a future period is determined by calculating the expected signature of the spread and by using the previously fitted model to predict the expected PnL. First, in this section, the expected signature is defined in the setting of this research. Second, the model and algorithm to calculate the expected signature is explained. Last, with the expected signature an estimation of the future PnL of a spread is determined.

4.2.1 Definition

There exists light variations when define the expected signature of a time series [Gregnanin et al., 2023], [Lyons, 2014], but a broad definition is given by Equation 4.1, with parameters p, q, n, m, where p and q represent the time steps being predicted and the historic time steps used for the prediction respectively, while n and m represent the signature order of the expected signature and the signature of the historic time series respectively. The function f then defines a relation between the signature of the historic spread and the expected signature of the future time steps of the spread.

$$\mathbb{X}^n_{t+1,t+p} = f(\mathbb{X}^m_{t,t-q}) \tag{4.1}$$

For this specific research, the expected signature becomes a two-step process. First, Equation 4.2 defines the expected signature as the signature of the predicted spreads,

where S defines the signature function, n indicates the truncation order of the signature, and s_{t+i} indicates the value of the spread at time $t+i$.

$$\mathbb{X}^n_{t+1,t+p} = S^n(s_{t+1}, s_{t+2}, .., s_{t+p}) \tag{4.2}$$

Second, we define a separate relation between the prediction of every spread value and the signature of the historic time steps by Equation 4.3, where the next spread value is a function of the signature of the historic spread values. Function g is defined in the following section. The choice for modeling signatures to spread values to signatures instead of modeling signature to signature values was made to keep the model simple. This approach is in line with [Lilian Weng, 2018] and [Fermanian, 2020]. The goal of the research is to leverage the characteristics and advantages of signatures to a maximum extent, without making use of complicated models and constructions to aid the outcome of the prediction.

$$s_{t+1} = g(\mathbb{X}^m_{t-q,t}) \tag{4.3}$$

4.2.2 Choice of model

Based on Equation 4.2 and Equation 4.3 an algorithm is build to determine the signature of the future spread over a specific trading window. However, the functional g needs to be defined in order to be able to predict the future spread values based on the signature of the historic spreads. As stated earlier, the aim of this research is to articulate the use of signatures for the prediction of financial time series values that can be used in a trading algorithm. To focus on the signature and its strength, the goal is to stay away from complicated models. This moves complicated machine learning models out of scope and draws the attention to regression models. Given the signature of a time series is a vector of numbers, each vector component can be seen as a feature that is part of describing the path. This way each signature can be seen as a set of feature labels which a model g can use in order to predict the next spread value.

In this feature framework, the family of linear regression models offers different possibilities. The standard linear regression model is in that case one of the first options to consider, however, given the nature of the data, this choice of model is not optimal. Firstly, the linear regression model has difficulties handling collinearity between different features. Given that different features belong to the same order of signature, there is a reasonable chance of collinearity between the these features. Secondly, for each order of the signature, there is a constant included due to the time augmentation of the spread data. The concept of time augmentation is discussed further in the text, but this causes some of the values to be constant. The linear regression does not handle these constants well. Thirdly, in the case of an abundance of features, standard linear regression tends to overfit [James et al., 2012]. In order to avoid these problems, g is chosen to be a Ridge regression model. A Ridge regression is known to handle collinearity better compared to linear regression. Furthermore, Ridge regression is a regularization model, which applies shrinkage of

coefficients, which is a penalty applied to the coefficients that can shrink the value of the coefficient close to zero, reducing the relevance of the linked feature. Next to indicating relevance, this form of regularization helps also with handling overfitting in an environment with many features.

A Ridge regression model requires as input the feature values and the output value that is specifically linked with these feature values. In case of the expected signature model, the feature values are the signature values of the m order signature of the previous q spread values, and the connected output value is the spread value at the next time step that is not included in the time series for which the signature was calculated. The optimization equation of the model is given by Equation 4.4.

$$minimize \quad ||y - \beta X||^2 + \alpha||\beta||^2 \tag{4.4}$$

where y is the output value, corresponding to the spread at time $t+1$, X are feature values, corresponding to the signature values of the signature of the previous q spread values, β are the coefficients, and α is the regularization parameter determining the strength of the applied penalty. Values for α can vary between $1e-4$ and $1e6$. A larger value for the regularization parameter will penalize the coefficients more, meaning that less coefficients will play a substantial role in the prediction of the next spread value. The closed solution of the Ridge minimization is given by Equation 4.5. The equation indicates why Ridge regression is more capable of coping with collinearity compared the the standard linear regression. Due to collinearity between features $X^T X$ can become close to singular which can lead to numerical instabilities when calculating the inverse of this matrix product. The regularization constant stabilises this process, making sure the inverse can always be calculated. However, when the regularization constant becomes very large, the predictive quality of the model will go down and the prediction will become more general.

$$\beta_{min} = (X^T X + \alpha I)^{-1} X^T y \tag{4.5}$$

4.2.3 Algorithm

Based Ridge regression model described above, an algorithm can be build in order to determine the expected signature for each spread that is available in the data, separately. A description of the algorithm is provided by Algorithm 4.
The first step of the algorithm creates the labels which are used to train the regression model for each single spread. The features are the signature values of the signature of the considered window. The window contains q spreads and is moved forwards through the data with k steps, with k chosen between 1 and q. With $k = 1$ there is more data available for the regression, but the data set has a low information entropy, meaning the amount of new information available for the model to learn from an additional data point is minimal. However, with $k = q$, and q large, the amount of available labels to train is limited, which will also hurt the model performance. This is why k is chosen between 1 and q. Once the dataframe of labels is build, all data is

Algorithm 4: Expected Signature Algorithm for a Single Spread

Step 1: Create the regression DataFrame
for *each window of size q and next spread value* **do**
| **Add** s_{t+1} ; $\mathbb{X}^m_{t-q,t}$ to the DataFrame
end

Step 2: Define the Ridge regression over the regression DataFrame
X = signature values
y = next spread values
X-scaled = scaled signature values by robust scaler
Fit model: RidgeRegression(X-scaled, y, α)

Step 3: Calculate the expected signature of the next p spreads
for *step i from 1 to p:* **do**
| Calculate signature for $[s_{t+i-q} : s_{t+i-1}]$
| Rescale the new signature with the robust scaler
| Predict s_{t+i} using the Ridge model
| Collect the predicted spreads
end
Calculate the expected signature $\mathbb{X}^m_{t+1,t+p}$ **return** expected signature

$\mathbb{X}^m_{t+1,t+p}$

available to train the Ridge regression model. However, before training the model, the feature values are scaled with a robust scaler. The scaler removes the median of the features and scales them according to the quantile range (defaults to interquartile range between 1st quartile and the 3rd quartile) [sckit-learn,]. It is very common to preprocess features, however outliers can have a substantial influence on the sample mean and variance. The two main advantages of the scaler are the fact that it helps the model to deal with outliers, and that is does not assume a specific distribution in the data.

Once the model is fitted, p future spreads can be predicted based on the signature of the latest q spread values. Given that it is the goal to determine the most optimal pair for pairs trading in the next trading window, the predicted i-th spread is used for forecasting the i+1-th spread. It is relevant to note that during the calculation of the signature of the time series of spreads a time augmentation is required for calculating the signature. The time augmentation changes the array of spreads to a two dimensional path. This step is integrated in a custom function for calculating the signatures and is required to calculate the signature of a 2D path. There exist different approaches for time augmentation, however, we went

with the simple approach of assigning an index to each value, which creates a linear interpolation between the spread values to generate a path for which the signature can be determined [Levin et al., 2013].

4.2.4 Expected PnL

From Algorithm 3 and Algorithm 4 it can be deducted that the signature order of the expected signature m is the same as the signature order of the PnL signature. This is by design, such that the model describing the relation between the potential PnL and the signature of a spread can be used in order to determine the expected PnL from the expected signature. A different option would have been to calculate the expected PnL straight from the predicted spread values for an entire trading period. However, as stated above, the goal of working with signatures it to get rid of the noise and keep the relevant information for pairs trading profitability and such improve the model predictions. By using signatures, one tries to filter out the noise of the prediction of the future spreads.

Once the expected PnL for every spread is calculated, the spreads are ranked from highest to lowest expected PnL. The highest ranking spreads will be the spreads that are of interest for the pairs trading strategy, given the highest returns are expected for these spreads in the next trading period. The research considers the 20 pairs with the highest expected return for the expected signature model and compares these to the actual best performing pairs and the classical benchmark for pair selection, which is discussed later.

4.3 Optimal execution with Reinforcement Learning

The following section elaborates on how reinforcement learning, and more specifically deep Q-learning, can be used for optimal execution of trades on a given spread. While Chapter 2 provided a background on reinforcement learning, the different elements, and the different available techniques, the focus here is on applying the theory in the given setting of our research and designing a trading environment and deep Q-learning agent. The first part of the section focuses on the trading environment, where the possible actions of the agent and the rewards for the agent are defined based on a provided spread. The second part of the section focuses on designing the trading agent. The trading agent's architecture is based on the theory behind deep q-learning and the architecture, parameters, and underlying steps are discussed. The code for the trading environment can be found in the class `TradingEnv` under Appendix A.3.1, while the code for the trading agent can be found in the class `DQLAgent` under Appendix A.3.2. The trading environment was modelled using the gyms package, a well-known package to design environments for reinforcement learning for different types of applications.

4.3.1 Trading environment

The trading environment sets the stage for the agent. Firstly, it decides what information about the environment the agent receives. Secondly, the environment decides which actions are available to the agent. Lastly, and perhaps most importantly, the environment defines the reward an agent receives after every action, which steers the behaviour of the agent.

State

The data available to the environment is technically all the historic spread data up to the present. This means all data from the latest spread to the earliest spread could be provided to the agent should this be required. Only providing the latest spread to the agent might not provide the agent with enough information to make a trading decision while providing a full history to the agent every time might be too much data such that the agent has difficulties processing all the data. The parameter that decides on the amount of information the agent receives is called the lag value in the RL setting of our research. The minimal lag value is 1, while the maximal lag value depends on the amount of data available. A reasonable choice for a lag value is 20, meaning that the state consists out of the 21 latest available spreads, although the code is build so the lag value can be changed for testing purposes. Additionally to the lagged spread values, another important element that needs to be added to the state is the current position of the agent. Just as depicted in Figure 2.1, there are two potential positions for the agent, either the agent has an open position and is holding the spread or the agent does not have a position. A value 1 for the position means the agent is currently holding a position, while a value 0 indicates that the agent is currently not holding a position. Together with the lagged spreads and the current spread, the position of the agent defines the state.

Action space

Through actions the agent is capable of changing the state, and for this change the agent receives a particular reward or punishment. It is clear the the agent is unable to influence the lagged spreads, however the agent can decide at every step about its position. There are three possible actions for the trading agent. Firstly, the agent can take a position, either the agent did not have a position and it enters one, or the agent already had a position, in which case nothing happens. Secondly, the agent can sell its position. In the case that the agent is holding a position, the trader exits the position, while in the case the the agent is not holding a position, nothing happens. Lastly, the agent can decide to hold, which changes nothing to the state. For further clarity, numbers are assigned to the different actions. The available values are zero, one, and two, which correspond to buying, holding, and selling respectively.

Reward function

The reward function is a tool to provide signals to the agent if the effect of its latest action was desired or not to reach the eventual goal which is maximizing the profit when trading the spread. Figure 4.1, which resembles how the PnL is estimated, also provides a proxy for the optimal strategy of the agent. In order to maximize profit, the agent should buy high and sell low. Given this strategy in mind, Algorithm 5 provides a pseudo-algorithmic description of the calculation of the reward. The agent is rewarded for holding a position if the previous spread is higher than the current spread, while the agent is penalized for holding a position if the previous spread is lower than the current spread. Furthermore, the agent is rewarded(penalized) extra, when it exits its position with a profit(loss). This reward function makes the agent conscious about holding a position for a long period and about avoiding a loss at exit. Additionally to the standard reward function, if an action of the agent is executed, an additional transaction penalty can be added. The transaction penalty can be chosen freely, but a realistic value would be 0.2% of the value of the spread that is being traded. The reward function that includes the transaction penalty is given by function `_calculate_reward_transaction_cost()` from class `TradingEnv`. This reward function is not only used to train the agent of interest, but also to rate the other trading methods, given its considerations on the ideal treatment of the positions.

Algorithm 5: Trading environment reward function

```
reward = 0
if agent holds position:
    reward = previous spread - current spread
if agent does not hold position and takes position:
    agent holds position
elif agent exits position it is holding:
    reward += previous spread - current spread
    agent doesn't hold position

return reward
```

Trading

With the state space, action space, and reward function defined, it is possible to trade in the trading environment over a pre-determined trading period. Figure 4.2 provides an illustration of a random agent trading a spread for 100 trading days and the total reward gathered by the random agent. From the figure it is clear that a random agent will only make a profit when it is lucky with the actions taken. There is no systematic way for the random agent to make a profit.

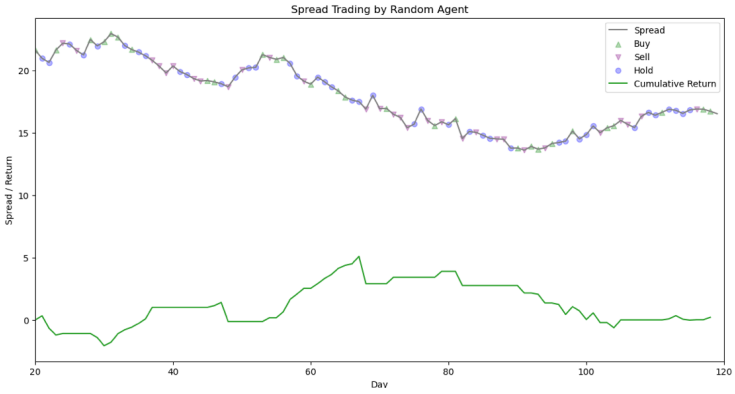

FIGURE 4.2: The figure shows the trading decisions made by a random agent trading a spread and the accumulated return resulting from the actions. The random agent got lucky when buying two times when the spread converged.

4.3.2 Deep Q-Learning agent

Given the trading environment, the goal of the research is to design a reinforcement learning agent that learns to optimally trade a spread of two assets, where optimally trading is defined by the reward function. Chapter 2 explained the goal of a reinforcement learner is to learn the optimal policy $\pi(a|s)^*$ where the agent knows which action a to take for a given state s to maximize the total reward. This policy can be learned through policy control where the policy is updated according to the state-action values $q(a, s)$. The Chapter further explains that there exist different methods in order to determine and continuously update these state-action values and that the Off-policy Temporal Difference method, also called Q-learning, is an ideal method to learn an agent on a live process. This method is of interest to our research, however, the chapter also explains that, due to the characteristics of financial time series, the state space is not discrete and not finite, and for such a case a function approximation is required to learn the state-action values for Q-learning. As a model for the function approximation, our research proposes a deep neural network. The specific characteristics of the neural network model can be found in the model building function `build_model()`, where neural network consists of one input layer, one hidden layer and one output layer. With each additional layer, the number of parameters increases substantially. The combination of the Off-Policy Temporal Difference method and neural networks for function approximation is called Deep Q-Learning (DQL) and is a popular technique when it comes to determining optimal execution in trading. Algorithm 6 provides a pseudo-algorithmic description how such an agent can learn. Some essential building stones required to complete the agent are highlighted in what follows.

ϵ-greedy policy

A first important building stone is the policy applied by the agent in order to be able to learn. Such a policy should enable the agent to exploit what it has already learned to maximize the profit, while the agent needs the freedom to explore other options to see if there are perhaps more preferred strategies by choosing different actions in different situations. Such a behaviour is typically imposed by an ϵ-greedy policy. For a given ϵ between zero and one, if a randomly drawn number between zero and one is smaller or equal to epsilon, a random action is selected. Otherwise, if the randomly drawn number is larger than ϵ, the neural network is used to predict the q-values for every available action given the current state and the action with the highest q-value is chosen. The ϵ-greedy policy is given by Equation 4.6.

$$a = \begin{cases} \underset{a}{\text{argmax}} Q(s,a), & u > \epsilon \\ \text{random action}, & u \leq \epsilon \end{cases} \tag{4.6}$$

With $Q(s,a)$ the state-action value for state s and chosen action a, u the randomly drawn number between zero and one, and with ϵ as a parameter. However, over time the desire to have an agent that exploits more or explores more changes. When the agent starts learning, the agent should explore more to get a sense of the effect different actions have, while near the end of the learning period the agent should exploit more given it has build up more certainty over time. Because of this reason, the value for ϵ is altered during training from a higher value at the start to a lower value near the end of the training. The code provides the flexibility to alter this ϵ decay to optimally fit the training purpose.

Online and offline network

A second building stone of high importance is the availability of a ground truth in order to train the parameters of the neural network, given a neral network is usually provided input-output labels to optimize its weight through gradient descent. Since the parameters of the neural network keep on being updated, the target values for training the parameters will also continuously change. This makes the process of learning the neural network unstable. In order to solve this, a second neural network, identical to the first network, is created. This network is typically named the target network or the offline network, and is used to produce the target q-values, while the original network is usually called the policy network or the online network. When training, the parameters in the online network are updated first for a pre-defined τ number of steps. After these steps, the parameters of the network are copied to the target network, and the process is repeated until all training episodes are done. This trick of using an online and offline neural network stabilizes the learning process.

Experience replay

A final element that is added to the Deep Q-Learning agent in order to improve the accuracy and the convergence speed of Deep Q-Learning is experience replay

(ER). Experience replay stores historic transitions as $(s_t, a_t, r_{t+1}, s_{t+1})$ with s_t the state before, a_t the action, r_{t+1} the reward, and s_{t+1} the state after. These historic transitions are sampled in mini-batches to update the network weights before a new ϵ-greedy action is selected. This improves sample efficiency, reduces autocorrelation of samples, and improves the stability of the neural network [Jansen, 2022]. Algorithm 6 provides a systematic description of the how the agent learns with experience replay.

Algorithm 6: Deep Q-learning with Experience Replay

for *Each episode* **do**
  for *Each step of the episode* **do**
    if $u \leq \epsilon$:
      select random action
    else:
      select optimal action based on the state and the online network
    Execute action and get next state and reward
    Memorize state transition
    Begin Experience replay:
      Sample batch from memorized transitions
      Calculate target Q-values using the target network
      Predict Q-values from the online network
      Adjust predicted Q-values with the target Q-values
      Train the online network with the adjusted Q-values
      Decrease epsilon to adjust the exploit-explore balance
      Each τ moves update the target network
  end

  Update current state to next state
  Add reward to total reward

end

return total reward

4.3.3 Training data

One major disadvantage of reinforcement learning, is the amount of data that is required to learn. Although the RL setup has been simplified as much as possible trading only one spread, with a limited amount of actions available and a limited amount of information available to process and with efforts made to increase sample efficiency with experience replay, to make sure there is enough data for the agent to learn, an additional technique is used to simulate spread data. A block bootstrap

function splits the available spread data in blocks of 20 consecutive trading days and creates 100 day trading periods from these different blocks at random. This technique allows to fairly simply generate different trading scenarios for the agent to learn from [Balesse and Lemahieu, 2024].

4.4 Benchmarking a Pairs Trading strategy

Previous two sections discuss an approach to solve the two main problems that occur in pairs trading. The first problem is how to find pairs that are optimal for a pairs trading strategy, and the second problem is once a pair has been identified how one can optimally trade the spread between these pairs. Chapter 2 mentions the cointegration strategy as a strategy for selection of optimal pairs. This is a straightforward strategy that is relatively easy to implement and has been used to benchmark other strategies [Krauss, 2017], hence we will implement a cointegration strategy as benchmark for optimal pair selection. Furthermore, Chapter 2 mentions the trading strategy applied in [Gatev et al., 2006], where the trader takes a position once the spread becomes larger than two times the standard deviation and exits the position when the spread converges to zero. This approach will be used to benchmark the trade execution.

4.4.1 Pair selection with cointegration

The cointegration strategy to benchmark the optimal pair selection for pairs trading is based on [Rad et al.,], which provides an off the shelf approach that is easily implemented in Python. The first step of the approach consists of calculating the sum of Euclidean Squared Distances (SSD) between the normalized prices for every spread, and a pre-selection is made based on this SSD value, where one only keeps a certain amount of spreads that are below a specific SSD threshold value. If the SSD becomes too large, there is already too much divergence between the asset prices in order to have a tradable spread. The second step of the approach relies on the principle of cointegration. Cointegration assumes that, although two time series can both be nonstationary, a linear combination of those two time series can be stationary, hence mean-reverting. Given this principle, the spread between two price process can be defined by Equation 4.7.

$$Spread_t = P_i, t - \beta P_j, t - \delta \tag{4.7}$$

Where P_i, t, P_j, t are the asset prices for asset i and j respectively at time t, β is defined as the regression coefficient after fitting a regression between P_i and P_j, and α is the intercept of the regression. In order to determine if this linear combination of asset-prices is mean-reverting, a stationarity test is required. There exist different stationarity tests, but the test used in our research is the Augmented Dickey Fuller (ADF) test. The ADF test is a unit root test, where the unit root is a feature of time series that indicates if there is any stochastic trend in the time series that drives it away from its mean value. The presence of the unit root feature in a time

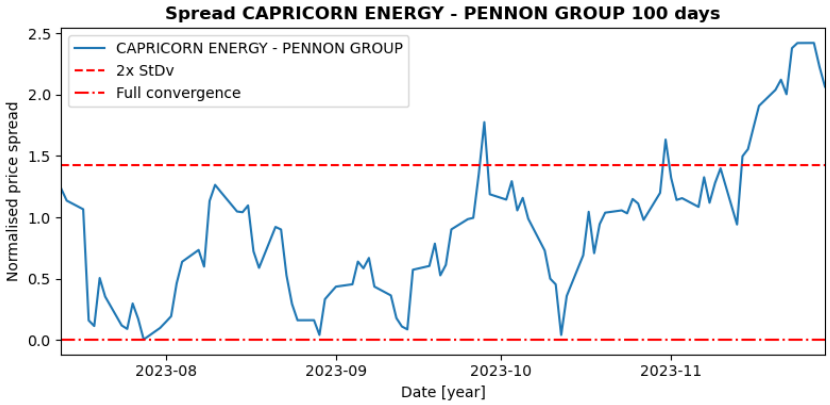

FIGURE 4.3: A representation of the borders used to open and close positions according to [Gatev et al., 2006]. The figure shows that there is only one complete trading opportunity in the 100 day trading period. The spread goes above the 2x standard deviation once to then convert to zero again.

series causes the nonstationarity [Ritu Santra,]. For each pair, the test is used in order to determine if the spread for the specific pair is mean-reverting or not. The test provides a t-statistic and a p-value, which provides an indicating of the mean-reverting character of the spread at hand. The original SSD ranking is kept, but the pairs with a less negative t-stat and/or a higher p-value are removed from the ranking, providing a top 20 ranking just as is the goal with the expected signatures approach.

4.4.2 Trade execution

Also, the original trade execution strategy from [Gatev et al., 2006] is straightforward and easy to implement in Python. The first step is to determine the standard deviation of the spread over the available history, which our research takes as the last year. The second step is for the agent to take position once the spread is larger than 2 times its standard deviation, and the last step is for the agent to exit the position once the spread fully converts back to zero. Figure 4.3 provides a visual representation of this trade execution approach on the Capricorn Energy vs Pennon Group spread. According to this strategy, the trader will take position half way the trading window and will exit the position again shortly after. The trader takes a position a second time, and this position is excited as the end of the trading period is reached at a loss.

Chapter 5

Results

The following chapter reports and discusses the results of pairs trading with expected signatures and reinforcement learning in line with the methodologies presented in Chapter 4 and using the data discussed in Chapter 3. First, the profit and loss regression is discussed. An analysis is made on how well signatures can indicate the potential PnL of a spread and which signatures are most important in this relation. Second, the section considers the Expected Signature regression model, where the Expected Signature of a spread is determined by predicting future spread values. Third, using the Expected Signature results and the PnL signature relation, expected PnLs for all eligible spreads are calculated and compared to the test results and the benchmark output of the cointegration method. Fourth, the optimal trade execution with the constructed reinforcement learner is applied and compared to the benchmark method and a random agent. In all sections the assumption of a trading period of 100 days was made which roughly translates to half a year and matches with the typical length for trading periods in pairs trading literature.

5.1 PnL regression

In this section the regression model between signatures and pairs trading profit and loss of a spread is analyzed. In Chapter 4 it is described how for a specific trading period a Ridge regression model is fitted on the signatures of the spreads and the linked PnLs over the respective trading period. In the following, the model is analyzed for different signature truncation orders in order to research the effect of the signature order and to try and understand the relation between the signature of a spread and its pairs trading performance. In the first part the model performance is discussed, and the second part zooms in on the importance of the different signatures.

5.1.1 Model performance

The performance of the model is evaluated for different signature orders. The orders evaluated are 3, 6, and 10. It is possible to test the model for higher orders, but given the exponential increase of variables with an increase in signature order, the

required computational efforts would increase substantially. For the evaluation of performance, the available set of labels is divided in a train and test set. With a given train and test split, the performance is evaluated by providing a visualisation of the predicted test values vs. the actual test values. If the model is performing one should expect a concentrated cloud around the diagonal dashed line. If the cloud is more dispersed away from the dashed line, the model is not performing well. In addition to the visual representation of the model performance, the median average error and the R^2 score are provided to as quantitative measure to compare the different regressions [skit-learn,]. Furthermore, in order for the model to use the optimal alpha, the code uses RidgeCV, a function provided by sklearn that uses cross-validation in order to select an optimal regression factor α from a provided set of α's. The available set of values for the regularization factor is $[1e(-2), 1e(-1), 1e, 1e1, 1e2, 1e3, 1e4]$.

Figure 5.1 shows the performance for the signature PnL regression with a signature order of 3,6, and 10. From both the visual representation, and the median average error and R^2 values it can be understood that the model improves for a higher order of signatures. This is not unexpected given that the additional features can provide additional information to improve the regression, while there is enough data provided to handle the high amount of features. The median average error should be as small as possible, while the R^2 value should be as close as possible to 1. With decreasing order the MAE increases, while the R^2 value decreases. Where for a signature order of 10 there is only a light dispersion of the actual vs. predicted values, the dispersion becomes substantially larger for signature order 3. While higher order signatures clearly outperform for the PnL regression model, the required computing effort can be a restriction when execution time matters.

5.1.2 Coefficient analysis

The previous analysis indicates that the regression model with a higher signature order provides better results compared to the regression model for lower signature orders. Given the improved model performance, it is expected that higher-order signature values are more important for the regression compared to the lower-order signature values. In order to asses the impact of single features, the coefficients of the model are analysed. In line with [skit-learn,], the size of the coefficients indicate the importance of the feature when the variance of the feature is also taken into account, or when the regression data has been pre-processed by a scaler. For our research specifically, the data has been pre-processed by a scaler, which normalizes the features and allows the comparison of the size of the coefficients in order to assess the importance of the different features. Figure 5.2 provides a list of the features with the largest coefficient and a list of the features with the lowest coefficient. The figure confirms the expectation that higher-order signatures carry more important features compared to the lower order signatures, with the higher coefficients mostly related to higher-order signatures and the smaller coefficients more, but not only related to the lower-order signatures. Due to the higher order, one cannot provide a geometrical interpretation of the features of higher importance. These features

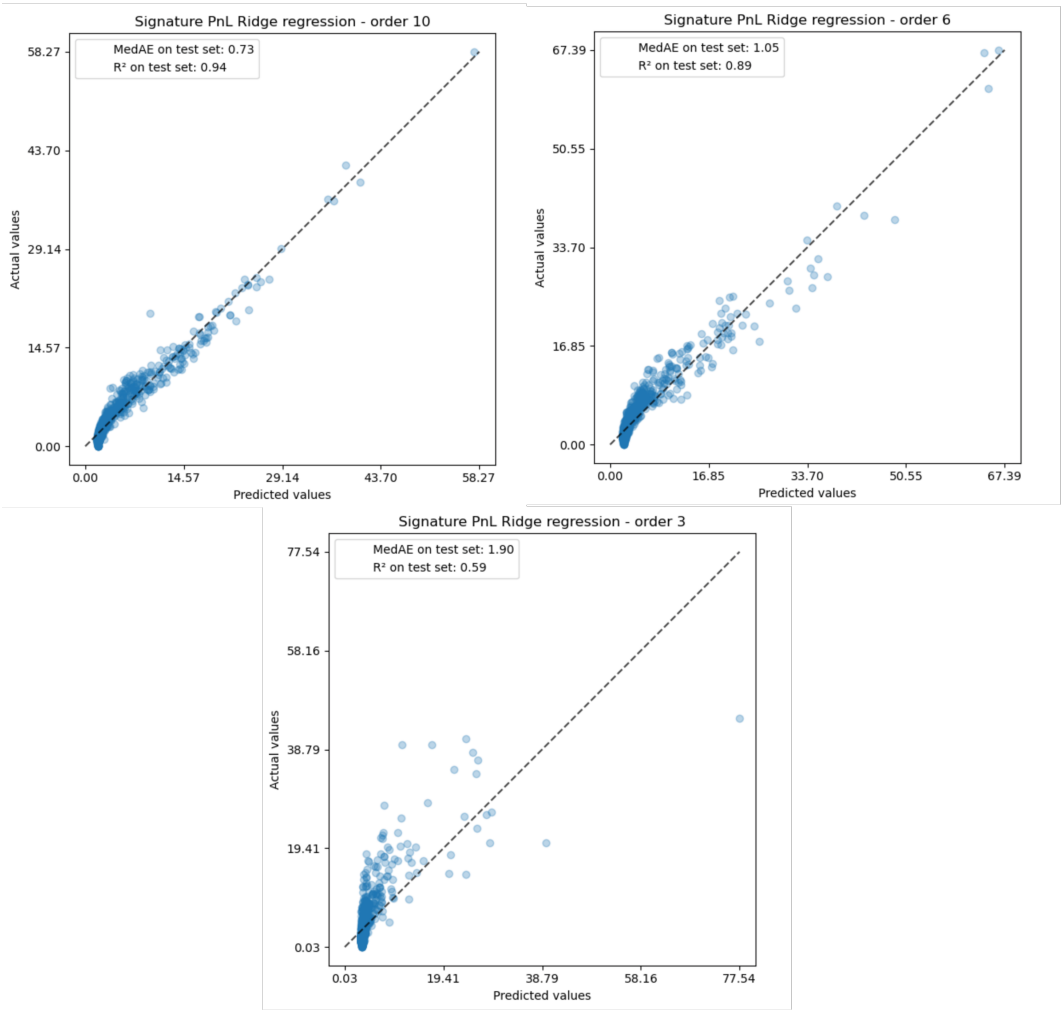

FIGURE 5.1: The figure shows the signature PnL regression analysis for signature order 10 (upper left), 6 (upper right), and 3(lower middle). The figure plots the actual values of the test set versus the predicted values of the test set and provides the median average error and the R^2 value o the test set for each regression. Both the visual representation and the quantitative measures show that the quality of the model declines for a declining signature order.

carry high dimensional information about the spread time series that is relevant for its tradability for a pairs trading strategy. From a pairs trading perspective this is unexpected given one would expect the drift, which translates to the first order signature, and the volatility, which translates to the second order signature of the spread to be the main contributors to the potential profit for trading that specific spread.

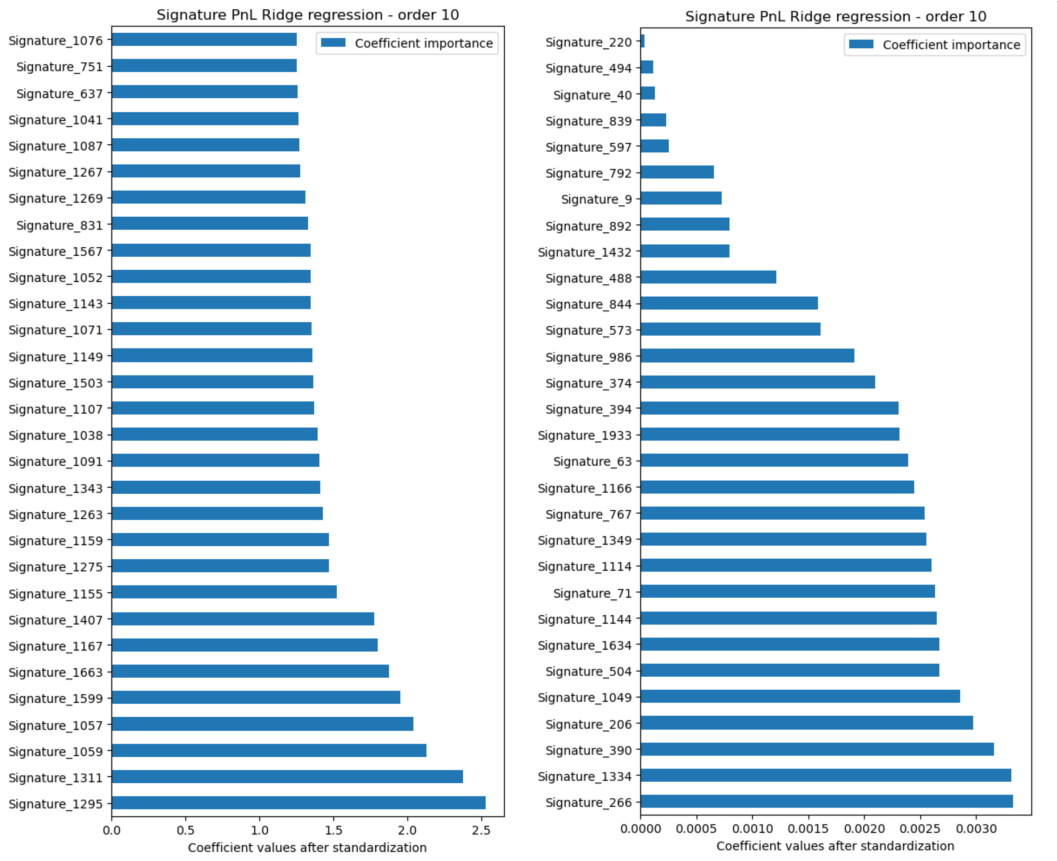

Figure 5.2: The figure provides an overview of the largest coefficient values (in absolute value) on the right and the smallest coefficient values (in absolute value) on the left. The most impactfull signatures fall under the highest signature order included in the analysis. For the least impactfull signatures, there is no clear tendense

5.2 Expected Signature regression

In the following section, the Ridge regression model to predict the next spread value based on the signature of historic spreads is discussed. An important difference with the previous section is that while for the modelling of the signatures to the PnL all data from all spreads was pooled to generalize the spread to PnL relation, the modelling of the signatures to the next spread value has to happen for each spread separately. The first reason to consider each spread separately, is that each spread has its own characteristics with the unique character of the interactions between the two assets that make the spread. These interactions and, as a consequence, the spread produced by these interactions, cannot be generalised. The second reason to not pool all the available data for different spreads, is that after testing this approach, the pooled model generalises too much under a high regularization constant, producing outcomes that are not usable for this thesis. In the first part of

this section the research considers the model performance and the influence of the different parameters like the window q, the step k, and the signature order n. In the second part of this section the important coefficients for the spread prediction are discussed.

5.2.1 Model performance

The data for this section considers the signature of a time series window of the spread and the subsequent spread value. Collecting the labels happens by stepping through the data with a specific step that is not necessarily equal to the window. The first label can be the signature of the spread from $t = 1$ until $t = 200$, where the window is 200, and the value of the spread at $t = 200 + 1$, while the next label will be signature of the spread from $t = 1 + step$ until $t = 200 + step$ and the value of the spread at $t = 200 + step + 1$. The choice of window and step substantially influences the amount of available labels, in addition to not pooling the regression data which decreases the available data points. Together with the signature order, the window and the step are the main parameters influencing the model performance. For different signature orders we analyse the influence of the window and the step on the average model performance. As measure for the model performance the R^2 value is used to determine the average out-of-sample performance. Tables 5.1, 5.2, 5.3 provide the average performance of the model for a max signature order of 2, 3, and 4 respectively. It is possible to use a higher signature order, but then the computational effort becomes a bottleneck and for many window/step combinations the regression would fail.

TABLE 5.1: Table provides the average R^2 value for all spreads running the expected signature regression model with a signature truncation at order 2. The positive scores are highlighted in blue. For signature of order 2, positive values are registered for a smaller step of 10 - 20 time points. The optimal value is found for a window of 400 spreads and a step of 10 time points.

window \step	**10**	**20**	**50**	**100**	**150**	**200**
10	1.91E-02	5.00E-02	-2.72E-01	-2.34E-01	-5.92E+01	-5.33E+01
20	6.24E-02	9.27E-02	-2.46E-01	-1.46E-01	-2.49E+01	-4.34E+01
50	1.23E-01	1.06E-01	-1.06E-01	-1.73E-01	-7.82E+02	-6.06E+01
100	1.78E-01	1.75E-01	1.66E-01	-1.80E+00	-3.23E+01	-9.56E+01
200	2.83E-01	2.68E-01	1.57E-01	-1.51E-01	-4.51E-01	-4.49E+01
400	3.72E-01	3.56E-01	-2.74E-01	1.32E-01	-2.11E+00	-1.75E+02

The results show that for many configurations, the average is well below zero. This is aided by the fact that the maximum R^2 value is 1, while there is no minimum value. A negative value for R^2 renders the regression pointless and can be substantially negative. This means that if the average R^2 value is positive, most of the values should be positive and there is a stable solution. The best scoring configuration can be found for the signature with truncation order 3, with a window of 200 and

Table 5.2: Table provides the average R^2 value for all spreads running the expected signature regression model with a signature truncation at order 3. The positive scores are highlighted in blue

window \step	10	20	50	100	150	200
10	-4.20E+01	-2.79E+07	-8.92E-01	-3.31E-01	-2.20E+00	-3.62E+08
20	-8.79E-01	1.50E-02	-4.10E+00	-1.33E+01	-2.35E+00	-3.20E+01
50	-4.07E+00	1.99E-01	-1.55E+00	-8.76E-01	-1.69E+02	-1.69E+09
100	-9.08E+00	-1.08E+01	-7.92E-02	-6.32E-01	-4.35E+00	-3.06E+08
200	-3.05E+02	3.86E-01	1.75E-01	1.46E-01	-2.69E+00	-7.22E+11
400	-7.73E+03	-3.99E+03	-5.12E+01	3.09E-01	-6.25E+00	-3.15E+06

Table 5.3: Table provides the average R^2 value for all spreads running the expected signature regression model with a signature truncation at order 4. No positive averages were registered, showing that the additional features decrease the overall model performance.

window \step	10	20	50	100	150	200
10	-3.62E+00	-6.24E+03	-1.52E+02	-6.92E+04	-2.67E+00	-2.03E+02
20	-2.62E+00	-3.01E+00	-2.66E+02	-3.85E+02	-2.22E+04	-7.20E+02
50	-3.06E+02	-8.57E+03	-2.59E+03	-1.47E+01	-1.02E+05	-2.65E+02
100	-3.57E+02	-3.82E+04	-5.47E-01	-9.04E+02	-2.99E+03	-9.48E+00
200	-5.82E+01	-8.05E-02	-2.55E+00	-1.32E+05	-6.08E+01	-2.04E+01
400	-1.29E+03	-3.24E+03	-8.84E+05	-2.07E+01	-1.24E+06	-9.08E+01

a step of 20, scoring an average of 0.39. This is below the 0.5 mark, which would be the result if the model would only predict the average of the test set, meaning the generated models do not perform as one would like. However, in general many of the individual performances need to be above 0.50 in order to compensate for the larger negative values that are still present. It is decided to take the most stable configuration with a window of 200 time steps, stepping through the historic spread with a step length of 20 time steps, for a maximum signature order of 3. A large window of 200 historic spreads is required in order to improve the prediction of the next spread, while the step is kept small to provide sufficient data to the learner.

The cause of the low model performance can be multiple. Firstly, when stepping through the historical data with a larger step, the amount of data is limited. If there are 8000 time points available, stepping through with a step of 200 only provides 40 labels, while stepping through with a step of 20 still only provides 400 labels. Even with a smaller step the amount of data is still very limited when comparing to the number of features. This is also one of the reasons why it is not possible to increase the signature order. Secondly, the opposite might also be through when stepping through the data with a small step, like 1, for a large window. Using a step of 1 time point on a large window would cause a high correlation between the different labels. Thirdly, a linear model might not be able to capture the relation between the next spread value and the signature of the historic spreads despite the signatures capturing

the non-linear characteristics of the spread. A potential solution to improve the results is to make use of synthetic data that is able to catch the characteristics of the spreads. E.g., [Balesse and Lemahieu, 2024] provides a framework for multivariate time series simulation, where the simulator can take into account specific desired characteristics.

5.2.2 Coefficient analysis

Just as for the PnL regression, this section considers the importance of the different features by looking at the absolute value of the coefficients after standardization of the features. Figure 5.3 provides an overview of the most important and least important features. Given the outcome of the regression model differs for each spread, a spread was used with a R^2 score of 0.95. The goal is to present the analysis for a spread for which the regression provides a strong solution and to indicate which features played an important role in this. Although a high model score is not a guarantee that the relation between the signature of the window of historic spread values and the concurrent spread value, it does provide an indication which features are most likely to provide a good prediction. Figure 5.3 shows that the signature components from signature order 3 play the largest role in the prediction of future spreads based on signatures, while the coefficients for the temporal constants is 0. This indicates that with more data using a higher order signature might deliver an improved performance of the model.

5.3 Expected PnL

After training the model for the signature PnL regression and for the prediction of the next spread value, the two models can be combined in order to determine the expected signature and estimate the expected PnL for each spread in line with the methodology described in Chapter 4. Using the expected signature regression, the future 100 spreads are predicted, and the signature of this predicted interval is calculated. This signature is then used in order to predict the expected PnL for the specific spread. Executing this process for each spread provides a set of expected PnLs. Ranking these according to the PnL value provides a ranking of spreads that go from most promising for pairs trading to least promising for pairs trading. However, as already discussed in the previous section, not for all spreads there exists a stable prediction of the future spread values, which can make the calculation of the expected signature either impossible or unstable. Because of this reason, our code filters out the spreads for which there is either no prediction or an unstable prediction which reflects in an outrageous PnL value. For the 4950 analysed spreads, 2532 spreads deliver an acceptable solution. With an acceptable solution, it is meant that the PnL exists and is in range of the historic PnL values. Historic PnL values vary between 1 and 70. Providing some additional space results in an acceptable range from 1 to 100. From this point onwards, only these 2532 spreads after filtering will be considered.

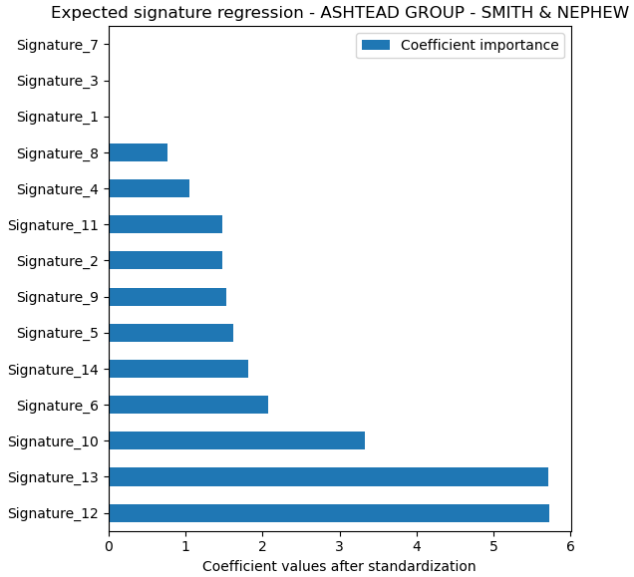

Figure 5.3: The figure shows the size of the coefficients of the respective features in the expected signature regression. Features 1, 3, and 7 correspond to a temporal constant, hence the coefficient is 0. Feature 2 corresponds to the drift. Features 4,5, and 6 play a role in the volatilty of the spread. However, it is the features from signature order 3 that play the more important role with the highest coefficients.

5.3.1 Pair ranking

At the start of the analysis, the last 100 days and their respective PnLs were removed from all data, to not have the values incorporated in training of the different models. For these 100 days, the mentioned approach is used to provide a ranking of expected PnLs, and the results are compared to the actual ranking and the ranking provided by the cointegration method to provide a benchmark. Table 5.4 provides the top-20 spreads based on expected PnL, the actual ranking in the test set, and the actual PnL for the test period. The table indicates that, although the parameters of the expected signature model and the PnL regression were optimised, the model is not capable of capturing the ranking of the test set. First, from the expected top 20 spreads, only one appears in the actual top 20 spreads of the test set. Secondly, the predicted PnL values are not comparable to the PnL values in the test set, and lastly, the internal ranking of the expected signature model does not respect the internal ranking of the spreads in the test set. Given the results of the previous sections, it can be assumed that the main causes can be found in the prediction of the future spread values. First, with a lack of data and hence a reduced amount of features, the prediction of the spread values becomes unstable. Second, predicting 100 consecutive values without adjusting for the correct previous value can cause the model to run away from the general spread values. Lastly, the low signature order might not be able to capture the driving forces for predicting the next spread value.

TABLE 5.4: The table shows the predicted best 20 pairs for pairs trading according to the expected signature model and compares the output with the actual ranking of the assets in the test set and with the actual PnL value from the test set. The expected signature model does not predict any actual top 20 spreads, except the spread that is highlighted in blue. Furthermore, the predicted values are much larger compared to the actual PnL values.

Predicted Rank	Spread	Predicted PnL Value	Actual Rank	Actual PnL Value
1	IMI - TATE & LYLE	98.05	673	6.92
2	HALMA - PERSIMMON	97.59	62	17.31
3	RIO TINTO - VANQUIS BANKING GROUP	97.42	284	9.04
4	ST.JAMES'S PLACE ORD - HOME RETAIL GROUP DEAD ...	97.19	767	5.98
5	SABMILLER DEAD - 06/10/16 - HAMMERSON	96.88	1897	2.08
6	BARRATT DEVELOPMENTS - HAMMERSON	95.78	1443	3.54
7	RIO TINTO - LAND SECURITIES GROUP	94.97	328	8.77
8	RIO TINTO - GKN DEAD - 21/05/18	93.28	281	9.06
9	SMITH & NEPHEW - TESCO	91.95	1432	3.55
10	ANTOFAGASTA - IMI	90.66	14	33.90
11	SMITH & NEPHEW - HSBC HOLDINGS	90.39	1424	3.57
12	IMPERIAL BRANDS - FIRST GROUP	90.20	544	7.37
13	FERGUSON - OLD MUTUAL LIMITED	90.06	1016	4.88
14	FERGUSON - CAPITA	90.05	949	5.12
15	HISCOX DI - CAPITA	90.03	557	7.28
16	BUNZL - POWERGEN DEAD - DEAD 01/07/02	88.49	887	5.31
17	ST.JAMES'S PLACE ORD - LEGAL & GENERAL	88.29	1192	4.20
18	RIO TINTO - TOMKINS DEAD - 28/09/10	87.82	279	9.06
19	DIAGEO - UBM DEAD - 18/06/18	87.42	803	5.75
20	HISCOX DI - CENTRICA	85.03	624	7.08

TABLE 5.5: The table shows the predicted best 20 pairs for pairs trading according to the expected signature model with a less strict filter for the PnL values (150 vs 100 max) and compares the output with the actual ranking of the assets in the test set and with the actual PnL value from the test set. The expected signature model does predict 3 actual top 20 spreads. However, the predicted PnLs are further away from the expected values.

Predicted Rank	Spread	Predicted PnL Value	Actual Rank	Actual PnL Value
1	ADVANCED MED.SLTN.GP. - HOME RETAIL GROUP DEAD...	149.98	468	7.82
2	RENISHAW - VANQUIS BANKING GROUP	149.20	611	7.17
3	PENNON GROUP - BALFOUR BEATTY	144.09	374	8.58
4	BERKELEY GROUP HOLDINGS (THE) - RIO TINTO	143.67	213	11.13
5	ANTOFAGASTA - ASTRAZENECA	141.17	3	35.69
6	NEXT - SSE	134.86	186	12.25
7	SCHRODERS - CAPITA	134.65	1543	3.21
8	WHITBREAD - MARKS & SPENCER GROUP	133.08	1112	4.63
9	UNILEVER (UK) - FIRST GROUP	129.66	1750	2.75
10	ST.JAMES'S PLACE ORD - POWERGEN DEAD - DEAD 01...	129.52	786	5.97
11	IMPERIAL BRANDS - HAMMERSON	128.71	568	7.31
12	AMLIN DEAD - 02/01/16 - GALLAHER GROUP DEAD - ...	127.98	1898	2.20
13	IMI - TAYLOR WIMPEY	127.26	757	6.41
14	HOWDEN JOINERY GP. - SEVERN TRENT	127.08	104	13.69
15	RIO TINTO - ENTERPRISE OIL DEAD - DEAD 25/06/02	123.75	279	9.17
16	DIAGEO - GKN DEAD - 21/05/18	123.72	823	5.78
17	ANTOFAGASTA - SEVERN TRENT	120.79	7	35.00
18	IMI - CRH PUBLIC LIMITED (LON)	115.34	776	6.10
19	WHITBREAD - CENTRICA	113.50	1069	4.73
20	ANTOFAGASTA - SSE	112.82	5	35.22

When widening the range of acceptable PnL values to 1-150, the model is capable of finding three original top 20 values as shown in Table 5.5.

Next to the performance of the expected signature model which is presented in previous tables, Table 5.6 presents the top 20 spreads from the test set and finds the rank of these specific spreads for the expected signature prediction and for the

cointegration strategy. In addition to the rank of the cointegration strategy, the table also reflects the t-statistic and the p-value that accompanies the cointegration analysis done for every spread. Where the methodology of [Rad et al.,] describes a pre-filtering based on the sum of Euclidean squared differences as explained in Chapter 4, for the purpose of this result and to be able to show all ranks, there is no pre-filtering based on SSD and it is only used for ranking. Several conclusions can be made from the table. The first conclusion is that when it comes to ranking according to potential PnL the expected signature model outperforms the classic cointegration model given the cointegration model ranks these top 20 spreads very low in its overall ranking. This brings us to the second conclusion: the top 20 is very biased towards spreads that contain specific assets like Antofagasta and Ashtead Group. Due to how the spreads are constructed based on normalizing prices for the complete data period, companies that have grown substantially over time will still have a large normalized price compared to companies that have stayed relatively stable over time. Figure 5.4 shows the normalized stock price of Antofagasta versus Reckitt Benckiser Group over the test period. The normalized price of Reckitt Benckiser Group stays flat, while the normalized price of Antofagasta varies substantially compared to common spread values. This creates pairs with a varying spread that presents trading opportunities due to a higher volatility and higher absolute spread values, but where only one of the assets actually deviates in price. The flat price asset then becomes a hedge of which the direction depends on the movement of the other stock, while the trading predictions and decisions are based on the movements of the varying asset. Figure 5.4 shows why these assets rank very low according to the cointegration method. Due to the substantial distance between the normalized prices the SSD is large and given the SSD determines the ranking for the cointegration approach they score poorly. This also indicates a previously mentioned flaw in the approach of [Rad et al.,], given some of these spreads score well on the stationarity test and show potential for trading, but due to the SSD ranking they would rank too low to be picked up. Overall, the expected signature approach provides an improved ranking compared to the classic method benchmark, however, the new method currently does not manage to predict the pairs trading PnL behaviour.

5.4 Trade execution

In the final results section, the results of the optimal execution strategy are discussed. Assuming the selection strategy is capable of selecting the optimal pairs for pairs trading over the next trading period, the optimal execution algorithm aims to maximise the potential profit of trading the selected spreads. Chapter 4 provides a description of the trading environment and the Deep Q-learning agent which are used in order to maximise the trading profit. To assess the performance of the trading agent and given the debatable results of the optimal pair selection, the DQL-agent is trained and tested on a pre-selected spread. The chosen spread is the spread between Capricorn Energy and Pennon Group. This spread is chosen because of two reasons. The first reason is that the normalized asset prices act more in line with an expected

TABLE 5.6: This table shows the top 20 spreads from the test set with their actual PnL value. Furthermore is shows where the assets rank for the expected signature (ES) model, and the cointegration model. The t-statistic (t-stat) and the p-value (p-val) are added as cointegration test result.

Actual Rank	Actual PnL	Spread	ES Model Rank	Cointegration Model Rank	t-stat	p-val
1	35.99	ASHTEAD GROUP - RECKITT BENCKISER GROUP	1146	2509	-1.941	0.313
2	35.90	ANTOFAGASTA - RECKITT BENCKISER GROUP	615	2416	-1.909	0.328
3	35.51	ANTOFAGASTA - RENISHAW	113	2478	-2.212	0.202
4	35.16	ANTOFAGASTA - DIAGEO	361	2459	-1.281	0.638
5	34.77	ANTOFAGASTA - ADVANCED MED.SLTN.GP.	392	2502	-2.330	0.163
6	34.65	ANTOFAGASTA - AMLIN DEAD - 02/01/16	544	2427	-1.974	0.298
7	34.58	ASHTEAD GROUP - ASTRAZENECA	561	2522	-2.477	0.121
8	34.44	ANTOFAGASTA - INTERMEDIATE CAPITAL GP.	432	2422	-1.466	0.550
9	34.29	ANTOFAGASTA - IMPERIAL BRANDS	28	2439	-2.642	0.085
10	34.24	ANTOFAGASTA - SCOTTISH MORTGAGE	497	2500	-3.734	0.004
11	34.16	ANTOFAGASTA - HISCOX DI	131	2466	-3.263	0.017
12	33.93	ASHTEAD GROUP - BG GROUP DEAD - 15/02/16	723	2530	-2.481	0.120
13	33.91	ASHTEAD GROUP - SEVERN TRENT	329	2515	-2.242	0.191
14	33.91	ANTOFAGASTA - NEXT	553	2381	-0.897	0.789
15	33.90	ANTOFAGASTA - IMI	9	2465	-2.092	0.248
16	33.90	ASHTEAD GROUP - REXAM DEAD - 01/07/16	22	2528	-2.481	0.120
17	33.88	ASHTEAD GROUP - GALLAHER GROUP DEAD - 18/04/07	415	2527	-2.481	0.120
18	33.80	ASHTEAD GROUP - SSE	322	2517	-2.340	0.159
19	33.80	ASHTEAD GROUP - RELX	548	2523	0.315	0.978
20	33.77	ASHTEAD GROUP - VANQUIS BANKING GROUP	237	2531	-1.419	0.573

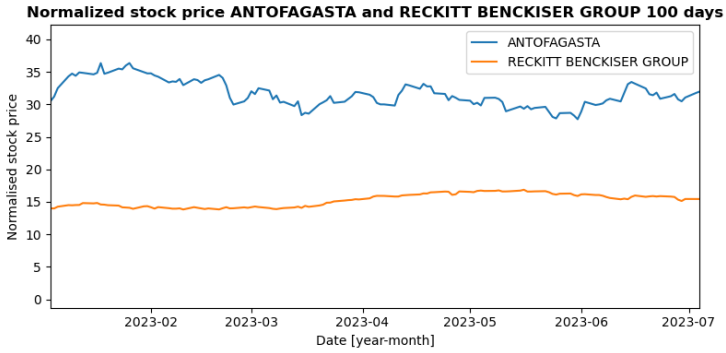

FIGURE 5.4: The figure shows the 100 days trading period normalized stock prices for Antofagasta and Reckitt Benckiser Group. The normalized price of Reckitt Benckiser Group stays flat, while the normalized price of Antofagasta changes substantially.

pair for pairs trading, given they are mean reverting but also present multiple trading opportunities over a 100 day trading period. The second reason is that for these two assets, there is actually a fundamental reason for them to be mean-reverting, given both assets are energy market related companies. In the following, first the training over multiple episodes is presented and discussed, and second the performance over the test set is presented and compared to the benchmark trading methodology.

5.4.1 Training

As described in the methodology, the deep Q-learning agent is trained over multiple synthetically produced episodes according to Algorithm 6. For the training of the

agent to trade the spread for Capricorn Energy and Pennon Group, 150 episodes of 100 trading days are generated. For each episode an additional 20 days of spread values are added to the data of every episode, with 20 days being the required lag values that form the state together with the current spread value and the current position. The reward of the agent is calculated according to Algorithm 5. For each episode, the cumulative reward is determined and saved. If the agent learns it is expected that over time the cumulative reward per episode should improve on average. Figure 5.5 shows the evolution of the average cumulative reward over the 150 training episodes of the deep Q-learning agent and the random agent. While the average cumulative reward of the random agent is close to zero, the average cumulative reward of the learning agent increases over time. In addition, the episodic rewards of the DQL agent are shown as well to show the variability in performance of the agent. From this figure, one can conclude that the agent manages to learn how to maximise the total reward over time for trading a spread. Furthermore, the smart agent, on average, outperforms an agent that is trading randomly indicating the agent is making thoughtful decisions when trading the spread for Capricorn Energy and Pennon group. Figure 5.6 provides the training evolution of the Deep Q-learning agent over the same episodes, but with a transaction penalty included in the reward function. From the figure it can be seen that it takes the agent longer to learn at the start, although this is by design. Under the same ϵ-greedy policy for the reward function with transaction costs the agent would revert to not trading at all. Hence, the agent required more time to explore and figure out the dynamics of the system. In order to facilitate for this, the decay was slowed down. Learning with transaction costs seems to have cause some form of regularization, with less large negative drops below the random agent performance.

5.4.2 Performance

In order to assess the performance of the trained agent over the test period, the algorithm is slightly adjusted. Figure 5.7 shows the normalized prices and for the Capricorn Energy - Pennon Group pair, while Figure 5.8 shows the spread and the borders for two times the standard deviation and full convergence. Table 5.7 provides the performance of the different methods based on the reward function. It is important to note that the reward function does not translate immediately to returns, as it punishes the trader for holding a position where the spread goes up, while this is not reflected in a buy and hold to conversion strategy. When assessing the trading strategy of the different actors, the DQL agent manages to take position at spread value 2.5, which is the maximum, but sells very fast. This is due to the additional reward provided for exiting a position, otherwise, the agent would learn to benefit from holding longer. The benchmark method would, however, hold the position for a lot longer. Analysing the eventual return of both strategies, the return is similar, however, the benchmark method would hold the position for longer which brings more exposure to risk.

FIGURE 5.5: The figure shows the learning process of the deep q-learning agent over 150 episodes plotting the cumulative reward from every episode and the average cumulative reward over the episodes. As a benchmark, a random agent is used for the same 15o episodes. The figure shows that the smart agent outperforms the random agent over time.

TABLE 5.7: Table shows the total episodic reward for trading on the test set according to the reward function of Trading Environment. The rewards are shown for the Deep Q-learning agent after learning 150 episodes, for the random agent which was run 10000 times to get an average performance on the test set, and for the standard deviation method (St. Dev. Method). The DQL Agent clearly outperforms the other methods when comparing total rewards. The best total reward is indicated in blue

Method	Test Result (Episodic Reward)
Deep Q-learning Agent	0.53
Random Agent (10'000 runs)	-0.34
St. Dev. Method	-0.55

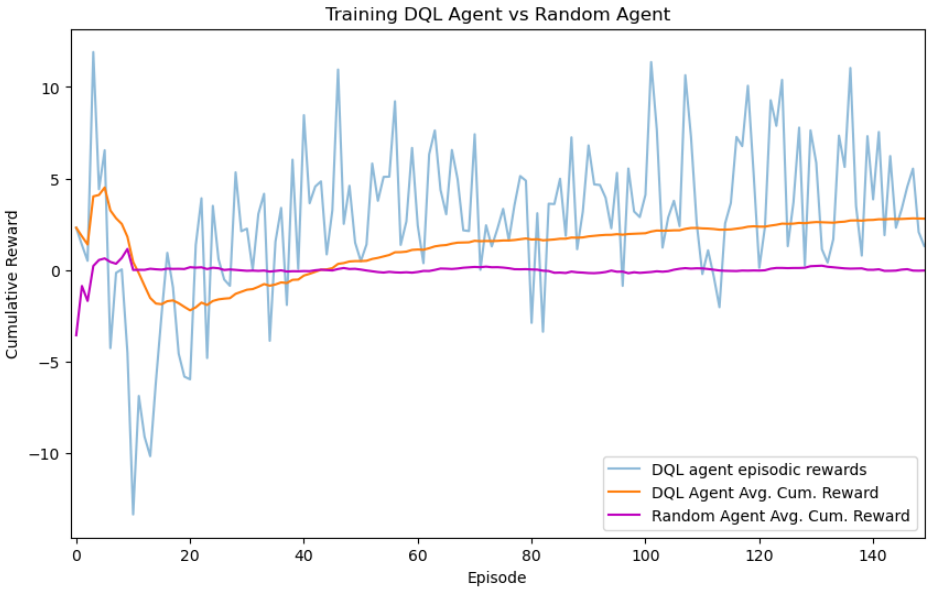

Figure 5.6: The figure shows the learning process of the deep q-learning agent over 150 episodes plotting the cumulative reward from every episode and the average cumulative reward over the episodes. In this case the reward function includes a transaction penalty. As a benchmark, a random agent is used for the same 15o episodes. The figure shows that the smart agent outperforms the random agent over time.

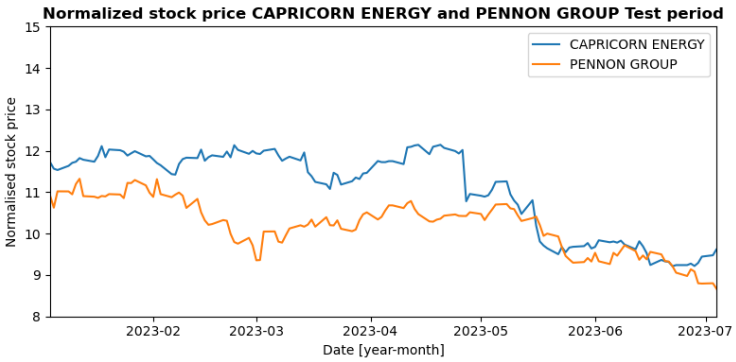

Figure 5.7: The figure shows the normalised prices of Capricorn Energy vs. Pennon Group over the test period

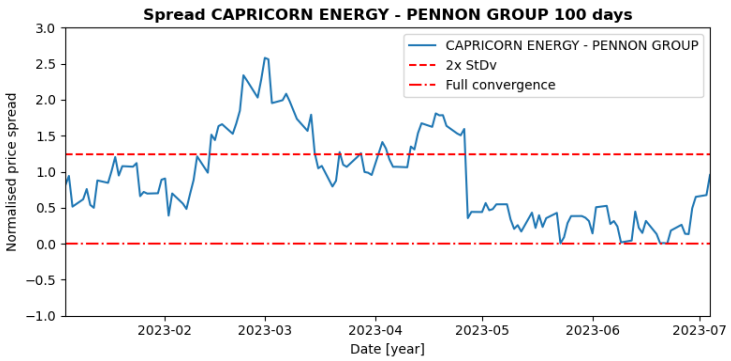

FIGURE 5.8: The figure shows the spread of Capricorn Energy vs. Pennon Group over the test period and the borders indicating the level of two times the standard deviation and full convergence. This provides an indication of the classic execution method

Chapter 6

Conclusion

The goal of this thesis was to research a new methodology for pairs trading that involves the selection of optimal pairs for pairs trading based on the expected signature of the spread and the trading of optimal pairs using a reinforcement learning agent. In order to rank the optimal pairs, the signature of the spread over the future trading period is predicted based on the relation between the signature of a window of historic spreads and the next spread value, with the expected PnL over that trading period calculated based on the relation between the signature and the pairs trading PnL of the spreads. For the optimal trading of the spread, a Deep Q-learner is used, providing the required infrastructure to learn an agent in a continuous setting. The results show that there is a strong relationship between the signature of the spread truncated at order 10 and the trading PnL. However, the relation between the signature of historic spreads and the next spread value proves difficult to ascertain. The main cause for a limited performance of the model is the lack of qualitative data labels. The method was not able to rank the best pairs for pairs trading, but outperformed the benchmark method based on cointegration. Looking at the optimal execution algorithm, the deep Q-learning agent shows to outperform the agent that chooses random actions and the benchmark trading method based on two times the standard deviation and full conversion when comparing the cumulative rewards. Comparing the cumulative rewards does not only reward buying high and selling low, but also punished holding an increasing spread. To conclude, when assessing the pair selection and optimal execution separately, the strategy shows the potential of outperforming the classical methods that are used as a benchmark.

The thesis shows strong potential for further research on the application of signatures in pairs trading and using a deep reinforcement learner for trading the spreads. Given the strong relation between the signatures of the spreads and the pairs trading PnL, it shows that signatures are a promising tool to identify high potential pairs for trading. There are two potential directions that can be indicated regarding the improvement of the selection model. The first option is a revision of the data and the expected signature model. With more data points and/or a different, more complicated, model like an LSTM, there is a potential to further improve the expected PnL prediction. A second option is only relying on historical data like the classical methods of distance

and cointegration and developing a signature kernel that can be used for ranking pairs. Regarding the reinforcement learner, here it is clear that the agent is able to learn, and the next step is to directly link the cumulative rewards of the agent with actual trading returns.

In conclusion, this thesis provides an insightful look on the predictive performance of the expected signatures model for ranking pairs for pairs trading and shows the potential of a deep reinforcement learning agent for profitably trading a spread. Furthermore, important insights in the relation between the signature of a spread and its PnL for pairs trading provides potential for further research.

Appendices

Appendix A

Python Code

The following Appendix considers the code that was used in order to implement the described methodology and to produce the reflected results.

A.1 Data

```
class DataLayer:
    def __init__(self, source: str, meta_source: str, nr_assets=None) ->
    None:
        self.source = source
        self.meta_source = meta_source
        self.nr_assets = nr_assets
        self.name_dictionary = None
        self.data = None
        self.return_data = None
        self.normalized_prices = None
        self.spreads = None
        self.spread_returns = None
        self.get_clean_data()
        self.get_asset_names()
        self.get_return_data()
        self.get_normalized_prices()
        self.get_spreads()
        self.get_spread_returns()

    def get_clean_data(self)->None:
        """Removes missing datapoints makes sure that all time series are
     of equal length and datetime index is set."""
        self.data = pd.read_csv(self.source)
        self.data = self.data.dropna(axis=1)
        self.data = self.data.rename(columns={'Unnamed: 0':'Time_stamp'})
        self.data.set_index('Time_stamp',inplace=True)
        if self.nr_assets:
            self.data = self.data[self.data.columns[:self.nr_assets]]

        # additional data cleaning
        # Set date time format
```

```
30          self.data.index = pd.to_datetime(self.data.index, format='ISO8601
    ')
31          # Filter out weekend days
32          self.data = self.data[self.data.index.dayofweek < 5]
33
34      def get_return_data(self)->None:
35          """Calculates the daily returns and removes the first line given
    it's an na."""
36          self.return_data = self.data.pct_change()
37          self.return_data = self.return_data.dropna()
38
39      def get_normalized_prices(self)->None:
40          """Calculates normalized prices in order to calculate the spreads
    ."""
41          # Get normalised prices by dividing through the first value
42          self.normalized_prices = self.data / self.data.iloc[0]
43          # Remove stocks that are delisted
44          stocks = self.data.columns
45          delisted_stocks = []
46          for stock in stocks:
47              if "DELIST" in stock:
48                  delisted_stocks.append(stock)
49          self.normalized_prices.drop(delisted_stocks, axis=1, inplace=True
    )
50          # Only keep stocks that do not die out in penny stocks and remove
     outliers
51          self.normalized_prices = self.normalized_prices[self.
    normalized_prices.sum().sort_values(ascending=False).index]
52          if len(self.normalized_prices.columns) > 102:
53              self.normalized_prices = self.normalized_prices[self.
    normalized_prices.columns[2:103]]
54          # Only keep the data points for which the spread becomes
    establihed
55          self.normalized_prices.iloc[500:]
56
57      def get_spreads(self)->None:
58          """Calculates the spreads betweeen the normalized prices"""
59          asset_pairs = list(itertools.combinations(self.normalized_prices.
    columns,2))
60          dict_spreads = {}
61
62          # Spread is defined as the absolute value of the difference
    between normalized prices
63          for pair in asset_pairs:
64              spread = abs(self.normalized_prices[pair[0]] - self.
    normalized_prices[pair[1]])
65              dict_spreads[f'{pair[0]} - {pair[1]}'] = spread
66
67          # Set spreads dataframe
68          self.spreads = pd.DataFrame(dict_spreads)
69          self.spreads = self.spreads.iloc[1:]
70
71      def get_spread_returns(self)->None:
72          """Calculates the daily returns on the spread."""
73          self.spread_returns = self.spreads.pct_change()
```

```
74          self.spread_returns = self.spread_returns.dropna()
75          self.spread_returns = self.spread_returns.iloc[1:-1]
76
77      def get_asset_names(self)->None:
78          """Replaces the ISIN with the asset full name."""
79          self.meta_data = pd.read_csv(self.meta_source)
80          self.name_dictionary = dict(zip(self.meta_data.iloc[:,0], self.
    meta_data['NAME']))
81          self.data = self.data.rename(columns=self.name_dictionary)
82
83      def visualise(self, nr_of_timeseries=10, data=None, title=None)->None
    :
84          """Allows to visualise the different data types."""
85          columns = data.columns
86          if nr_of_timeseries < len(columns):
87              columns = columns[:nr_of_timeseries]
88
89          plt.figure(figsize=(14,7))
90          for column in columns:
91              plt.plot(data.index, data[column], label=column)
92          plt.title(title)
93          plt.xlabel('Time')
94          plt.ylabel('Value')
95          plt.legend()
96          plt.show()
```

A.2 Pair selection

```
1  class PairSelection:
2      """Determines the pairs that are optimal for pairs trading."""
3
4      def __init__(self, data_source: str, data_meta_source: str,
       trading_window=6, expected_signature_order=3,
       expected_signature_lag_value=200, pnl_signature_order=10, alpha=1,
       step=20, nr_assets=None) -> None:
5          self.data_layer = DataLayer(source=data_source, meta_source=
       data_meta_source, nr_assets=nr_assets)
6          self.df_data = self.data_layer.data
7          self.df_returns_data = self.data_layer.return_data
8          self.df_returns_data_cum = self.data_layer.normalized_prices
9          self.df_spreads = self.data_layer.spreads
10         self.df_spreads_returns = self.data_layer.spread_returns
11         self.trading_window = trading_window
12         self.expected_sig_order = expected_signature_order
13         self.expected_signature_lag_value = expected_signature_lag_value
14         self.pnl_sig_order = pnl_signature_order
15         self.pnl_regression_data = None
16         self.expected_regression_data = None
17         self.spreads = None
18         self.start_day = None
19         self.end_day = None
20         self.pnl_model = None
```

```
21          self.optimal_pairs = None
22          self.alpha = alpha
23          self.step = step
24
25      def split_datetime_range_by_months(self):
26          """Creates a split of the available trading days for training and
         testing purposes."""
27          # Initialize a list to store the split points
28          split_points = []
29
30          current_date = self.start_day
31
32          while current_date < self.end_day:
33              next_split_date = current_date + relativedelta(months=self.
    trading_window)
34
35              if next_split_date > self.end_day:
36                  next_split_date = self.end_day
37
38              split_points.append(pd.to_datetime([current_date,
    next_split_date]))
39              current_date = next_split_date
40
41          return split_points
42
43      def calculate_PnL(self, df_spreads_pnl, col):
44          """Calculates a specific Pnl for a given column of a spreads
    dataframe, which means calculating the
45          PnL of a certain asset pair over a certain period."""
46          # find the indices of all local maxima and minima in the time
    series
47          indices_max = argrelextrema(df_spreads_pnl[col].values, np.
    greater)[0]
48          indices_min = argrelextrema(df_spreads_pnl[col].values, np.less)
    [0]
49          # Combine indices of maxima and minima
50          extrema_indices = np.concatenate((indices_max, indices_min))
51          # Sort the indices to preserve the order
52          extrema_indices = np.sort(extrema_indices)
53          # Calculate absolute differences between consecutive extrema
54          differences = [-(df_spreads_pnl[col].iloc[extrema_indices[i+1]] -
         df_spreads_pnl[col].iloc[extrema_indices[i]])
55                         for i in range(len(extrema_indices) - 1)]
56          # Given we only trade from the max in the ideal case, we only
    count the positive differences
57          positive_differences = [diff for diff in differences if diff>0]
58          return sum(positive_differences)
59
60      def calculate_signature_es(self, time_series):
61          """Calculate the signatures of the 2d path of the spread. Should
    be given as a list with dimenions nxd."""
62          time_points = [x/len(time_series) for x in range(0, len(
    time_series))]
63          # time augmentation of the path
64          path = np.array(list(zip(time_points, time_series)))
```

```
65          return iisignature.sig(path, self.expected_sig_order)
66
67      def calculate_signature_pnl(self, time_series):
68          """Calculate the signatures of the 2d path of the spread. Should
    be given as a list with dimenions nxd."""
69          time_points = [x/len(time_series) for x in range(0, len(
    time_series))]
70          # time augmentation of the path
71          path = np.array(list(zip(time_points, time_series)))
72          return iisignature.sig(path, self.pnl_sig_order)
73
74      def get_pnl_regression_data(self):
75          """Generates the Dataframe that can be used for regressing the
    PnL of a trading period on the signature of the time series of that
    trading period."""
76          self.spreads = self.df_spreads.columns
77          self.start_day = self.df_spreads.index[0]
78          self.end_day = self.df_spreads.index[-1]
79          date_splits = self.split_datetime_range_by_months()
80          regression_data = {spread: [] for spread in self.spreads}
81
82          for date_split in date_splits:
83              df_spreads_trading_period = self.df_spreads.loc[date_split
    [0]:date_split[1], :]
84              df_spreads_trading_period.reset_index(drop=True, inplace=True
    )
85              for spread in self.spreads:
86                  values = [f'{date_split[0]} - {date_split[1]}', 0, 0]
87                  values[1] = self.calculate_PnL(df_spreads_trading_period,
     spread)
88                  values[2] = self.calculate_signature_pnl(
    df_spreads_trading_period[spread].values)
89                  regression_data[spread].append(values)
90
91          return regression_data, date_splits
92
93      def get_pnl_regression_dataframe(self):
94          """Transforms the dictionary with PnLs and corresponding
    signatures for each spread in a dataframe that can be used for linear
     regression."""
95          signature_pnl_data, date_splits = self.get_pnl_regression_data()
96          flattened_data = []
97
98          # prepare regression dataframe from pnl_regression_data list
99          for spread in self.spreads:
100             for entry in signature_pnl_data[spread]:
101                 dates = entry[0]
102                 value = entry[1]
103                 array_values = entry[2].tolist()
104                 flattened_data.append([dates, value] + array_values + [
    spread])
105         columns = ['Date', 'Value'] + [f'Signature_{i+1}' for i in range
    (2**(self.pnl_sig_order+1)-2)] + ['Spread']
106         self.pnl_regression_data = pd.DataFrame(flattened_data, columns=
    columns)
```

```
107
108         # Cut off test data to avoid any data snooping
109         self.test_dates = f'{str(date_splits[-2][0])} - {date_splits
    [-2][1]}'
110         self.test_data = self.pnl_regression_data.loc[self.
    pnl_regression_data['Date'] == self.test_dates,:]
111         self.pnl_regression_data = self.pnl_regression_data.loc[~(self.
    pnl_regression_data['Date'].isin([self.test_dates])),:]
112
113         # Sort PnL data by
114         self.pnl_regression_data.sort_values(by='Value', ascending=False,
     inplace=True)
115
116     def get_pnl_regression(self):
117         """Regresses the sigantures on the PnL values for all available
    data and provides the coefficients to calculate the expected PnL."""
118         # Set parameters
119         y_pnl = self.pnl_regression_data['Value']
120         X_pnl = self.pnl_regression_data.drop(['Date', 'Value', 'Spread'
    ], axis=1)
121         self.X_pnl = X_pnl
122
123         self.pnl_scalar = RobustScaler()
124         X_pnl_scaled = self.pnl_scalar.fit_transform(X_pnl)
125         alphas = [1e-2, 1e-1, 1, 1e1, 1e2, 1e3, 1e4, 1e5]
126
127         # Set model and regress
128         self.pnl_model = RidgeCV(alphas=alphas)
129         self.pnl_model.fit(X_pnl_scaled, y_pnl.values)
130
131     def get_expected_regression_dataframes(self, spread):
132         """Determines for each spread the dataframe that is used for
    regression analysis and production of the expected signature."""
133         data = []
134
135         for k in range(0, len(self.df_spreads)-(self.
    expected_signature_lag_value+3), self.step):
136             data.append(list(self.calculate_signature_es(
137                 self.df_spreads[self.spreads[i]].values.tolist()[k:k+(
    self.expected_signature_lag_value+1)]))
138                 + [self.df_spreads[self.spreads[i]].values.tolist()[k+(
    self.expected_signature_lag_value+1)]]  + [spread])
139
140         return data
141
142     def get_expected_regression_dataframes_parallel(self):
143         # Use ProcessPoolExecutor to parallelize the task
144         columns = [f'Signature_{i+1}' for i in range(2**(self.
    expected_sig_order+1)-2)] + ['Value'] + ['Spread']
145
146         with ProcessPoolExecutor() as executor:
147             # Schedule the execution of the function for each spread
148             futures = [executor.submit(self.
    get_expected_regression_dataframes, spread) for spread in self.
    spreads]
```

```
149
150             # Collect the results as they are completed
151             list_results = []
152             for future in futures:
153                 list_result = future.result()  # Blocks until the future
    is done
154                 list_results += list_result
155
156         self.expected_regression_data_all = pd.DataFrame(list_results,
    columns=columns)
157
158     def get_expected_signature(self, spread):
159         """Calculates the expected signature for the next trading window
    for a given spread."""
160         # Set data
161         df = self.expected_regression_data_all.loc[:, self.
    expected_regression_data_all['Spread']==spread]
162         result = [spread, 0, True, True]
163
164         # Set input and output
165         y = df['Value']
166         X = df.drop(['Value', 'Spread'], axis=1)
167         # Scale features
168         scaler = RobustScaler()
169         X_scaled = scaler.fit_transform(X)
170         # Fit model
171         alphas = [1e-1, 1, 1e1, 1e2, 1e3, 1e4]
172         model = RidgeCV(alphas=alphas)
173         model.fit(X_scaled, y.values)
174
175         # Predict the next spreads
176         trading_days = int(self.trading_window/12*200)
177         new_spreads = []
178
179         last_values = self.df_spreads[spread].values.tolist()[-self.
    expected_signature_lag_value:]
180
181         try:
182             for _i in range(1,trading_days+1):
183                 last_spread_sig = scaler.transform(self.
    calculate_signature_es(last_values).reshape(1,len(X.columns)))
184                 new_spread = model.predict(last_spread_sig)
185                 self.new_spreads.append(new_spread[0])
186                 last_values = last_values[1:] + list(new_spread)
187
188             # Calculate the expected signature and rescale with PnL
    scalar
189             final_signature = self.calculate_signature_pnl(new_spreads)
190             rescale_final_signature = self.pnl_scalar.transform(
    final_signature.reshape(1,len(self.X_pnl.columns)))
191
192             # predict PnL from the PnL regression
193             try:
194                 pnl = self.pnl_model.predict(rescale_final_signature)
195                 result[1] = pnl[0]
```

```
196             except:
197                 result[3] = False
198                 print('pnl_failure')
199         except:
200             result[2] = False
201 
202         return result
203 
204     def get_optimal_pairs(self):
205         """Calculates the expected pnl for every pair and ranks the
    different pairs."""
206         # Get PnL data and Regression
207         self.get_pnl_regression_dataframe()
208         self.get_pnl_regression()
209         # Get expected signature data
210         self.get_expected_regression_dataframes_parallel()
211         # Initialize results
212         self.results = []
213         for spread in self.spreads:
214             result = self.get_expected_signature(spread)
215             self.results.append(result)
216 
217         # Filter results but keep the old resutls
218         self.unfiltered_results = self.results
219         columns = ['Spread', 'PnL Value', 'ES Succes', 'PnL Succes']
220         results_dataframe = pd.DataFrame(self.results, columns=columns)
221         useful_results = results_dataframe.loc[(results_dataframe['ES
    Succes']==True) & (results_dataframe['PnL Succes']==True) & (abs(
    results_dataframe['PnL Value'])<150),:]
222         useful_results.sort_values(by=['PnL Value'], ascending=False,
    inplace=True)
223         self.filtered_results = useful_results
224 
225         return useful_results
```

A.3 Optimal trade execution

A.3.1 Trading Environment

```
1 class TradingEnv(gym.Env):
2     def __init__(self, spread_data, lag_values=20, transaction_included=
    False, transaction_cost=0.002):
3         """Initiates the trading environment."""
4         super(TradingEnv, self).__init__()
5 
6         self.transaction_included = transaction_included
7         self.transaction_cost = transaction_cost
8 
9         self.lag_values = lag_values
10        self.spread_data = spread_data
11        self.action_space = spaces.Discrete(3)
```

```
12          self.observation_space = spaces.Box(low=np.array([np.min(
    spread_data), -1]), high=np.array([np.max(spread_data), 1]), dtype=np
    .float32)
13
14          self.actions = []  # To record actions
15          self.profits = []  # To record cumulative rewards
16          self.capital_positions = [] # To record invested capital
17
18          self.reset()
19
20      def reset(self):
21          """Resets the trading environment after every episode."""
22          self.position = 0
23          self.current_step = self.lag_values
24          self.total_profit = 0
25          self.current_capital = 1
26          self.actions.clear()
27          self.profits.clear()
28          return self._next_observation()
29
30      def _next_observation(self):
31          """Provides the next state."""
32          return np.array(self.spread_data[self.current_step-self.
    lag_values:self.current_step+1].tolist() + [self.position])
33
34      def step(self, action):
35          """Steps to the next state given a specific action by the agent.
    """
36          previous_spread = self.spread_data[self.current_step]
37          self.current_step += 1
38          current_spread = self.spread_data[self.current_step]
39
40          if self.transaction_included==True:
41              reward = self._calculate_reward_transaction_cost(action,
    previous_spread, current_spread)
42          else:
43              reward = self._calculate_reward(action, previous_spread,
    current_spread)
44
45          self.total_profit += reward
46
47          self.actions.append(action)
48          self.profits.append(self.total_profit)
49          self.capital_positions.append(self.current_capital)
50
51          done = self.current_step >= len(self.spread_data) - 1
52
53          obs = self._next_observation()
54
55          return obs, reward, done, {}
56
57      def _calculate_reward(self, action, previous_spread, current_spread):
58          """Rewards the agent for holding a position when the spread
    converges."""
59          reward = 0
```

```
60          if self.position == 1:
61              reward = previous_spread - current_spread
62              self.current_capital *= 1 + (previous_spread - current_spread
    )/previous_spread
63          if action == 0 and self.position == 0:
64              self.position = 1
65          elif action == 2 and self.position == 1:
66              self.position = 0
67              reward += previous_spread - current_spread
68          return reward
69
70      def _calculate_reward_transaction_cost(self, action, previous_spread,
     current_spread):
71          """Rewards the agent for holding a position when the spread
    converges."""
72          reward = 0
73          if self.position == 1:
74              reward = previous_spread - current_spread
75              self.current_capital *= 1 + (previous_spread - current_spread
    )/previous_spread
76          if action == 0 and self.position == 0:
77              self.position = 1
78              reward -= current_spread * self.transaction_cost
79          elif action == 2 and self.position == 1:
80              self.position = 0
81              reward += previous_spread - current_spread - current_spread *
     self.transaction_cost
82          return reward
83
84      def render(self, mode='human', close=False):
85          """Provides a status update."""
86          print(f'Step: {self.current_step}, Position: {"Bought" if self.
    position == 1 else "Sold" if self.position == -1 else "None"}, Total
    Profit: {self.total_profit}')
87
88      def plot_trading_results(self):
89          """Plots the action of the agent and the return made by the agent
    ."""
90          # Identifying the indices for buy, sell, and hold actions
91          buy_signals = [i+self.lag_values for i, action in enumerate(self.
    actions) if action == 0]
92          sell_signals = [i+self.lag_values for i, action in enumerate(self
    .actions) if action == 2]
93          hold_signals = [i+self.lag_values for i, action in enumerate(self
    .actions) if action == 1]
94
95          plt.figure(figsize=(14, 7))
96          # Plotting the spread data
97          plt.plot(self.spread_data, label='Spread', color='gray', alpha=1)
98          # Marking buy actions with green arrows
99          plt.scatter(buy_signals, self.spread_data[buy_signals], label='
    Buy', marker='^', color='green', alpha=0.3)
100         # Marking sell actions with red arrows
101         plt.scatter(sell_signals, self.spread_data[sell_signals], label='
    Sell', marker='v', color='purple', alpha=0.3)
```

```
102         # Adding blue dots for hold actions
103         plt.scatter(hold_signals, self.spread_data[hold_signals], label='
    Hold', marker='o', color='blue', alpha=0.3)
104         # Plotting the cumulative returns
105         plt.plot([0]*20 + self.profits, label='Cumulative Return', color=
    'C2')
106 
107         plt.title('Spread Trading Strategy and Returns')
108         plt.xlabel('Day')
109         plt.ylabel('Spread / Return')
110         plt.xlim(self.spread_data.index[-100], self.spread_data.index
    [-1])
111         plt.legend()
112         plt.show()
```

A.3.2 Deep Q-learning agent

```
1 class DQLAgent:
2     def __init__(self, state_dim, num_actions, learning_rate=0.001, gamma
    =0.99, epsilon_start=0.9,
3                  epsilon_end=0.01, epsilon_decay_steps=500,
    epsilon_exponential_decay=0.99,
4                  replay_capacity=1000, tau=5, batch_size=64):
5         """Initializes the agent with all relevant parameters."""
6 
7         # Initialize agent parameters
8         self.state_dim = state_dim
9         self.num_actions = num_actions
10        self.gamma = gamma
11        self.epsilon = epsilon_start
12        self.epsilon_end = epsilon_end
13        self.epsilon_decay_steps = epsilon_decay_steps
14        self.epsilon_decay = (epsilon_start - epsilon_end) /
    epsilon_decay_steps
15        self.epsilon_exponential_decay = epsilon_exponential_decay
16        self.tau = tau
17        self.batch_size = batch_size
18        self.total_steps = 0
19 
20        # Experience replay buffer
21        self.experience = deque(maxlen=replay_capacity)
22 
23        # Build the online and target networks
24        self.online_network = self.build_model(state_dim, num_actions)
25        self.target_network = self.build_model(state_dim, num_actions,
    trainable=False)
26 
27        # Update the target network initially
28        self.update_target()
29 
30    def build_model(self, state_dim, num_actions, trainable=True):
31        """Generates a multi-layered neural network."""
```

```
32          # Define the neural network architecture
33          model = Sequential()
34          model.add(Dense(64, input_dim=state_dim, activation='relu',
    kernel_regularizer=l2(0.01), trainable=trainable))
35          model.add(Dense(64, activation='relu', trainable=trainable))
36          model.add(Dropout(0.2))
37          model.add(Dense(num_actions, activation='linear', trainable=
    trainable))  # Q-values for each action
38          model.compile(loss='mean_squared_error', optimizer=tf.keras.
    optimizers.Adam())
39          return model
40
41      def update_target(self):
42          """Updates the weight of the target network, depending on tau."""
43          # Copy weights from online network to target network
44          self.target_network.set_weights(self.online_network.get_weights()
    )
45
46      def epsilon_greedy_policy(self, state):
47          """Decides to either exploit based on current Q-values or explore
    ."""
48          # Implement epsilon-greedy policy
49          if np.random.rand() <= self.epsilon:
50              return np.random.choice(self.num_actions)  # Exploration
51          else:
52              q_values = self.online_network.predict(np.expand_dims(state,
    0))  # Exploitation
53              return np.argmax(q_values)  # Choose action with the highest
    Q-value
54
55      def memorize_transition(self, state, action, reward, next_state, done
    ):
56          """Memorizes the transitions so they can be used in the
    experience replay."""
57          # Store transition in the replay buffer
58          self.experience.append((state, action, reward, next_state, done))
59
60      def experience_replay(self):
61          """Trains the agent using experience replay."""
62          # Ensure enough samples for a batch
63          if len(self.experience) < self.batch_size:
64              return
65
66          # Randomly sample a minibatch from the replay buffer
67          minibatch = list(zip(*sample(self.experience, self.batch_size)))
68          states, actions, rewards, next_states, done = map(np.array,
    minibatch)
69
70          # Calculate target Q-values using the target network
71          next_q_values = self.target_network.predict(next_states)
72          max_next_q_values = np.max(next_q_values, axis=1)
73          target_q_values = rewards + (1 - done) * self.gamma *
    max_next_q_values
74
75          # Get predicted Q-values from the online network
```

```
76          q_values = self.online_network.predict(states)
77
78          # Set the Q-value for the taken action to the target Q-value
79          q_values[range(self.batch_size), actions] = target_q_values
80
81          # Train the online network using the adjusted Q-values
82          self.online_network.train_on_batch(states, q_values)
83
84          # Decay epsilon for exploration-exploitation balance
85          if self.epsilon > self.epsilon_end:
86              self.epsilon -= self.epsilon_decay
87          else:
88              self.epsilon *= self.epsilon_exponential_decay
89
90          # Periodically update the target network
91          self.total_steps += 1
92          if self.total_steps % self.tau == 0:
93              self.update_target()
```

Bibliography

[Avellaneda and Lee,] Avellaneda, M. and Lee, J.-H. Statistical Arbitrage in the U.S. Equities Market. Technical report.

[Balesse and Lemahieu, 2024] Balesse, L. and Lemahieu, E. (2024). Simulation of Multivariate Financial Time Series Data for Portfolio Optimization Student. Technical report.

[Bertram, 2010] Bertram, W. K. (2010). Analytic solutions for optimal statistical arbitrage trading. *Physica A: Statistical Mechanics and its Applications*, 389(11):2234–2243.

[Caldeira and Moura,] Caldeira, J. F. and Moura, G. V. Selection of a Portfolio of Pairs Based on Cointegration: A Statistical Arbitrage Strategy. Technical report.

[Chen et al., 2019] Chen, H. J., Chen, S. J., Chen, Z., and Li, F. (2019). Empirical investigation of an equity pairs trading strategy. *Management Science*, 65(1):370–389.

[Cummins and Bucca, 2011] Cummins, M. and Bucca, A. (2011). Electronic copy available. Technical report.

[Deng et al., 2017] Deng, Y., Bao, F., Kong, Y., Ren, Z., and Dai, Q. (2017). Deep Direct Reinforcement Learning for Financial Signal Representation and Trading. *IEEE Transactions on Neural Networks and Learning Systems*, 28(3):653–664.

[Do and Faff,] Do, B. and Faff, R. Does Simple Pairs Trading Still Work? Technical Report 4.

[Elliott et al., 2005] Elliott, R. J., Van Der Hoek, J., and Malcolm, W. P. (2005). Pairs trading. In *Quantitative Finance*, volume 5, pages 271–276.

[Engelberg et al.,] Engelberg, J., Gao, P., Jagannathan, R., Battalio, R., Corwin, S., Cosimano, T., Da, Z., Guntay, L., Loughran, T., Pulvino, T., Schultz, P., and Shive, S. Comments Welcome An Anatomy of Pairs Trading: the role of idiosyncratic news, common information and liquidity We are grateful to. Technical report.

[Fermanian, 2020] Fermanian, A. (2020). Functional linear regression with truncated signatures.

[Futter et al.,] Futter, O., Horvath, B., and Wiese, M. Signature Trading: A Path-Dependent Extension of the Mean-Variance Framework with Exogenous Signals. Technical report.

[Gatev et al., 2006] Gatev, E., Goetzmann, W. N., and Rouwenhorst, K. G. (2006). Pairs trading: Performance of a relative-value arbitrage rule.

[Gregnanin et al., 2023] Gregnanin, M., De Smedt, J., Gnecco, G., and Parton, M. (2023). Signature-Based Community Detection for Time Series.

[Gyurkó et al., 2013] Gyurkó, L. G., Lyons, T., Kontkowski, M., and Field, J. (2013). Extracting information from the signature of a financial data stream.

[Han et al., 2023] Han, C., He, Z., and Toh, A. J. W. (2023). Pairs trading via unsupervised learning. *European Journal of Operational Research*, 307(2):929–947.

[Huck, 2009] Huck, N. (2009). Pairs selection and outranking: An application to the S&P 100 index. *European Journal of Operational Research*, 196(2):819–825.

[Huck, 2015] Huck, N. (2015). Pairs trading: does volatility timing matter? *Applied Economics*, 47(57):6239–6256.

[Huck and Afawubo, 2015] Huck, N. and Afawubo, K. (2015). Pairs trading and selection methods: is cointegration superior? *Applied Economics*, 47(6):599–613.

[Jacobs et al., 2014] Jacobs, H., Weber, M., Mclean, D., Nyberg, P., Tetlock, P., and Zdorovtsov, V. (2014). On the determinants of pairs trading profitability. Technical report.

[James et al., 2012] James, G., Witten, D., Castle, T., and Tibshirani, R. (2012). *Introduction to Statistical Learning: With Applications in R*. Springer.

[Jansen, 2022] Jansen, S. (2022). Deep Reinforcement Learning: Building a Trading Agent.

[Karpe et al., 2020] Karpe, M., Fang, J., Ma, Z., and Wang, C. (2020). Multi-agent reinforcement learning in a realistic limit order book market simulation. In *ICAIF 2020 - 1st ACM International Conference on AI in Finance*. Association for Computing Machinery, Inc.

[Krauss, 2017] Krauss, C. (2017). STATISTICAL ARBITRAGE PAIRS TRADING STRATEGIES: REVIEW AND OUTLOOK. *Journal of Economic Surveys*, 31(2):513–545.

[Lemahieu et al.,] Lemahieu, E., Boudt, K., and Wyns, M. Generating drawdown-realistic markets using path signatures. Technical report.

[Levin et al., 2013] Levin, D., Lyons, T., and Ni, H. (2013). Learning from the past, predicting the statistics for the future, learning an evolving system.

[Lilian Weng, 2018] Lilian Weng (2018). A (Long) Peek Into Reinforcement Learning.

[Liu et al., 2018] Liu, X.-Y., Xiong, Z., Zhong, S., Yang, H., and Walid, A. (2018). Practical Deep Reinforcement Learning Approach for Stock Trading.

[Lyons, 2014] Lyons, T. (2014). Rough paths, Signatures and the modelling of functions on streams.

[Panos Patrinos, 2023] Panos Patrinos (2023). *Optimization*. Leuven.

[Patrinos, 2023] Patrinos, P. (2023). *Optimization*. Leuven.

[Rad et al.,] Rad, H., Kwong, R., Low, Y., and Faff, R. The profitability of pairs trading strategies: distance, cointegration, and copula methods. Technical report.

[Ritu Santra,] Ritu Santra. Tests for Stationarity in Time Series Dickey Fuller Test & Augmented Dickey Fuller(ADF) Test.

[Sarmento and Horta, 2020] Sarmento, S. M. and Horta, N. (2020). Enhancing a Pairs Trading strategy with the application of Machine Learning. *Expert Systems with Applications*, 158.

[sckit-learn,] sckit-learn. RobustScaler.

[skit-learn,] skit-learn. Common pitfalls in the interpretation of coefficients of linear models.

[Stefan Jansen, 2020] Stefan Jansen (2020). *Machine learning for Algorithmic Trading*. Packt Publishing, 2 edition.

[Sutton and Barto, 2018] Sutton, R. and Barto, A. (2018). *Reinforcement Learning, An Introduction*. The MIT Press, Cambridge, 2 edition.

[Vidyamurthy, 2004] Vidyamurthy, G. (2004). *Pairs Trading: Quantitative Methods and Analysis*. Wiley.

[Xie et al., 2014] Xie, W., Liew, R. Q., Wu, Y., and Zou, X. (2014). Pairs Trading with Copulas. Technical report.

[Zhang et al., 2019] Zhang, Z., Zohren, S., and Roberts, S. (2019). Deep Reinforcement Learning for Trading.