UiO : Department of Mathematics
University of Oslo

Portfolio Optimisation under Rough Stochastic Volatility via Machine Learning

Kewei Wang

Master's Thesis, Spring 2019

Contents

List of Figures

List of Tables

Abstract

In this thesis we investigate the problem of optimising stock portfolios by using methods from machine learning. The simple model of Black Scholes for the dynamics of stock prices is considered. This model has deficiency that it is not able to describe the market realistically due to constant volatility. The latter problem can be overcome by introducing rough volatility model, which is empirically shown to provide better predictions. We look at the Oslo Børs Index prices and following some specific company stock prices, we also consider also the currency market, namely Euro (EUR) to Norwegian Kroner (NOK). Finally we try to predict the rough volatility and make new predictions for stock dynamics based on a geometric Brownian model with rough volatility.

Acknowledgements

This thesis was highly influenced by my supervisors from the University of Oslo, Frank Nobert Proske, and Sven Haadem from FinStart Nordic. I would thank Frank for giving me the opportunity of doing this project under his supervision. His dedication and interest in the topic of the thesis has inspired me to learn and ask questions in a way I have not thought of. I would also like to thank Sven for taking the time to be my co-supervisor for this thesis. His encouragement and hard work ethics have influenced both my academic understanding and personal inspiration in the field of applied mathematics. Providing me with expertise during the writing of this project, I would also thank Erlend Storrøsten at FinStart Nordic for inspirational conversations and motivating words through the writing process. Taking the time to chat through hard problems. I'm grateful for everyone at FinStart Nordic who made me feel like a part of their team. This has made the writing process much more enjoyable.

Last but not least I would like to thank my family for their encouragement and motivation through the process. Specially my girlfriend Sissi, for giving me support through the writing process.

Introduction

Financial markets have been known to be hard to predict based on the weak Efficient Market hypothesis, see section 2.1.1 namely the information coming from the future asset prices. Stock price models have been studied over the decades, where the most known model is the Black-Scholes model.

In chapter 1 we will discuss the scope of problems in connection with portfolio selection as studied in this thesis, and introduce the necessarily notation for this thesis. This chapter covers e.g. the problem of portfolio allocation, and a discussion of transaction costs.

In chapters 2 and 3, we introduce the relevant theory for this thesis. Focusing on mathematical finance in chapter 2, and introducing machine learning in chapter 3, I find it educationally beneficially to write chapter 3, since machine learning has become more and more relevant in practice. I see that much work done by the FinStart Nordic team in Oslo relates to the field of applied machine learning. This motivates me to give a description of methods used in practice, which I also believe can be useful for the reader of this thesis who may be from academia or industry.

Chapter 4 is the core of this thesis. We look at how models are implemented, and explain the methodology based on empirical data from the financial market. This chapter also includes the simulated results such the rough volatility found in the data set, and the estimated parameters connected to the model. We also study the Geometric Brownian motion with rough volatility as price model.

Chapter 5 concludes with comments on the results obtained in this thesis. In addition we briefly discuss further work that can be done in connection with this project.

Chapter 1

Portfolio Selection

The portfolio selection problem aims to optimise the best portfolio by allocating assets in a such way of achieving the best long term return. In practice this means that how we can trade stocks in the market in best way as possible. These problems were first investigated by [Markowitz, 1952], where in his work, he considered a tradeoff between expected mean return and the risk concerning the variance. This method of portfolio optimisation applied with parameters estimated from data are known to give exceptionally volatility portfolio weights. This is due to the expected mean return being hard to estimate accurately. The important message from his work is that assets could not be selected only based on the characteristics that where unique to the assets. The investor has to consider the behaviour of assets movement with other assets behaviours.

More advanced approaches have been introduced to asses the future portfolio growth by considering a model by combining the mean-variance optimisation framework with the capital asset allocation pricing model [Black and Litterman, 1992]. In general, the portfolio distribution is often heavy tailed due to uncertainty of the prices. Measuring these portfolio distribution risk can be done by the common quantile measure, Value at Risk (VaR).

In this chapter we introduce the notation used in this thesis. We will also be stating the portfolio equation, and further introducing transaction costs for asset allocation.

1.1 Mathematical Formalism

1.1.1 Financial Instrument

We assume that the portfolio consists of a finite amount of stock assets from certain companies. The assets number denoted by m. The time period is defined as the time when an asset is reallocated by either hold, buy or sell. The time between each reallocation can be in the interval of minutes, hourly, weekly or monthly. As for this project, we will be using time periods of 10 and 30 min as well as daily return. This is reasonable since the data we have in hand come from frequently trading. Further denoting the price vector with respect to the time period and m assets we have the price vector $\mathbf{v_t} = (v_t^{(1)}, v_t^{(2)}, \dots, v_t^{(m)})$. Price features such as high, low, closing and opening can be formalised by the subscription $\mathbf{v_t}^{(\mathrm{hi})}, \mathbf{v_t}^{(\mathrm{lo})}, \mathbf{v_t}^{(\mathrm{cl})}, \mathbf{v_t}^{(\mathrm{op})}$ respectively. The closing price for period $t-1$ should be the same as the opening price at t, so $\mathbf{v_{t-1}}^{(\mathrm{cl})} = \mathbf{v_t}^{(\mathrm{op})}$. The relative price change of the trading period $[t, t+1)$ is given by elementary division of the price vector

$$\mathbf{y_t} = \left(1, y_t^{(1)}, y_t^{(2)}, \dots, y_t^{(m)}\right) = \left(1, \frac{v_t^{(1)}}{v_{t-1}^{(1)}}, \frac{v_t^{(2)}}{v_{t-1}^{(2)}}, \dots, \frac{v_t^{(m)}}{v_{t-1}^{(m)}}\right).$$

Denoting the portfolio vector of a market investor $\mathbf{w_t} = (w_t^{(1)}, w_t^{(2)}, \dots, w_t^{(m)})$ where each $w_t^{(j)} \in \mathbf{w}$ corresponds to the asset i weight in time t. The portfolio vector is a weighted value where the restriction are formulated by that $w_t^{(i)} \geq 0$ and $\sum_{i=1}^{m} w_t^{(i)} = 1$ for all $t = 1, 2, \dots$. These restrictions assume that our market model will not handle short selling of assets, and the consumption of capital is non-existing.

1.1.2 Portfolio

The investor initial portfolio should be subscripted with a certain time period. Beginning with the notation of the initial portfolio value at $t = 0$, where P_0 is the starting wealth. The relative price vector can be used to calculate the change in total portfolio value in a period. We first assume that the time scope is at t with portfolio $\tilde{P}_t^i$ before reallocating, then letting P_t^i denote the portfolio after reallocating time t. First we will ignore the transaction costs. Further letting n_t^i be the number of shares in asset i in period t. Due to relative price change in $[t, t+1)$ and introducing the relative price change $y_t^i = v_t^i / v_{t-1}^i$

$$\tilde{P}_t^i = n_{t-1}^i v_t^i = \frac{v_t^i}{v_{t-1}^i} n_{t-1}^i v_{t-1}^i = y_t^i p_{t-1}^i,$$

is the relative price change due to change in time period. The allocation weight

$$w_t = \frac{P_t^i}{P_t},$$

is the fraction of investment made in asset i. Still assuming that the transaction fee is zero, denoted by μ_t, the portfolio value after allocation is

$$P_t = \mu_t \tilde{P}_t = \tilde{P}_t.$$

Summing over all assets m, the portfolio value is then

$$P_t = \sum_{i=1}^{m} y_t^i p_{t-1}^i = P_{t-1} \sum_{i=1}^{m} y_t^i w_{t-1}^i = P_{t-1} \langle \mathbf{y_t}, \mathbf{w_{t-1}} \rangle. \tag{1.1.1}$$

The symbol $\langle \cdot, \cdot \rangle$ stands for the inner product that in our case gives the factor of investment in asset j grows during the period. The shares holder's capital progress can be tracked by the portfolio vectors $P_0, P_1, \cdots \in \mathbb{R}$ describing the portfolio capital without any transaction cost.

The equation (1.1.1) does not include the transaction cost of buying and selling stocks in the real market. Consequently this becomes a problem of finding the optimal portfolio from selection optimal weight vector $\mathbf{w}$ for m assets in $t = 1, 2, \dots$ steps. [Algoet et al., 1988] maximises the conditional expected log return given the current market information up to time t so that

$$\mathbf{w_t^*} = \mathbb{E}\left[\log(P_t) \middle| \mathcal{F}_t\right] = \sup \mathbb{E}\left[\log(P_t) | \mathbf{y_{t-1}}, \dots, \mathbf{y_0}\right], \tag{1.1.2}$$

is the log optimal portfolio strategy.

1.1.3 Transaction Cost

Implementing a transaction cost introduces some complications to the previous portfolio equation (1.1.1), due to not having a closed form solution. By assuming that the initial investment portfolio is $P_0 = 1$ of a unit, the cost of buying and selling is c_p and c_s respectively. Constraining $c_s, c_p \in [0, 1]$, We let N_t is the net wealth at time t with the gross wealth at time t given by

$$P_t = N_{t-1}\langle \mathbf{w_t}, \mathbf{y_t} \rangle$$

The fee under a reallocating of a investment strategy from $\mathbf{w}_n$ to $\mathbf{w}_{n+1}$. The capital of asset j moves from $w_t^{(j)} y_t^{(j)} N_{t-1}$ before reallocating, to $w_{t+1}^{(j)} N_t$ after reallocating. If $w_t^{(j)} y_t^{(j)} N_{t-1} > w_{t+1}^{(j)} N_t$, we would sell where

$$(1 - c_s)(w_t^{(j)} x_t^{(j)} N_{t-1} - w_{t+1}^{(j)} N_t),$$

is the transaction cost. We can further generalise for m number of assets. Summing over all fees for selling m assets is then

$$\sum_{j=1}^{m} \Big((1 - c_s)(w_t^{(j)} x_t^{(j)} N_{t-1} - w_{t+1}^{(j)} N_t) \Big)^{+}$$

where $(x)^+ = \max(0, x)$, also know as the rectifier function(ReLu) in machine learning. The total income for trading m assets is then

$$\sum_{j=1}^{m} \Big\{ (w_t^{(j)} x_t^{(j)} N_{t-1} - w_{t+1}^{(j)} N_t)^+ - c_s (w_t^{(j)} x_t^{(j)} N_{t-1} - w_{t+1}^{(j)} N_t)^+ \Big\}, \quad (1.1.3)$$

with transaction fee c_s. In general, c_s is proportional to the investment amount, usually by 2-3% depending on the investment instrument and the bank of choice. We also assume that the investor's portfolio is only allocated in stocks, meaning when an asset is sold, we immediately buy new assets. The cost of obtaining new assets is then c_p, with the relation to selling fee given as

$$\begin{aligned}\sum_{j=1}^{m} &\Big\{ (w_t^{(j)} x_t^{(j)} N_{t-1} - w_{t+1}^{(j)} N_t)^+ - c_s (w_t^{(j)} x_t^{(j)} N_{t-1} - w_{t+1}^{(j)} N_t)^+ \Big\} \\ &= \sum_{j=1}^{m} \Big\{ (w_{t+1}^{(j)} N_t - w_t^{(j)} x_t^{(j)} N_{t-1})^+ + c_p (w_{t+1}^{(j)} N_t - w_t^{(j)} x_t^{(j)} N_{t-1})^+ \Big\},\end{aligned}$$

where cost of buying new stocks is essentially an extra c_p added to the unit, so $1 + c_p$. An alternative way of seeing the is by noting that the portfolio value shrinks (given no profit or loss is made) with a constant μ_t for a given trading period t, with the relation to $P_t = \mu_t P_{t-1}$. The constant μ is then the transaction remainder factor, which we will be determined by an approximation given in [Jiang et al., 2017] where μ is approximated. We let c denote the transaction cost where $c = c_p = c_s$.

$$\mu = c \sum_{j=1}^{m} |w_t^{(j)} - w_{t+1}^{(j)}| \tag{1.1.4}$$

Describing the cost of moving the m assets a step ahead with cost c. The reasoning can be found in [Jiang et al., 2017].

Chapter 2

Theoretical Framework

In this chapter we present the necessary theoretical framework needed in our thesis. This also includes some economic theory about the financial market. Further we also recall some basic concepts from probability theory. Then we pass in review some basic elements and results from stochastic analysis, which we want to apply to the modelling of the dynamics of stock prices. We study the Merton problem for portfolio optimisation. Finally we discuss rough volatility stochastic volatility models and concludes this chapter with an introduction to the theory of risk measures.

2.1 Economic Theory

Doing finance requires understanding a broader aspect of the world of economics. This section will introduce some economic theory related to the capital market.

2.1.1 Efficient Market Hypothesis

The capital market in general can be described by the capital allocation of investors. The stock market purpose is for investors to make investments in ownership of firms under certain assumptions that the investments are fully informed of the market information, which should be "efficient". The article of [Malkiel and Fama, 1970] presents three ways of dividing market information given certain conditions. The proposed theory is the Efficient Market Hypothesis (EHM), where the three forms are namely the weak form, semi-strong form and strong form. The EMH implies that the market information only depends on the prices, and thus knowing more then the prices, will not give a advantage. Future knowledge is also incorporated in future prices, meaning that an market participant gains knowledge in the same period as the market in time time of stock price changes. The three form are given as follows:

- **Weak form:** All prices on the capital market fully reflect the past history prices, this includes all other information such as trading volume or market news. This claim is based on the assumption that stock prices on the market are unpredictable and independent.
- **Semi-strong form:** This extends the previous form, where market movements changes at a fast pace as public information such as annual firm report or firm announcements are known.
- **Strong form:** The strong form of EHM implies that market fluctuations also reflect the fact that certain groups have access to information that are not available. This can be information that a firm holds private and are not publicly announced. This knowledge does

effect the firm leaders believes but not necessarily have effect on the market participants and their decisions.

2.1.2 Fundamental Financial Time Series Properties

Estimating properties of the financial market are by means an important aspect of gaining valuable knowledge. The general framework has been introduced by [Cont, 2001] for asset returns. This knowledge is common for a range of financial instruments and market, which are classified into 11 stylised facts.

1. **Absence of autocorrelations:** (linear) autocorrelations of asset return are often insignificant, except for very small intraday time scale ($\simeq$ 20 minutes) for which microstructure effects come into play.

2. **Heavy tails:** the (unconditional) distribution of returns seems to display a power-law or Pareto-like tail, with a tail index that is finite, higher than two and less than five for most data sets studied. In particular this excludes stable laws with infinite variance and the normal distribution. However the precise form of the tail is difficult to determine.

3. **Gain/loss asymmetry:** One observes large drawdowns in stock prices and stock index values but not equally large upward movements.

4. **Aggregational Gaussianity:** As one increases the time scale Δt over which returns are calculated, their distribution looks more and more like a normal distribution. In particular, the shape of the distribution is not the same at different time scales.

5. **Intermittency:** Returns displayed at any time scale has a high degree of variability. This is quantified by the presence of irregular bursts in time series of a wide variety of volatility estimators.

6. **Volatility clustering:** Different measure of volatility display a positive autocorrelation over several days, which quantifies the fact that high-volatility events tend to cluster in time.

7. **Conditional heavy tails:** The residual time series exhibits heavy tails even after correcting for volatility clustering via e.g. GARCH-models, although they are less heavy than before clustering correction.

8. **Slow decay of autocorrelation in absolute returns:** The autocorrelation of absolute returns decreases as the time lag is increased, sometimes interpreted as a long-range dependence sign.

9. **Leverage effect:** Most measure of volatility and return of an asset are negatively correlated.

10. **Volume/volatility correlation:** Trading volume is correlated with all measures of volatility.

11. **Asymmetry in time scales:** Long time scales measure of volatility predict short time scales volatility better then the contrary.

2.2 Point Estimation

The expected value, variance and correlation are common parameters of common probability density function. These parameters are often estimated due to being unknown in nature. Random variables of $X_1, X_2, \dots, X_n$ with observation $x_1, x_2, \dots, x_n$ have density function $f(x; \boldsymbol{\theta})$ with the unknown parameter $\boldsymbol{\theta}$. The approximate values of $\boldsymbol{\theta}$ given the point estimation is denoted $\hat{\boldsymbol{\theta}}$, based on the data observation $x_1, x_2, \dots, x_n$. We define a point estimation as a function of observed measured values defined by

$$\hat{\boldsymbol{\theta}} = g(x_1, x_2, \dots, x_n).$$

For fixed observations of the estimator vector $\hat{\boldsymbol{\Theta}}$ is

$$\hat{\boldsymbol{\Theta}} = g(X_1, X_2, \dots, X_n).$$

2.2.1 Properties of Point Estimator

The distribution for a single random variable $\hat{\Theta}$ determine what values $\hat{\theta}$ can be, hence investigating whether the point estimator is biased, consistent and efficient is of interest. The estimator is called unbiased if

$$\mathbb{E}[\hat{\Theta}] = \theta,$$

and biased if quality does not apply. The variance of the estimator can denoted $Var[\hat{\Theta}]$. Estimating the variance is done by using the sample variance $\hat{\sigma}^2$

$$\hat{\sigma}^2 = \frac{1}{n-1}\sum_{i=1}^{n}(\hat{\theta}_i - \bar{\theta})^2,$$

where $\bar{\theta} = \frac{1}{n}\sum_{i=1}^{n}\hat{\theta}_i$ is the average of the sample of estimations. In large samples, the asymptotic properties of the estimator can be of interest. The estimator $\hat{\Theta}_n$ for sample size n is said to be consistent if for every $\epsilon > 0$

$$Pr(|\hat{\Theta}_n - \theta| > \epsilon) \to 0, \quad \text{when} \quad n \to \infty.$$

For two unbiased estimators Θ_1 and Θ_2, then Θ_1 is said to be more efficient than Θ_2 if

$$Var[\Theta_1] < Var[\Theta_2].$$

2.2.2 Point Estimators

We introduce some different point estimators, that are beneficial in different settings.

Least squares estimation

Consider a sample $x_1, x_2, \ldots, x_n$ of the random variable $X_1, X_2, \ldots, X_n$, further assuming that the expected value for each X_i is $\mathbb{E}[X_i] = \mu_i(\theta)$ for $i = 1, 2, \ldots, n$ and $\mu_i(\theta)$ function that is known except for θ. Then $X_i = \mu_i(\theta) + \epsilon_i$, where ϵ_i are assumed to be i.i.d. with expected value 0. The squared sum error is then defined as

$$Q(\theta) = \sum_{i=1}^{n}(x_i - \mu_i(\theta))^2.$$

The value of $\hat{\theta}$ that minimises $Q(\theta)$ is the estimate of θ, so

$$\hat{\theta} = \theta^* = \arg\min_{\theta} Q(\theta).$$

For all the $\mu_i\theta$ are identical, we have that

$$\frac{\partial Q}{\partial \theta} = -2\mu'(\theta)\sum_{i=1}^{n}(x_i - \mu(\theta)),$$

setting this equal to 0, give that $\mu(\theta) = \frac{1}{n}\sum_{i=1}^{n} x_i = \bar{x}$, which can be solved for θ and is the least square estimate.

Maximum likelihood estimation

The maximum likelihood estimation (MLE) defines the value for unknown parameters that are most likely for a set of sample with a known probability function. Assuming that we have a joint probability function $pdf(x_1, x_2, \ldots, x_n|\boldsymbol{\theta})$ and the set of $\boldsymbol{\theta}$ that maximises the likelihood function is the MLE, $\boldsymbol{\theta}^*$ as

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} L(\boldsymbol{\theta}).$$

The estimated vector, $\hat{\boldsymbol{\theta}} = \boldsymbol{\theta}^*$, in the case of i.i.d. sample the likelihood function is then

$$L(\boldsymbol{\theta}) = pdf(x_1, x_2, \ldots, x_n|\boldsymbol{\theta}) = \prod_{i=1}^{n} pdf(x_i|\boldsymbol{\theta}).$$

Further the log-likelihood is used for computational advantages as the likelihood estimator $\boldsymbol{\theta}^*$ as the logarithm function is strictly increasing, then we have that

$$\log L(\boldsymbol{\theta}) = \log\left(\prod_{i=1}^{n} pdf(x_i|\boldsymbol{\theta})\right) = \sum_{i=1}^{n} \log\left(pdf(x_i|\boldsymbol{\theta})\right).$$

[Devore and Berk, 2007]

2.3 Financial Return

Financial returns are important for market interpretation, the estimation method of future asset prices are unknown, with many methods to choose from. Uncertainty is often measured in terms of price changes given a time horizon, such as relative price change, absolute price change and the log price change. The absolute price change at time t can be defined as

$$\Delta S_t = S_t - S_{t-1},$$

where S_t is the actual price at time t. Relative price changes, are often preferred as this measure are compared between assets on different price levels. The percentage return is then defined as

$$r_t^{perc} = \frac{S_t - S_{t-1}}{S_{t-1}} = \frac{S_t}{S_{t-1}} - 1.$$

The log price change, or log-return are also common, and defined as

$$r_t^{log} = \log \frac{S_t}{S_{t-1}}. \tag{2.3.1}$$

In a multi-period model, the log-return can be computed by the sum of single-period returns across the time interval, T. This percentage returns are additive across the assets, i, which yields the return of portfolio asset calculated as weighted sum of the individual returns. Based on the different application, the return measure should be chosen accordingly as aggregation convenience differ between the two metrics.

2.3.1 Expected Return

A random variable X has the expected mean denoted $\mathbb{E}[X] = \mu$. For a probability function $f(x)$, the mean is $\int_{-\infty}^{\infty} xf(x)dx$. The expected value is more realistic appropriate due to the actual expected value is rarely known. Expected values are often calculated by historical data where the estimated arithmetic mean of historic returns is defined as

$$\hat{\mu} = \frac{1}{T} \sum_{i=1}^{T} r_i. \tag{2.3.2}$$

The log-returns are suitable for such estimation due to its aggregating property. As for the percentage returns, there will be an overestimate of the result, unless the time period is limited. We define the geometric mean for a percentage return given as

$$\hat{\mu} = \left(\prod_{i=1}^{T} (1 + r_i) \right)^{1/T} - 1,$$

is unbiased during a period.

2.4 Stochastic Finance

We begin this section by introducing by some basic probability concepts used in finance, as e.g. the Brownian motion. We then move on to the well known Black Scholes model for stock prices. We will also define some other mathematical preliminaries used for our thesis.

The sample space of Ω contains events (i.e. subsets) to which we can assign probabilities. The following definition for collections of subsets in Ω, the events in a class $\mathcal{F}$, known as the σ-algebra is given by

Definition 2.4.1 (σ-Algebra)**.** *The σ-algebra $\mathcal{F}$ on a given non-empty set Ω is a family $\mathcal{F}$ of subsets of Ω with the following properties*

1. $\emptyset \in \mathcal{F}$.

2. $F \in \mathcal{F} \implies F^C \in \mathcal{F}$, *where* $F^C = \Omega \backslash F$ *is the complement of* F *in* Ω.

3. $A_1, A_2, \dots \in \mathcal{F} \implies A := \bigcup_{i=1}^{\infty} A_i \in \mathcal{F}$.

Such a class $\mathcal{F}$ contains all the events that we are interested in. $(\Omega, \mathcal{F})$ is the measurable space. The probability measure denoted by $\mathbb{P}$ on the measurable space $(\Omega, \mathcal{F})$ is a function $\mathbb{P} : \mathcal{F} \mapsto [0, 1]$, such that

- $\mathbb{P}(\emptyset) = 0, \mathbb{P}(\Omega) = 1,$

- if $A_1, A_2, \cdots \in \mathcal{F}$ and $\{A_i\}_{i=1}^{\infty}$ are disjoint, then

$$\mathbb{P}\left(\bigcup_{i=1}^{\infty}\right) = \sum_{i=1}^{\infty} \mathbb{P}(A_i).$$

The triplet $(\Omega, \mathcal{F}, \mathbb{P})$ is then the probability space.

Definition 2.4.2 ($\mathcal{F}$-Measurable). *The subsets $F \subset \Omega$, which belong to $\mathcal{F}$ are called $\mathcal{F}$-Measurable sets. We have the following interpretation in connection with a probability measure $\mathbb{P}$:*

$$\mathbb{P}(F) = \text{" the probability that } F \text{ occurs".}$$

Definition 2.4.3. *Given a family $\mathcal{U}$ of subsets of Ω, there is a smallest $\sigma-$algebra $\mathcal{H}_{\mathcal{U}}$ containing $\mathcal{U}$*

$$\mathcal{H}_{\mathcal{U}} = \bigcap \{\mathcal{H} | \mathcal{H} \ \sigma\text{-algebra of } \Omega, \mathcal{U} \subset \mathcal{H}\}.$$

known as the σ-algebra generated by $\mathcal{U}$. The Borel σ-algebra on Ω is the $\sigma-$algebra $\mathcal{H}_{\mathcal{U}}$, where $\mathcal{U}$ is the collection of all open sets of a (topological) space Ω.

Definition 2.4.4 (Random Variable). *Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space. Then a random variable is a real-valued function X defined on the sample space Ω with the property that for every Borel subset B of $\mathbb{R}$, the subset of Ω given by*

$$\{X \in B\} = \{\omega \in \Omega : X(\omega) \in B\},$$

is in the σ-algebra $\mathcal{F}$.
[Shreve, 2004]

The general stochastic process X_t is then defined as

Definition 2.4.5 (Stochastic Process). *A stochastic process X is a collection of random variables*

$$(X_t, t \geq 0) = (X_t(\omega), t \geq 0, \omega \in \Omega),$$

defined on the space Ω.
[Shreve, 2004]

Definition 2.4.6 (Filtration). *The collection* $(\mathcal{F}_t, t \geq 0)$ *of* $\sigma-$*fields on* Ω *is called a filtration if*

$$\mathcal{F}_s \subset \mathcal{F}_t, \textit{ for all } 0 \leq s \leq t.$$

The filtration is the increment of information stream.

Definition 2.4.7 (Adaptedness). *The stochastic process* X_t *is said to be adapted to the filtration* $(\mathcal{F}_t, t \geq 0)$ *if*

$$\sigma(X_t) \subset \mathcal{F}_t, \textit{for all } t \geq 0.$$

Definition 2.4.8 (Martingale). *A stochastic process* X_t *is called a martingale with respect to the filtration* $\mathcal{F}_t$ *if it is adapted,* $\mathbb{E}[|X_t|] < \infty$, *for all t, and*

$$\mathbb{E}[X_t|\mathcal{F}_s] = X_s$$

for every $0 \leq s \leq t \leq$ *holds.*
[Benth, 2003]

The martingale definition can be thought as the best prediction of X_t under the information known up to time $s \leq t$.

2.5 Stochastic Processes

In this section we will take a look at different types of stochastic processes and their properties.

2.5.1 Brownian Motion

An application of Brownian motion in finance is e.g. to description of the fluctuation of asset prices. The Brownian motion, denoted W_t, is a stochastic process with the following properties:

Definition 2.5.1 (Brownian Motion). *Let* $(\Omega, \mathcal{F}, \mathbb{P})$ *be a probability space. A stochastic process* W_t *is a Brownian motion if for all* $0 = t_0 < t_1 < \cdots < t_n$, *the increments*

$$W(t_1) - W(t_0), W(t_2) - W(t_1), \ldots, W(t_n) - W(t_n - 1) \qquad (2.5.1)$$

are independent and each of the increments is normally distributed with

$$\mathbb{E}[W(t_{i+1}) - W(t_i))] = 0$$
$$Var[W(t_{i+1}) - W(t_i))] = t_{i+1} - t_i,$$

and if $\omega \in \Omega$, $W_t(\omega)$ is a continuous function in t with $W_0 = 0$.
[Shreve, 2004]

Because of the properties of the Brownian motion, the random values $W(t_i), i = 1, 2, \dots$ are jointly normally distributed, whose joint distribution is determined by the covariance structure. Each $W(t_i)$ has mean zero, and the covariance of $W(s)$ and $W(t)$ is

$$\mathbb{E}[W(s)W(t)] = \min(s, t).$$

The Brownian motion paths $t \mapsto W_t(\omega), \omega \in \Omega$ are useful for describing the stock price movements. For each ω we will have a realisation of a path, namely the sample paths $(t \mapsto W_t(\omega))$ of the Brownian motion. These paths will have the following properties:

Proposition 2.5.1. *Let $W(t)$ be Brownian motion paths, then the following properties holds:*

1. *for almost every $\omega \in \Omega$, the path $W(t, \omega)$ is continuous.*

2. *for almost every $\omega \in \Omega$, the path $W(t, \omega)$ is not differentiable.*

[Mikosch, 1998]

2.5.2 Itô's Lemma

Let us consider an adapted stochastic process X_t, which satisfies the stochastic differential equation (SDE):

$$X_t = X_0 + \int_0^t \mu(s, X_s)ds + \int_0^t \sigma(s, X_s)dW_s. \tag{2.5.2}$$

Here ds-integral is the usual integral, and $\mu, \sigma : [0, 1] \times \mathbb{R} \to \mathbb{R}$ are Borel measurable functions and W_t a one-dimensional Brownian motion. The integral with respect to the differential dW_s is a so-called Itô integral of the form

$$\int_0^t X_s dW_s, \tag{2.5.3}$$

where X_s is an Itô integrable stochastic process, see [Mikosch, 1998] for its construction. The class of Itô integrable processes is defined as follows

Definition 2.5.2 (Itô Integrability). *A stochastic process X_s is called Itô integrable on the interval $[0,t]$ if:*

1. *X_s is adapted for all $s \in [0,t]$*
2. *$\int_0^t \mathbb{E}[X_s^2] ds < \infty$*

The Itô integral (2.5.3) is itself a stochastic process as it is parametrised by time t, and the process is adapted over every time interval since it is a limit of a sum of Brownian increments. The following properties of the Itô integral are

Theorem 2.5.1 (Expectation and Variance). *The expectation of the Itô integral is*

$$\mathbb{E}\left[\int_0^t X_s dB_s\right] = 0,$$

and the variance is

$$Var\left[\int_0^t X_s dB_s\right] = \int_0^t \mathbb{E}\left[X_s^2\right] ds.$$

[Mikosch, 1998]

Definition 2.5.3 (Semi-martingale). *A stochastic process $X(t)$ is semi-martingale if there exist two Itô integrable stochastic processes $Y(t)$ and $Z(t)$ such that*

$$X(t) = X(0) + \int_0^t Y_s dW_s + \int_0^t Z_s ds.$$

Knowing the definition and properties of the Itô integral, we now can state the Itô lemma in the following special case:

Brownian Motion Itô's Lemma

Theorem 2.5.2 (Itô's formula for Brownian motion)**.** *Let $f : \mathbb{R} \to \mathbb{R}$ be two times continuously differentiable function, then the formula holds*

$$f(W_t) = f(W_s) + \int_s^t f'(W_u)dW_u + \frac{1}{2}\int_s^t f''(W_u)du$$

[Benth, 2003]

2.5.3 Geometric Brownian Motion

We introduce a process which is derived from the Brownian motion, namely the Geometric Brownian motion. This process is also known as the Black-Scholes model for stock prices. In order to obtain this process a as solution to a SDE, we need the following general Itô Lemma:

Theorem 2.5.3 (General Itô Lemma)**.** *Assume that $f(t, x)$ is a function which is once continuously differentiable in t and twice continuously differentiable in x, and let $X(t)$ be a semi-martingale. Then*

$$\begin{aligned} f(t, X(t)) =& f(0, X(0)) + \int_s^t Y(s)\frac{\partial f(s, X(s))}{\partial x}dW_s \\ &+ \frac{\partial f(s, X(s))}{\partial t} + Z(s)\frac{\partial f(s, X(s))}{\partial x} + \frac{1}{2}Y^2(s)\frac{\partial f(s, X(s))}{\partial x^2}ds. \end{aligned}$$

Definition 2.5.4 (Geometric Brownian motion)**.** *Let $S_t, t \in [0, T]$, then a stochastic process of the stock price is defined as*

$$dS_t = \mu S_t dt + \sigma S_t dW_t$$

where μ is the drift rate of S_t and σ is the standard deviation of S_t. W_t is a Brownian motion
[Black and Scholes, 1973]

Further applying Itô's lemma to the dynamics of stock prices $S(t)$, where we let $f(t, S(t)) = \log(S(t))$, we get that

$$\begin{aligned} df(t, S(t)) = d(\log(S(t))) &= \frac{dS(t)}{S(t)} - \frac{1}{2S(t)^2}(dS(t)^2) \\ &= \mu dt + \sigma dW_t - \frac{\sigma^2}{2}dt, \end{aligned}$$

which gives that

$$\log(S(t)) - \log(S(0)) = \left(\mu - \frac{\sigma^2}{2}\right) t + \sigma W_t$$

$$S(t) = S(0)\exp\left(\left(\mu - \frac{\sigma^2}{2}\right) t + \sigma W_t\right), \tag{2.5.4}$$

where we used that $(dS(t)^2) = \sigma^2 S(t)^2 (dW_t)^2$, $dt^2 = 0$ and $dtdW_t = dW_t dt = 0$, [Benth, 2003].

Application of GBM

The formula for proportional return of a stock can be defined as follows:

$$\frac{\Delta S(t)}{S(t)} = \mu \Delta t + \sigma \epsilon \sqrt{\Delta t}, \tag{2.5.5}$$

where the first component show the expected rate of return μ that a stock will earn over a short period of time Δt. The second component follows a random process where σ is the expected volatility of the stock and $\epsilon\sqrt{\Delta t}$ represents the random volatility which magnifies as the period of time increases. We assume that the stock prices are log-normally distributed with mean of the first component and a standard deviation of the uncertain component. This then leads to the following distribution of the log-price increments

$$\log \frac{S(T)}{S(0)} \sim N\left((\mu - \frac{\sigma^2}{2})T, \sigma\epsilon\sqrt{T}\right), \tag{2.5.6}$$

where $S(0)$ is the present stock price and $S(T)$ is the price at time T. The formula for simulation of the GBM stock price at any time interval $t+\Delta t$, given its price at t is shown in the following equation:

$$S(t+\Delta t) = S(t)\exp\left[(\mu - \frac{\sigma^2}{2})\Delta t + \sigma\epsilon\sqrt{\Delta t}\right]. \tag{2.5.7}$$

The expected value $\mathbb{E}(S(t))$ of the stock price at the future time t is given by:

$$\mathbb{E}(S(t)) = S(0)\exp\left(\left(\mu + \frac{\sigma^2}{2}\right) t\right). \tag{2.5.8}$$

2.5.4 Fractional Brownian Motion

The extension of the Brownian motion to the Fractional Brownian motion process is the following

Definition 2.5.5 (Fractional Brownian Motion). *Let $H \in (0,1)$. A Fractional Brownian motion (fBm) with Hurst parameter H is a centred continuous Gaussian process $B^H = (B_t^H)_{t \geq 0}$, with covariance function*

$$\mathbb{E}[B_t^H B_s^H] = \frac{1}{2}(t^{2H} + s^{2H} - |t-s|^{2H}).$$

[Nourdin, 2012]

The distribution of the Fractional Brownian motion B^H is uniquely determined by the specific covariance structure. The existence can be confirmed by checking that the covariance function is non-negative definite. We will first introduce some properties of the fBm process. When the Hurst parameter $H = 1/2$, the fBm is just the Brownian motion [Nourdin, 2012].

We formulate the self-similar property of the fBm process.

Definition 2.5.6 (Homogeneous Function). *A homogeneous function f of variable x and y is a real-valued function that satisfies*

$$f(tx, ty) = t^k f(x,y),$$

for some constant k and for all real numbers t. The constant k is the degree of homogeneity.

We note that the covariance function of the fBm is homogeneous of the order $2H$. This means the the fBm is H self-similar, for $\alpha > 0$, $\{B_{\alpha t}^H, t \in \mathbb{R}\}$ has the same distribution as $\{\alpha^H B_t^H, t \in \mathbb{R}\}$. Further noting that the fBm increments are stationary

$$\mathbb{E}[|B_t^H - B_s^H|^2] = |t-s|^{2H},$$

where $s, t \in \mathbb{R}$. When $H \in (0, \frac{1}{2}) \cup (\frac{1}{2}, 1)$, the increments of the fBm on disjoint intervals are not independent. The covariance between two increments $(B^H_{t+h} - B^H_t)$ and $(B^H_{s+h} - B^H_s)$, where $s + h \leq t,\ t - s = nh$ with the following covariance

$$\begin{aligned} R_H(n) &= \mathbb{E}[(B^H_{t+h} - B^H_t)(B^H_{s+h} - B^H_s)] \\ &= \frac{1}{2}h^{2H}((n+1)^{2H} + (n-1)^{2H} - 2n^{2H}) \\ &\sim h^{2H}H(2H-1)n^{2H-2} \to 0, \quad \text{as } n \to \infty. \end{aligned}$$

Then we have the following

- When $H \ \in (0, \frac{1}{2})$, $R_H(n) < 0$ and $\sum_{n=1}^{\infty} |R_H(n)| < \infty$,
- When $H \ \in (\frac{1}{2}, 0)$, $R_H(n) > 0$ and $\sum_{n=1}^{\infty} |R_H(n)| = \infty$.

Meaning in the both cases, the increments of the fBm process are not independent.

Mandelbrot-Van Ness Representation

In terms of the Wiener process, denoted $\{W_t, t \in \mathbb{R}\}$, with two independent processes $\{W_t, t \geq 0\}$ and $\{W_{-t}, t \ \geq 0\}$ on $[0, \infty]$. The step function can be defined as

$$h(t) = \sum_{k=1}^{n} a_k \mathbf{1}_{[s_k, t_k]}(t),$$

and the following integral

$$I(h) = \int_{\mathbb{R}} h(t) dW_t = \sum_{k=1}^{n} a_k \left(W_{t_k} - W_{s_k}\right).$$

The last integral can be extended to functions in $L^2(\mathbb{R})$, since the integral is isometric and linear. The following properties can be summarised as follows

- **Linearity:** for some $\alpha, \beta \in \mathbb{R}$ and function $f, g \in \mathbb{R}$ holds
$$I(\alpha f + \beta g) = \alpha I(f) + \beta I(g).$$
- **Mean:** $\mathbb{E}[I(f)] = 0$
- **Isometry:** $\mathbb{E}[I(f)^2] = \int_{\mathbb{R}} f(x)^2 dx$, moreover, for $f, g \in L^2(\mathbb{R})$.

- **Distribution** For $f_1, f_2, \ldots, f_n \in \mathbb{R}$ the random variables $I(f_1), I(f_2), \ldots I(f_n)$ are jointly Gaussian distributed.

We now assume that the fBm process can be defined as

$$B_t^H = I(K_H(t)) = \int_{\mathbb{R}} K_H(t,x)dW_t(x),$$

where $K_H(t)$ is some deterministic kernel defined in $L^2(\mathbb{R})$. In order to show that the integral is a fBm process we need to show that it has the same covariance function as a fBm.

Theorem 2.5.4 (Mandelbrot Van-Ness Representation of fBm)**.** *Define*

$$K_H(t,u) = (t-u)_+^\kappa - (-u)_+^\kappa,$$

where $\kappa = H - 1/2$. The Mandelbrot Van Ness representation of the fBm process in terms of integral

$$B_t^H = \left(\int_{\mathbb{R}^+} ((1+s)^\kappa - s^\kappa)^2 ds + \frac{1}{2H} \right)^{1/2} \int_{\mathbb{R}} K_H(t,u)dW_u \qquad (2.5.9)$$

Proof. Since $B_0^H = 0$ and $\mathbb{E}[B_t^H] = 0, t \geq 0$. Then for $r \geq 0$ we have

$$\mathbb{E}[(B_t^H)^2] = \int_{\mathbb{R}^+} ((1+s)^\kappa - s^\kappa)^2 ds + \frac{1}{2H} \int_{-\infty}^0 K_H^2(t,u)du + \int_0^t (t-u)^{2\kappa} du = t^{2H}.$$

For $t < 0$

$$\mathbb{E}[(B_t^H)^2] = \int_{\mathbb{R}^+} ((1+s)^\kappa - s^\kappa)^2 ds + \frac{1}{2H} \int_{-\infty}^0 K_H^2(t,u)du + \int_t^0 (-u)^{2\kappa} du = (-t)^{2H}.$$

We can see that for some $h < 0$ yields

$$\begin{aligned}
B_{s+h}^H - B_s^H &= C^{(1)}(H) \int_{-\infty}^s \left(K_H(s+h,u) - K_H(s,u)\right) dW_u \\
&+ C^{(1)}(H) \int_s^{s+h} K_H(s+h,u)dW_u \\
&= C^{(1)}(H) \int_{-\infty}^0 \left(K_H(h,u) - K_H(0,u)\right) dW_u \\
&+ C^{(1)}(H) \int_0^h K_H(h,u)dW_u \\
&= C^{(1)}(H) \int_{-\infty}^h K_H(h,u)dW_u = B_h^H,
\end{aligned}$$

where

$$C^{(1)}(H) = \left(\int_{\mathbb{R}^+} ((1+s)^{\kappa} - s^{\kappa})^2 ds + \frac{1}{2H} \right)^{1/2}.$$

Further the following holds

$$\mathbb{E}[(B^H_{s+h} - B^H_s)^2] = \mathbb{E}[(B^H_h)^2] = h^{2H}.$$

Then the covariance function is

$$\begin{aligned}\mathbb{E}[B^H_t B^H_s] &= \frac{1}{2}\left(\mathbb{E}[(B^H_t)^2] + \mathbb{E}[(B^H_s)^2] - \mathbb{E}[(B^H_{s+h} - B^H_{s+h})^2]\right) \\ &= \frac{1}{2}(t^{2H} + s^{2H} - |t-s|^{2H}),\end{aligned}$$

which is the covariance function for the Fractional Brownian motion. □

2.5.5 Predicting Volatility

We forecast the log-volatility for the Fractional Brownian process. We assume the information is generated by the fBm B^H_t with filtration $\mathcal{F}_t$. We can define the fBm process from the Mandelbrot Van-Ness representation as

$$\begin{aligned}B^H_t &= \left(\int_{\mathbb{R}^+} ((1+s)^{\kappa} - s^{\kappa})^2 ds + \frac{1}{2H} \right)^{1/2} \int_{\mathbb{R}} K_H(t,u) dW_u \\ &= a_H \int_{\mathbb{R}} K_H(t,u) dW_u,\end{aligned}$$

where

$$a_H = \left(\int_{\mathbb{R}^+} ((1+s)^{\kappa} - s^{\kappa})^2 ds + \frac{1}{2H} \right)^{1/2}.$$

The conditional expectation of the fBm is then

$$\mathbb{E}[B^H_{t+\Delta} | \mathcal{F}_t] = \mathbb{E}\left[\left(a_H \int_{-\infty}^{t} K_H(t+\Delta, u) dW_u + a_H \int_{t}^{\infty} K_H(t+\Delta, u) dW_u \right) \middle| \, \mathcal{F}_t \right].$$

We note that the last term disappears as being independent of the filtration $\mathcal{F}_t$ and the expectation is zero, i.e. $\mathbb{E}[\int_t^{\infty} K_H(t+\Delta, u) dW_u] = 0$. This leads to

$$\mathbb{E}[B^H_{t+\Delta} | \mathcal{F}_t] = a_H \int_{-\infty}^{t} K_H(t+\Delta, u) dW_u.$$

From [Gatheral et al., 2014] we have that the expected conditional value of the fBm is

$$\mathbb{E}[B^H_{t+\Delta}|\mathcal{F}_t] = C_{\Delta,H}\int_{-\infty}^{t} \frac{B^H_s}{(t-s+\Delta)(t-s)^{H+1/2}}ds.$$

This gives us the following equation when we use the definition of the fBm.

$$\mathbb{E}[B^H_{t+\Delta}|\mathcal{F}_t] = C_{\Delta,H}\int_{\mathbb{R}}\left(\int_{-\infty}^{t} a_H\frac{K_H(s,u)}{(t-s+\Delta)(t-s)^{H+1/2}}ds\right)dW_u,$$

where the inner integral w.r.t. s is just $K_H(t+\Delta,u)$, and we showed that

$$\mathbb{E}[B^H_{t+\Delta}|\mathcal{F}_t] = C_{\Delta,H}\int_{\mathbb{R}} K_H(t+\Delta,u)dW_u,$$
$$C_{\Delta,H} = \frac{\cos(\pi H)H}{H}\Delta^{H-1/2}.$$

Further to see the prediction for log-volatility, we define $\sigma_t = \exp(\theta B^H_t)$. This leads to the log-volatility being defined as

$$\log(\sigma^2_{t+\Delta}|\mathcal{F}_t) = 2\log + \theta B^H_t.$$

Further we have

$$\begin{aligned}\mathbb{E}\left[\log(\sigma^2_t)|\mathcal{F}_t\right] &= \mathbb{E}\left[\log(\sigma^2_{t+\Delta})\middle|\,\mathcal{F}_t\right]\\ &= C + 2\theta\mathbb{E}\left[B^H_{t+\Delta}|\mathcal{F}_t\right]\\ &= C + 2\theta\frac{\cos(H\pi)}{\pi}\Delta^{H+1/2}\int_{-\infty}^{t}\frac{B_s}{(t-s+\Delta)(t-s)^{H+1/2}}ds,\end{aligned}$$

where $C = 2\log(\sigma)$, and the last equation is the prediction formula for log-variance [Gatheral et al., 2014]. The variance can be presented as

$$\begin{aligned}
\operatorname{Var}\left[B^H_{t+\Delta}|\mathcal{F}_t\right] &= \mathbb{E}\left[(B^H_{t+\Delta} - \mathbb{E}[W^H_{t+\Delta}])^2|\mathcal{F}_t\right] \\
&= \mathbb{E}\left[\left(c\int_0^{t+\Delta}(t-s+\Delta)^{H-1/2}dW_s\right)^2\middle|\mathcal{F}_t\right] \\
&= \mathbb{E}\Bigg[\Bigg(c\int_0^{t+\Delta}(t-s+\Delta)^{H-1/2}dW_s \\
&+ c\int_0^t(t-s+\Delta)^{H-1/2}dW_s\Bigg)^2\Bigg|\mathcal{F}_t\Bigg] \\
&= c^2\int_t^{t+\Delta}\left((t-s+\Delta)^{H-1/2}\right)^2 ds \\
&+ c^2\int_0^t\left((t-s+\Delta)^{H-1/2}\right)^2 ds \\
&\geq c^2\int_t^{t+\Delta}\left((t-s+\Delta)^{H-1/2}\right)^2 ds.
\end{aligned}$$

The last equation follows from the strong local non-determinism of fBm, see [Berman, 1973] being a Gaussian distributed process. The variance prediction can be derived by noting that

$$\operatorname{Var}\left[B^H_{t+\Delta}|\mathbb{F}_t\right] = c\Delta^{2H},$$

where

$$c = \frac{\Gamma(3/2+H)}{\Gamma(1/2+H)\Gamma(2-2H)}.$$

We obtain that the predicted estimate for variance is given by

$$\widehat{\sigma^2_{t+\Delta}} = \exp\left[\widehat{\log(\sigma_{t+\Delta})} + 2\theta^2 c B^H_t\right]. \tag{2.5.10}$$

[Gatheral et al., 2014]

2.5.6 Simulating Fractional Brownian Motion

We introduce a method of simulating the Fractional Brownian motion, where the idea is from [Shevchenko, 2014]. The method uses the idea that a Gaussian vector denoted γ with mean μ and covariance matrix C as $\gamma = \mu + S\epsilon$, where $SS^T = C$ and ϵ is a standard Gaussian vector.

Finding S matrix can be done by taking the square root of the covariance matrix C. We first define a grid of points in $[0,T]$ of points $t_k^n = \frac{kT}{N}$ for $k = 0, 1, \ldots, N$, where N is large. The task is then to simulate values of the fBm and multiply with $\frac{T}{N}^H$. We proceed with simulating B_t^H, then it suffices to simulate the increments $B_1^H, B_2^H - B_1^H, \ldots B_N^H - B_{N-1}^H$. where each increment can be denoted by $\gamma_i, i = 1, 2, \ldots, N$. Now the covariance of γ is

$$R_H(n) = \mathbb{E}[\gamma_1\gamma_{n-1}] = \frac{1}{2}((n+1)^{2H} + (n-1)^{2H} - 2n^{2H}), n \geq 1.$$

$$\mathrm{Cov}(\gamma) = \begin{bmatrix} 1 & R_H(1) & R_H(2) & \ldots & R_H(N-2) & R_H(N-1) \\ R_H(1) & 1 & R_H(1) & \ldots & R_H(N-3) & R_H(N-2) \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ R_H(N-2) & R_H(N-3) & R_H(N-4) & \ldots & 1 & R_H(1) \\ R_H(N-1) & R_H(N-2) & R_H(N-3) & \ldots & R_H(1) & 1 \end{bmatrix}.$$

We extend to a bigger model to solve $SS^T = \mathrm{Cov}(\gamma)$. Let $M = 2(N-1)$, $c_0 = 1$ and

$$c_i = \begin{cases} R_H(i), & i = 1, 2, \ \ldots N-1 \\ R_H(M-i), & i = B, N+1, \ldots, M-1. \end{cases}$$

Also defining the circulant matrix

$$C = \begin{bmatrix} c_0 & c_1 & c_2 & \ldots & c_{M-1} & c_{M-1} \\ c_{M-1} & c_0 & c_1 & \ldots & c_{M-3} & c_{M-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ c_2 & c_3 & c_4 & \ldots & c_0 & c_1 \\ c_1 & c_2 & c_3 & \ldots & c_{M-1} & c_0 \end{bmatrix}.$$

Also that the matrix $Y = (y_{jk})_{j,k=0}^{M-1}, with$

$$y_{jk} = \frac{1}{\sqrt{M}} \exp\left(-2\pi i \frac{jk}{M}\right).$$

We have that $YY^* = Y^*Y = I_M$, where Y^* is denoted the conjugate transposition matrix of Y, and I_M is the identity matrix. The C matrix can be given as $C = Y\Lambda(C)Y$ since it is a circulant matrix. Λ is the diagonal matrix of eigenvalues of C, and Y is unitary matrix. The matrix C is also positive definite and symmetric, which means the eigenvalues will be positive and real. The eigenvalues, which constitute the matrix Λ is given as follows

$$\lambda_k = \sum_{i=0}^{M-1} r_j \exp\left(-2\pi i \frac{jk}{M}\right), \quad fork = 0, 1, \dots M-1,$$

where r_j is the $(j+1)$th element of the first row of C. This result in that the eigenvalues $\sqrt{\lambda_1}, \sqrt{\lambda_2}, \dots, \sqrt{\lambda_{M-1}}$ are positive and real. Note that now we have $S = Y\Lambda^{1/2}Y^*$ satisfies $SS^* = SS^T = C$. The sampling of the fBm then comes to simulate the $Y\Lambda^{1/2}Q^*\epsilon$ matrix.

1. Compute the eigenvalues $\lambda_k, k = 0, 1 \dots M-1$. This can be done by the Fast Fourier Transform (FFT) for a given sequence $(\alpha_k)_{k=0}^{j-1}$, the algorithm gives

   $$\sum_{k=1}^{j-1} \alpha_k \exp\left(2\pi i \frac{nk}{j}\right), \quad n = 0, \dots, j-1,$$

   as the the power of j of 2, the method is faster in computation.

2. Take the real part of the inverse FFT method of the generated Gaussian distributed $\epsilon_1, \dots, \epsilon_M$ to get $\frac{1}{\sqrt{M}}\Lambda^*\epsilon^T$.

3. Multiply step 3. with $\sqrt{\lambda_1}, \sqrt{\lambda_2}, \dots, \sqrt{\lambda_{M-1}}$.

4. Taking FFT of step.4 to get

   $$(\gamma_0, \dots, \gamma_M)^T = \sqrt{M}Y\Lambda^{1/2}\frac{1}{\sqrt{M}}\Lambda\epsilon^T = S\epsilon^T.$$

5. Take the real part of $\gamma_0, \gamma_1 \dots$, and multiply with $\left(\frac{T}{N}\right)^H$, getting the increments fo the fBm.

6. Take cumulative sums to obtain the values of fBm.

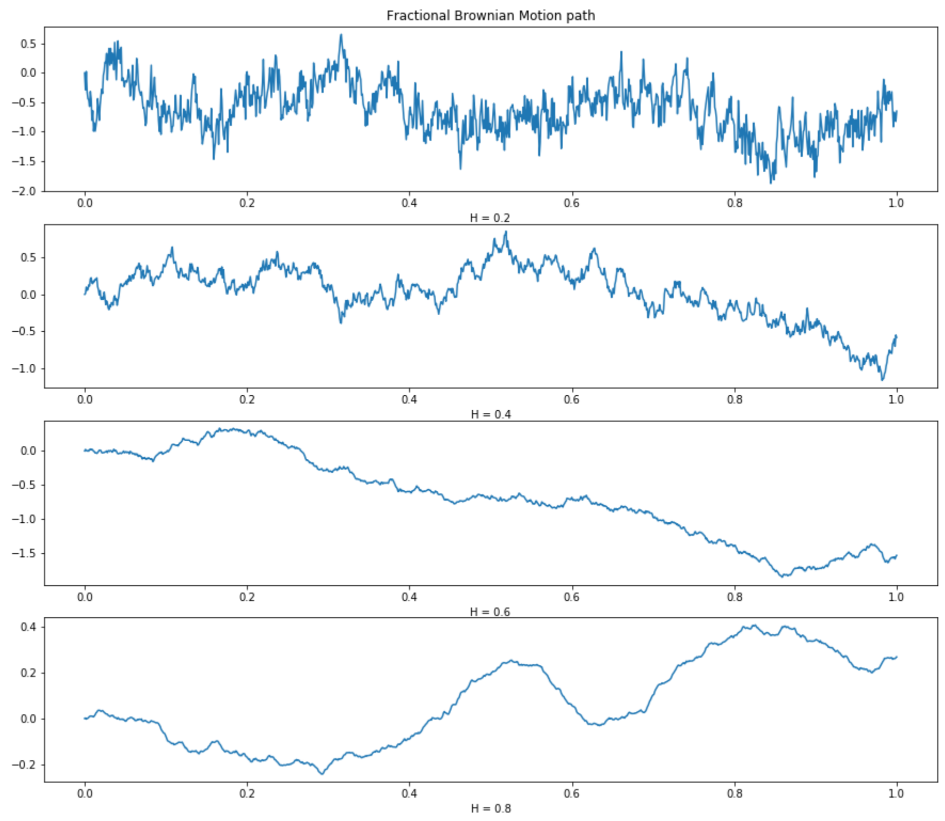

Figure 2.1: Fractional Brownian motion paths.

2.5.7 Geometric Brownian Motion with Rough Volatility

Geometric Brownian Motion with Rough Volatility (GBMRV) is extended from (2.5.4). We can now replace the volatility σ for the standard GBM model with the rough volatility model mentioned in 2.7. For the GBMRV model, the logarithmic returns can be expressed as

$$R_{H,\Delta t}(t) = \log \frac{S_H(t+\Delta t)}{S_H(t)} = (\mu - \frac{\sigma_{\mathrm{RV}}^2}{2})\Delta t + \sigma_{\mathrm{RV}} W_t. \tag{2.5.11}$$

2.6 Optimal Portfolio

We can now introduce a measures U for a portfolio value W, which measured the utility of the amount W. The utility function satisfies the following conditions:

1. $U(W)$ is a increasing function in W.
2. $U(W)$ is a concave function in W.
3. $U(W)$ is twice differentiable.

Most common utility functions are the quadratic function, power function and exponential function, where firstly the quadratic function can be given as

$$U(\omega) = \omega - a\omega^2, \quad a > 0, \quad \omega < \frac{1}{2a},$$

where a is the risk aversion parameter. The power utility with a risk aversion parameter $\gamma > 0$ has the following form

$$U(\omega) = \begin{cases} \log(\omega) & \text{for } \gamma = 0 \\ \frac{\omega^\gamma}{\gamma} & \text{for } \gamma \leq 0, \gamma \neq 0. \end{cases}$$

The last function is the exponential utility with risk aversion parameter b

$$U(\omega) = -e^{b\omega}, \quad b > 0.$$

We consider a market which consists of a stock and a risk-free investment. The price process dynamics is modelled by the geometric Brownian motion

$$dS_t = \mu S_t dt + \sigma S_t dW_t$$

and the risk-free dynamics, known as the bond denoted B_t defined as

$$dB_t = rB_t dt,$$

with interest rate $r > 0$. Further we denote the number of shares invested in the risk-free asset and the stock asset by n_t^0 and n_t^1 respectively. Then a self-financing portfolio can be defined as

Definition 2.6.1 (Self-Financing Portfolio)**.** *A portfolio strategy* (n_t^0, n_t^1) *is called self-financing if*

$$dP_t = n_t^0 dB_t + n_t^1 dS_t, \quad P_t = n_t^0 B_t + n_t^1 b_t^1 S_t$$

where P_t *is the portfolio at time point* t*.*

In words, the self-financing portfolio does not allow the investors to withdraw any money gained for consumption or investing in additional funds.

Portfolio Allocation

From the definition of the self-financing portfolio, we can show that the portfolio optimisation can be defined in terms of finding the optimal weights pair (ω^0, ω^1). We then have that

$$\begin{aligned}
\frac{dP_t}{P_t} &= \frac{n_t^0 dB_t B_t}{P_t B_t} + \frac{n_t^1 dS_t S_t}{P_t S_t} \\
&= \frac{n_t^0 B_t}{P_t}\frac{dB_t}{B_t} + \frac{n_t^1 S_t}{P_t}\frac{dS_t}{S_t} \\
&= \omega^0 \frac{dB_t}{B_t} + \omega^1 \frac{dS_t}{S_t} \\
&= \omega^0 r dt + \omega^1 \left((\mu - \frac{\sigma^2}{2}) dt + \sigma dW_t \right) \\
&= (\omega^0 r + \omega^1 \mu - \omega^1 \frac{\sigma^2}{2}) dt + \sigma dW_t
\end{aligned}$$

where we divide by P_t. Assuming the portfolio is $P_t = P_0 e^{Z_t}$, where $dZ_t = a(t)dt + b(t)dW_t,\ Z_0 = 0$ is a general stochastic process. Using the Itô lemma, we get the following solution

$$\frac{dP_t}{P_t} = \left(a + \frac{b^2}{2}\right) + bW_t.$$

The terms a and b can then be found by comparing the equations

$$a = \omega^0 r + \omega^1 \mu - \frac{\sigma^2}{2}\omega^1 - \frac{\sigma^2}{2}(\omega^1)^2, \quad b = \omega^1 \sigma.$$

This gives that the stock dynamics with $Z_0 = 0$ is

$$dZ_t = \left(\omega^0 r + \omega^1 \mu - \frac{1}{2}\omega^1\sigma^2 - \frac{1}{2}(\omega^1\sigma)^2\right) dt + \omega^1 \sigma dW_t \tag{2.6.1}$$

2.6.1 Portfolio Optimisation

The classical optimal asset allocation problem was first introduced by [Merton, 1969]. The expected utility function U_t is $\log(P_t) = \log(P_0) + Z_t$. This is then given by the following equation

$$\mathbb{E}[U_T] = \log(P_0) + \int_0^T \left(\omega^0 r + \omega^1 \mu - \frac{1}{2}\omega^1\sigma^2 - \frac{1}{2}(\omega^1\sigma)^2\right) dt, \tag{2.6.2}$$

where the integrand is a constant. Thus our goal is to find the optimal set of (ω^0, ω^1) that

$$\begin{aligned} \text{Maximise:} \quad & \left(\omega^0 r + \omega^1 \mu - \frac{1}{2}\omega^1\sigma^2 - \frac{1}{2}(\omega^1\sigma)^2\right) \\ \text{Subject to:} \quad & \omega^0 + \omega^1 = 1, \quad (\omega^0, \omega^1) \in [0,1] \end{aligned}$$

The solution of the weights are then given as follow

$$\omega^0 = 1 - \frac{\mu - \frac{\sigma^2}{2} - r}{\sigma^2}, \quad \omega^1 = \frac{\mu - \frac{\sigma^2}{2} - r}{\sigma^2}. \tag{2.6.3}$$

2.7 Stochastic Volatility Model

The classical Merton dynamic portfolio choice model returns both the return rate and the volatility of the risky asset which are assumed to be constant. However in many applications, volatility may depend on the time of return. This case, the constant volatility can't explain the dynamics of the model, since the volatility and the stock price can be correlated. Consider the following stochastic volatility model

$$dS_t = \mu S_t dt + \sigma(X_t) S_t dW_t,$$

where W_t is the Brownian motion, and now $\sigma(X_t)$ is the stochastic volatility. The models of $\sigma(X_t)$ can be on many forms. The most simple one, where $\sigma(X_t)$ is another Brownian motion W_t^* that can be correlated with the first Brownian motion W_t.

$$\operatorname{corr}(W_t, W_t^*) = \rho, \quad \rho \in [-1, 1],$$

where ρ is the correlation coefficient that can be depending on the market we want to focus on. The ρ can capture effects that the constant volatility are not able to model, such as heavy tails or skewness. The negative side of this modelling comes often in forms where the closed form of solutions can be derived, so simulations will be necessarily.

$$dX_t = \exp(\theta dB_t^H),$$

is the dynamics of the volatility. The calculations are the same as in the case of the Geometric Brownian, this yields a portfolio dynamics given by

$$\frac{dP_t}{P_t} = (\omega^0 r + \omega^1 \mu + \omega^1 \frac{\sigma^2(X_t)}{2})dt + \sigma(X_t) dW_t.$$

Assuming that the price is $P_t = P_0 e^{Z_t}$, where $Z_t = a(t)dt + b(t)dW_t$, $Z_0 = 0$, we have that

$$dZ_t = \left(\omega^0 r + \omega^1 \mu + \frac{1}{2}\omega^1 \sigma^2(X_t) - \frac{1}{2}(\omega^1 \sigma(X_t))^2\right) dt + \omega^1 \sigma(X_t) dW_t$$

2.7.1 Smoothness of Volatility

Assuming that we have observed the volatility $\sigma_0, \sigma_\Delta, \ldots, \sigma_{k\Delta}, \ldots,$ $k \in \{0, [T/\Delta]\}$, where Δ is grid mesh on $[0, T]$. Let $N = [T/\Delta]$, for some $q \geq 0$

$$m(q, \Delta) = \frac{1}{N} \sum_{k=1}^{N} |\log(\sigma_{k\Delta}) - \log(\sigma_{(k-1)\Delta}))|^q. \tag{2.7.1}$$

For some $s_q > 0$ and $b_q > 0$, as Δ tends to zero,

$$N^{qs_q} m(q, \Delta) \to b_q,$$

where s_q is the smoothness parameter which controls the regularity of the trajectory of the volatility process. The volatility process is Hölder continuous of order less than s_q. The volatility process is not directly observable in the real world setting. To estimate the smoothness, we need to approximate the volatility by the estimate of the realised variance. We simulate different $\log(m(q, \Delta))$ as a function of $\log(\Delta)$ for different q values. The scaling property between the simulated values are in expectation

$$\mathbb{E}[|\log(\sigma_\Delta) - \log(\sigma_0)|^q] = C_q \Delta^{\xi_q}, \tag{2.7.2}$$

where $\xi_q > 0$ is the regression coefficient. The estimate of H is found by regressing ξ_q against q, and is independent of q. The relationship is approximately $\xi_q \sim Hq$. The estimated volatility of out index of interest, have shown a low value of $H << 1/2$, which means rough volatility. The following plots show the relationship.

2.7.2 Rough Volatility Model

We suggest a simple model based on the empirical findings from the previous subsection, where the increments of log-volatility has a scaling property with constant smoothness parameter and their distribution being close to the Gaussian.

$$\log(\sigma_{t+\Delta}) - \log(\sigma_t) = \theta \left(B^H_{t+\Delta} - B^H_t \right), \tag{2.7.3}$$

where B^H is fbm with Hurst parameter H, and θ is a positive constant. We may write this equation under the form

$$\sigma_t = \sigma_0 \exp(\theta B^H_t),$$

where σ is another positive constant. This model however is not stationary, being a property that is desirable both of mathematical tractability and also to ensure reasonableness of the model at very large times. We introduce the a stationary model known as the Fractional Ornstein-Uhlenbeck process (fOU) denoted X_t with the stochastic differential equation

$$dX_t = \nu dB_t^H - \alpha(X_t - m)dt,$$

where ν and α are positive parameters, and $m \in \mathbb{R}$. The explicit solution is given as

$$X_t = \nu \int_{-\infty}^{t} e^{-\alpha(t-s)} dB_t^H + m. \tag{2.7.4}$$

This is a stochastic integral with respect to the fBM which is simply a pathwise Young integral. We see that

$$\sigma_t = \exp\{X_t\}, t \in [0, T],$$

where X_t satisfies equation (2.7.4). $H < 1/2$, measures the smoothness of the volatility. For $\alpha << 1/T$, the log volatility behaves locally as a fBM process.

Proposition 2.7.1. *Let B^H be a fBM and X^α be defined by equation* (2.7.4) *for a given $\alpha > 0$, then*

$$\mathbb{E}\left[\sup_{t\in[0,T]} |X_t^\alpha - X_0^\alpha - \nu B_t^H|\right] \to 0, \textit{as } \alpha \to 0.$$

We note that the Proposition implies that the model if $\alpha << 1/T$, we can proceed as if the log-volatility process were a fBM. The exact scaling property of the fBM is approximately reproduced by the fOU process when α is small.

Corollary 2.7.0.1. *Let $q > 0, t > 0, \Delta > 0$, we have that*

$$\mathbb{E}\left[|X_{t+\Delta}^\alpha - X_t^\alpha|^q\right] \to \nu^q K_q \Delta^{qH}, \ \textit{as } \alpha \to 0.$$

2.8 Risk Measure

The key goal is to find the optimal portfolio when constraints are introduced which maximises the expected return. The threshold is a measure of the distribution quantile given a certain confidence. These kinds of problem were first introduced by [Markowitz, 1952]. His work in the financial field concerns the fact that a market investor would maximise the portfolio by using the expected value and the variance as criteria. He shows that under some simple constraints, portfolio maximising can be done by diversifying allocations of assets where the weights are derived by the E-V rule to find a attainable set of E-V.

This method in practice is limited as computations are not efficient. In the past years, new methods have been developed to solve the optimal portfolio problem. Linear programming are common where the variance is estimated by the absolute variance. Further there are methods that ignore the E-V rule and use the risk as a measure itself. This means that the model can assess the worst case scenario using a distribution quantile of the portfolio returns. For an overview, see [Duffie and Pan, 1997]. We will introduce risk measure in brief and further present the conditional risk measure based on the same idea.

Assessing portfolio risk is done through the Value at Risk (VaR) or Conditional Value at Risk (CVaR), which is closely related to VaR. We will introduce both risk measures, but CVaR measure is a coherent risk measure.

[Artzner et al., 1999] denotes $\rho(X)$ as the measure of risk for a given position X, defined formally to be

Definition 2.8.1. *A risk measure is a mapping $\mathcal{X} : t \to \mathbb{R}$, where $\mathcal{X}$ is a set of real-values functions.*

The set $\mathcal{X}$ is the risk set of real-valued functions $X \in \mathcal{X}$. This can be understood as the risk over the final net stock or portfolio value. As this value is higher, the portfolio holder are prone to a riskier position. Being in a comfortable risk position in the sense of trading stock depends on the agents risk preference and acceptability. The preference might be depending on the fact that a higher risk position might lead to a higher net return. The acceptability is related to the some set $\mathcal{A}$ of final net portfolio value accepted by the agents. The set $\mathcal{A}$ depends on the individual risk preference, meaning the $\mathcal{A}$ and $\rho(X)$ is connected that can be defined such

Definition 2.8.2. *Let the return of the portfolio be denoted S_T where T denotes the final time period of trading. Then the risk measure with an associated acceptance set $\mathcal{A}$ is a mapping from $\mathcal{X}$ to $\mathbb{R}$ defined as*

$$\rho_{\mathcal{A},S_T} = \inf\{m | m \cdot S_T + X \in \mathcal{A}\}, \tag{2.8.1}$$

where m is the the minimum extra investment capital needed to enter a tolerable position of the portfolio with respect to the final outcome portfolio value.

Definition 2.8.3. *The acceptance set with respect to a risk measure is the set denoted $\mathcal{A}_\rho$ defined as*

$$\mathcal{A}_\rho = \{X \in \mathcal{X} | \rho(X) \leq 0\}. \tag{2.8.2}$$

This means that if $f(X,Y)$ real valued functions of the loss of a decision made in the space $X \in \mathcal{X}$ and $Y \in \mathcal{Y}$ be a random vector. Further $P(Y)$ is the probability density function of Y, where we have that

$$\Psi(X, g_0) = \int_{f(X,Y) \leq g_0} P(Y)dY, \tag{2.8.3}$$

is the probability of loss with g_0 is the threshold. The VaR is then denoted

$$\phi(X) = \min\{g_0 \in \mathbb{R} \ : \ \Psi(X, g_0) \geq \beta\}, \tag{2.8.4}$$

for a probability $\beta \in [0,1]$. The Value at Risk is the value where the probability is greater or equal to the probability level, usually β is 99% or 95%

2.8.1 Coherent Risk Measure

We introduce the axioms that governs a coherent risk measure, which are to satisfy these four axioms below and further relate them to a portfolio perspective.

1. **Translation Invariance**: For all $X \in \mathcal{X}$ and for all $m \in \mathbb{R}$, we have
$$P(X+m) = P(X) - m$$
2. **Sub-additivity**: For $X_1, X_2 \in \mathcal{X}$, we have
$$P(X_1 + X_2) \leq P(X_1) + P(X_2)$$
3. **Positive Homogenity**: For all $X \in \mathcal{X}$ and for all $\tau > 0$, we have
$$P(\tau \cdot X) = \tau \cdot P(X)$$
4. **Monotonicity**: For all $X_1, X_2 \in \mathcal{X}$ with $X_1 \leq X_2$, we have
$$P(X_1) \geq P(X_2)$$

Translation invariance covers the fact that by adding some risk free amount of cash investments of size m to the portfolio, does not increase or decrease any further risk to the portfolio risk at the current position. Sub-additivity allows diversification of portfolios, meaning that the composed portfolio risk will contain strictly less risk then individual assets. This also requires that the assets are somewhat uncorrelated. This property captures the fact that adding a new assets of some sort, not correlated with value 1 from previous asset in the portfolio, will increase the risk by the risk of the single asset. The third axiom explains that a certain portfolio risk is increased with the factor τ, then the portfolio risk would increase with at least the amount of τ. The last axiom of monotonicity covers the fact that a portfolio of X_2 that does perform better than a portfolio X_1, then the risk of holding X_1 compared to holding X_2 will be greater for each state of portfolio holding.

2.8.2 Law Invariance

The risk measuring methods assess its risk by the loss distribution of the portfolio from the empirical data we have. The definition for a risk measure that is law invariant is the following:

Definition 2.8.4. *Let X_1 and X_2 be random variables with a distribution function F_{X_1} and F_{X_2}, then $\rho(\cdot)$ is a law invariant risk measure if*

$$F_{X_1} = F_{X_2} \to \rho(X_1) = \rho(X_2)$$

.

Notice that if the distribution of the random variables are identical, the risk should also be identical. This implies measuring a non-identically loss distributions, where the risk measure can't be deducted through the distribution itself.

The VaR measure does not satisfy the sub-additivity property which when aggregating the risk for a portfolio, it will lead to a risk reduction as shown here.

$$VaR_\beta(L_1 + \cdots + L_d) > VaR_\beta(L_1) + \cdots + VaR_\beta(L_d)$$

2.9 Deep Learning Approximation for Stochastic Control Problems

The common way to solve stochastic control problems is through dynamic programming. The problem that the dynamic programming encounters, are due to the data being in high-dimensions or known as the "curse of dimensionality". In the recent years, machine learning have shown good results to solving such problems. Dealing with high-dimension problems in the sense of stochastic control problems, have be done by the approximating dynamic programming [Han et al., 2016]. The method replaces the true value function with an approximated function, then advancing forward in time from a sample path with backward steps to update the value function. We mention briefly the theory under stochastic control problems and stock prediction using binary indicators. The following two problems was considered during the project, but did not make any meaning full progress to be included more then a short introduction.

2.9.1 Stochastic Optimal Control Problem

We now define the stochastic optimal control problem where we first introduce the key elements required. The general setup consist of the stochastic differential equation with initial condition given as follows:

$$\begin{cases} dx(t) = f(t, s(t), u(t))dt + b(t, s(t), u(t))dW_t, \\ s(0) = s_0, \end{cases}$$

where $f(t, s, u)$ is the drift, and $b(t, s, u)$ is the diffusion. The state variable is denoted $s(t) \in \mathbb{R}^n$ and the control variable is $u(t) \in U \subset \mathbb{R}^m$. W_t is the Wiener process. We define the optimal control variable u to be

$$u(t) = \mathbf{u}(t, s(t)).$$

The control problem aims to minimise the loss or a performance $J(t, s, u)$ given

$$J(x, y; u) = \mathbb{E}_{x,y}\left[\int_x^T l(\tau, s(\tau), u(\tau))d\tau + \psi(s(T))\right].$$

The value function is defined as:

$$V(x, y) = \inf_{u \in U} \mathbb{E}_{xy}\left[\int_x^T L(\tau, s(\tau), u(\tau))d\tau + \psi(s(T)))\right] = J(x, y; u^*),$$

where the value function is the minimum cost reachable given some initial condition $s(x) = y$.

2.9.2 Problem 1

We consider a stochastic control problem with finite time horizon T on the probability space $(\Omega, \mathcal{F}, \mathbb{P})$ with the increasing filtration $\mathcal{F}_0 \subset \mathcal{F}_1 \subset \cdots \subset \mathcal{F}_T = \mathcal{F}$. We assume that any variable indexed by t is $\mathcal{F}_T$-measurable. We further denote the state variable $s(t) \in \mathcal{S}_t \subset \mathbb{R}^m$, where $\mathcal{S}_t$ is the set of states. The control variable is $u(t) \in \mathbb{R}^n$. We assume that the evolution of the system is described by the stochastic model

$$s(t-1) = s(t) + b_t(s(t), u(t)) + \xi_{t+1},$$

here b_t is the deterministic drift term given by the model. $\xi_{t+1} \in \mathbb{R}^n$ is a $\mathcal{F}_{t+1}$-measurable random variable that contains all the noisy information arriving in the period $[t, t+1)$. This can be viewed as as a discretised version of a stochastic differential equations. The problem can be formulated as

$$\min \mathbb{E}(C_T|s(0)) = \min \mathbb{E}\left(\sum_{t=0}^{T-1} c_t(s(t), u(s(t))) + c_T(s(T))|s(0)\right),$$

where $c_t(s_t, a_t)$ is the intermediate cost, $c_T(s_T)$ is the final cost and C_T is the total cost. we define the cumulative cost

$$C_t = \sum_{\tau=0}^{t} c_\tau(s_\tau, a_\tau), \quad t < T.$$

In applications in finance, where we are concerned with minimising the expected cost of trading blocks of stocks over a fixed time horizon. Let a_t denote the number of shares of each stock brought in period t, and the respective price be p_t. Then the investor's objective is then to

$$\min \mathbb{E}\sum_{t=0}^{T-1} p_t a_t, \quad \text{s.t. all } n \text{ stocks within time } T.$$

We let the price be constructed of two components

$$p_t = \tilde{p}_t + \delta_t,$$

where $\tilde{p}_t$ is the no-impact price given by the Geometric Brownian motion, and δ_t is the impact price (the price of transactions etc.).

2.9.3 Problem 2

We want to predict the prices either going up or down. Let the prediction denote by Z_t, what is our optimal allocation of assets based on the indicator Z. We then seek to maximise

$$\mathbb{E}[U(P_{t+1})|Z_t = z] = \int U(x) p_{P_{t+1}|Z_t=z}(x) dx,$$

where $p_{P_{t+1}|Z_t=z}$ is the conditional probability distribution of P_{t+1} and U is some utility function. We let $Y_t := \log(y_t)$.

$$\begin{aligned}\mathbb{E}[U(P_{t+1})|Z_t = z] &= \mathbb{E}[U(P_t\mu_t\langle y_{t+1}, \omega_t\rangle)|Z_t = z] \\ &= \mathbb{E}[U(P_t\mu_t\langle \exp(Y_{t+1}), \omega_t\rangle)|Z_t = z] \\ &= \mathbb{E}[U(P_t\mu_t\langle e^y, \omega_t\rangle)|Z_t = z] \\ &= \int U(P_t\mu_t\langle e^y, \omega_t\rangle)dP_{Y_{t+1}\ |Z_t=z}(y)dy\end{aligned}$$

Further we consider a simple model with one risk asset, and cash. The log-return can then be assumed to have a normally distributed random variable, that is,

$$Y_{t+1}|Z_t = z \sim N(\mu_z, \sigma_z)$$

for some μ_z, σ_z. It follows that

$$\mathbb{E}[U(P_{t+1}|Z_t = z)] = \int U(P_t u_t(e^r\omega_t^0 + e^y\omega_t^1))\varphi_z(y)dy,$$

where $\varphi_z(y)$ is the density of $N(\mu_z, \sigma_z)$. We optimise with respect to the weights $\omega_t^0 = 1 - \lambda$ and $\omega_t^1 = \lambda$. First consider the classical case $u = \log$. That is

$$\begin{aligned}\mathbb{E}[\log(P_{t+1})|Z_t = z] = \log(P_t) &+ \log(\mu_t) \\ &+ \int \log(e^r(1-\lambda) + e^y\lambda)\varphi_z(y)dy.\end{aligned}$$

Denoting the last term

$$\Psi(\lambda) = \int \log(e^r(1-\lambda) + e^y\lambda)\varphi_z(y)dy.$$

Optimising the last term with respect to λ, differentiating we obtain

$$\Psi'(\lambda) = \int \kappa(\lambda, y)\varphi_z(y)dy, \quad \kappa(\lambda, y) = \frac{e^{y-r} - 1}{1 + \lambda(e^{y-r} - 1)}.$$

The indicator Z does not require to be discrete, we assume that the estimated conditional expectation have a mean distribution with μ_z and σ_z. Then the continuous indicator is a function which can be optimised as $z \mapsto (\mu_z, \sigma_z)$.

Chapter 3

Machine Learning

In this chapter we discuss the most common methods in machine learning used for solving certain problem. We begin by introducing some history of the machine learning field. We present the general theory neural networks, where we describe more in detail how neurons work and the methods for learning using neural network. Further we briefly mention some machine learning methods such as convolutional network and recurrent neural network. We end this chapter by introducing the reinforcement learning method, where the goal is to learn from some experience.

3.1 Neural Network

The objective of this section is to give the readers an understanding of the deep learning algorithms used in practice. Mainly focusing on the neural networks based on the idea of human neural system, but also giving a more general view of that machine learning as statistical problem solver.

3.1.1 Machine Learning

The general framework of machine learning definition is by means of constructing algorithms that solve problems and possibly make predictions. This process is done by learning some task where key characteristics of the problem to be solved, are to be trained on. The output result should validated, and then to be tested on some test sample data to enable to say something about the result. A famous quotation goes as follows: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." [Mitchell, 1997]

General learning Task
The task of the machine learning can be of many variations, but it is mainly categorised in two groups, this is directly connect to what the desired output of the model is. The regression model is such that the learning task is to find a relationship between a set of independent variables. The dependent variables then changes based on what happens when the independent input variable changes, or a new input is given. This model is widely used w.r.t. prediction, forecasting and function approximation. The regression method that is a method can be used to predict a problem with outcome of either "yes" or "no", where output are of classes. This problem is from supervised learning, whereas an unsupervised learning method classifies observations without determining its output classes. More material can be found in [Goodfellow et al., 2016] or [Bishop, 2006].

3.1.2 Introducing Neural Network

The field of neural network is based on the idea of how the human brain works on a biological level. To understand better what neural nets are, we will give a brief overview of how neural networks is inspired by the biological brain structure of neurons. We will be mainly focusing on the process of receiving an input signal from nerve or other sources, then reaching a neuron. [Bishop, 2006] mentions some application where the neural network models are more relevant where problems such as: pattern recognition, function approximation and control problems. These problems are of mapping the input to a specified output by adjusting network weights, hence the leaning methodology is a supervised learning task.

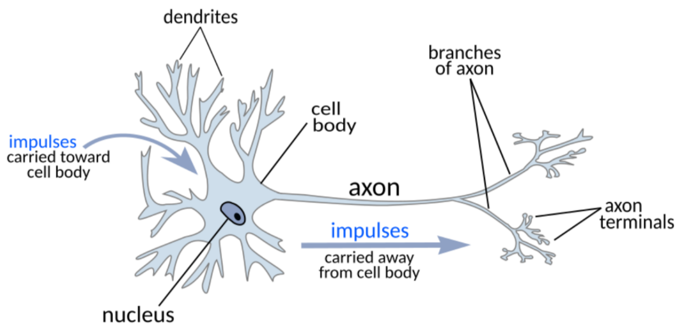
dendrites
branches of axon
cell body
impulses carried toward cell body
axon
axon terminals
impulses carried away from cell body
nucleus

Figure 3.1: Illustration of a neuron with the body and axons

The structure of a neuron which we are mainly interested in is the cell body and it's branches, the dendrites and axons. The dendrites are the input surface of the neuron, signals are transferred through the cell body to the axon terminals. The terminal axons that interacts with other neurons are called a synapse. The end terminal axons connects to a vast amount of neurons, which further connects to more neurons, making it a neural system. Which neurons connected to which neurons or synapse decides what the end output will be. The neuron has a certain threshold for a synapse to be activated, this sequentially triggers the next neuron to be activated. The threshold is either a on or off alternative, defined by the difference of the membrane of the cell neuron electrical charge or chemical structure. The difference of the inside part of membrane and the outside part of membrane i.e. the adjacent cells.

In [McCulloch and Pitts, 1943] gave a simple mathematical defined computational model of neurons with a binary classification. The inputs which where either excitatory or inhibitory, the on or off principle. Their model did have limitations where it was not able to do any learning. It was not until 1958 the paper [Rosenblatt, 1958] introduced a solution using perceptrons.

The perceptron model inherited some of it's key ideas of how signal transmissions of neurons happens. Using the on or off property of neurons, they were able to formulate weights to model this. The synapse connected to output and input of neurons has some weights w_i on the i'th input attached to it. For all inputs, the weights are multiplied and summed, so if we have m inputs, we then get $a = \sum_{i=1}^{m} x_i w_i + b$. The b term is bias, often uses as a offset value set to 1. The returning y^* value is compared with the threshold θ to decide the output of y^*.

$$y^* = \begin{cases} -1 & \text{if } a \leq \theta \\ 1 & \text{if } a > \theta \end{cases}$$

The output here used in classification of whether the model gives values of $+1$ or -1. Further the output value is compared with the "correct" value y. The learning procedure requires that the threshold and weights are changed according to the classification. The learning rule employed here can be written formally as

$$\begin{aligned} w_i(t+1) &= w_i(t) + \Delta w_i(t) \\ \theta(t+1) &= \theta(t) + \Delta\theta(t), \end{aligned}$$

where t is some time point under training. If $y^* \neq y$ then the weights will be changed in the direction of $\Delta w_i = y x_i$, and we start with the summation again with the new weights until the correct classification occurs. The threshold is modified after the following rule

$$\Delta\theta = \begin{cases} 0 & \text{classification is correct} \\ y^* & \text{otherwise} \end{cases}$$

The learning process of a human, for example with a young child involves a certain input of a object such as a dog or a cat. Assuming that the child as never seen a dog before, the child would learn that the object in front is a dog, by his parents. The parents then act as a teacher which then correct the child if he/she would assume that the dog is a cat. As the child is more exposed to dogs and cats through multiple encounters in daily life, the child will learn the features of a dog or a cat. In the sense of neural nets, this can be simulated by adjusting the weights through training the model and to be able to classify different objects. [Deng et al., 2009]

In the early stages when the perceptron model was introduced, the model it self didn't get much attention since [Minsky and Papert, 2017] showed in in 1960 that the model had limitations of not being able to solve nonlinear function. The discovery lead to a natural breaking point of interest in neural networks of almost three generations. The work of [Rumelhart et al., 1986] introduced the back-propagating, a technique to back trace errors made in a multiplayer perceptron model. The method will be described further in this thesis. Their paper brought a new spark in the interest of neural networks.

3.1.3 Network Architectures

Setting up a network of neurons and layers is essential to make a usable model for problem solving. The network structure in defined as the network architecture, where we denote the number of layers L, which can be categorised to three different types of layers. The first is the input layer, hidden layers and output layer. Define the within layer l, the design must decide how many nodes, M_l each layer should be made of. The general rule is to not chose a complex model, when a simple model are sufficient for the problem.

Input Layer

The number of input layers, denoted the vector $\boldsymbol{x} \in \mathbb{R}^{M_1 \times 1}$, is the general case for simple input vector. This corresponds to one neuron, where for simple problems, the number of input nodes is defined. As for simple problem, the determination of number of inputs are fairly easy compared to network solving financial time series. The specifications and the dependencies of the output result are often unknown which leads to a more guessing and trying exercise. The choices are plenty, spending time on the right set of inputs are often needed to do well in the model [Walczak, 2001]. In general we must consider the fact that adding a large amount of input data does not necessarily mean good. The problem arises when the data are to complex, which requires more computational power, also adding tendency to overfit the data. Further following [Walczak, 2001], the choice of inputs sets can be based on the beliefs on the input being sufficient to explain the data structure. This involved consulting the domain of input where noisy and not relevant data is being removed. This can be done by correlation and dependence investigation. A simple model or complex model should be ideally uncorrelated with each other. A set of highly correlated data leads to overlapping information, which can make the network predictions bad.

Hidden layers

Further the number of hidden layers must be specified. The number of hidden layers allows detection of data features, where nonlinear mapping between input and output are performed. [Hagan et al., 1996] mentions that a large function with a large number of points requires also requires a large number of neurons in the hidden layers. This meets the problem of having a smaller model compared to a larger model where increased learning time are required.

Output Layer

Number of output layer are connected to the problem to be solved, and are relatively easy to determine. The output number are often decided if the problem is of classification or a direct consequence of the number of classes or labels the selected input should be divided into. For regression problems, the number of outputs can be more determined by the problem which one want to solve. The model architecture is crucial for all modelling problems. The number of training examples limits the size that should be used, as bigger models adds more parameters to be estimated, and thus making it harder. Some suggesting for [Zhang et al., 1998] that each parameter of interest should at least have 10 training data sample, meaning that more weights too be estimated, the bigger the network it requires.

3.1.4 Data Preprocessing

Preprocessing the input and output data are often necessarily for most real world application where often data comes in a raw format. The structure of such raw data are often unstructured, which is quite useless. Reworking data are often done by removing noise, highlighting relationships and flattening distributions and detect the trends to help the network. As we mentioned in the first chapter, we did remove days which had no occurred trading. Determining the right set of inputs which data can be used in preprocessing techniques such as the PCA dimension reduction method [Smith, 2002].

3.2 Optimisation of Neural Network Parameters

The optimisation of the neural network parameter consists of two parts, which are the optimisation of the network weights given some a set of hyperparameters. This is known as the training process of the network. The weights vector $\boldsymbol{w}^*$, that minimises the error function for a given training sample. The second part is optimisation of the hyperparameters, which determines the network structure for best modelling result.

3.2.1 Training, Validation and Testing

The training process of a neural network model requires that we process the data samples in batches for first the training, validation and a testing process. The training sample are used to optimise the network weights. The validation data are used to adjust the network weights for any overfitting. This means by verifying that any increase in accuracy over the training data set actually yields an increase in accuracy over a data set that has not been seen by the model before. Should the training accuracy data set increase, but the accuracy over the validation set stay on the same level or decreases, then there are overfitting in the network. The final test set is made such that there are no overlapping data set where the network can be evaluated on.

3.3 Feedforward Network

Modern models neural nets such as a feedforward network are based on the perceptron model. How the model is build is described using its architecture features such as the depth, width and the number of hidden units involved. Deepness of the model corresponds to the network of functions that are connected in chain. Let classifier be denoted $y = f(x)$ with input x and activation function $f(\cdot)$. The function can be composed with a number of functions where each function acts as a new layer to give the model depth. This can be written as $f(\mathbf{x_t}) = f^{(n)}(f^{(n-1)}(\cdots f^{(2)}(f^{(1)}(\mathbf{x_t}))))$ where n is the depth of the model. The first layer, here $z_1 = f^{(1)}(\mathbf{x_t})$ takes directly on the input vector of $\mathbf{x_t}$, such as the market vector. Layer $z_n = f^{(n)}$ is the output layer of the model, when the training is finished, the neural network hands over the predicted value of the true value of label y, in this case denoted y^*. Layers between the first and the n'th layer are the hidden layers or units. These layers are vector valued with a width, which also determines the network width.

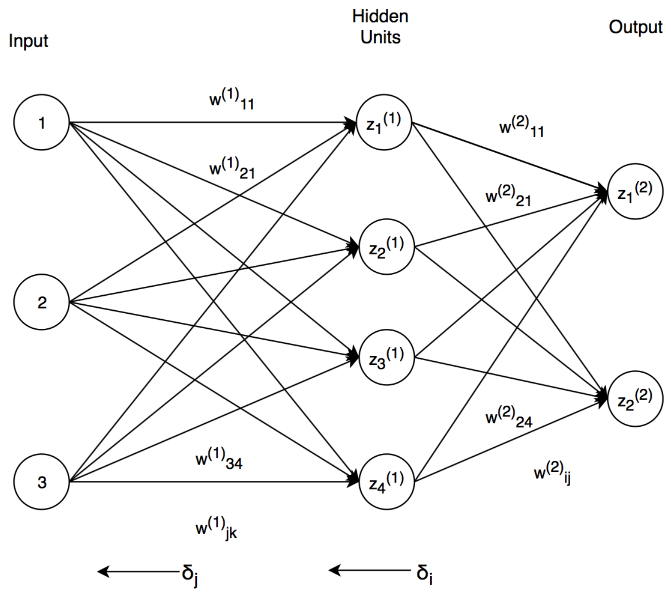

Figure 3.2: Perceptron network.

Further a simplified model will be used to illustrate the feedforward network, then generalised to a deep feedforward model in done by adding more hidden layers. We have that, when a neural input is $\mathbf{x_t} = \{x_t^{(1)}, \ldots, x_t^{(m)}\}$ the first and second layer from the model above is

$$z_j^1 = f^1\left(\sum_{k=1}^{m} w_{jk}^{(1)}\mathbf{x_k} + b_j\right)$$
$$z_i^2 = f^2\left(\sum_{j=1}^{n} w_{ij}^{(2)} z_j^1\right),$$

where m and n are the numbers of input units and number of hidden units respectively. We denote the number of layer l, our model have two layers. The weights $w_{ij}^{(l)}$ are subscribed with the layer l, and going from a unit j to unit i. For a single layer output we have that

$$z^l = f(\mathbf{W}^l \cdot z^{l-1} + b^l),$$

where the $\mathbf{W}^l$ is the weight matrix with size $n \times m$. By using the mean squared error as a measure of loss we have

$$J(\mathbf{x}) = \frac{1}{m}\sum_{i=1}^{m}\left[y(\mathbf{x_i}) - \mathbf{z}(\mathbf{x_i})\right]^2,$$

where $\mathbf{z}(\mathbf{x_i})$ is the output of the prediction from the last layer. This is the more commonly used mean squared error function, that has certain pros and cons depending on the training data set available.

3.3.1 Loss Function and Optimisation

The loss function topic regards the choosing of a suitable function that can be optimised under network training. In general the weights are updated iteratively as we described in the introducing section of neural networks. The loss function measures the distance from predicted value of y^* from the true value y. We introduce maximum likelihood to train networks. This means that the cost function is the negative log-likelihood function. The cost or loss function $J(\boldsymbol{\theta})$ is then given as in [Goodfellow et al., 2016].

Through iterative computation, the learning are done by maximising or minimising the loss function with the input vector $\mathbf{x}$. We assume first that the data has some probability distribution denoted $p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})$.

$$\begin{aligned} J(\boldsymbol{\theta}) = -\mathbb{E}_{\mathbf{x},y}\left[\mathrm{L}(\mathbf{x}, y, \boldsymbol{\theta})\right] &= \frac{1}{m}\sum_{j=1}^{m} \mathrm{L}(\mathbf{x}^{(\mathbf{j})}, y^{(j)}, \boldsymbol{\theta}) \\ &= -\frac{1}{m}\sum_{j=1}^{m} \log\, p(y^{(i)}|\mathbf{x};\boldsymbol{\theta}), \end{aligned}$$

where m is the number of training set. Using the gradient method based on finding critical points with respect to θ we have that the gradient of the loss function is

$$\nabla_{\theta} J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{j=1}^{m} \nabla_{\boldsymbol{\theta}} \mathrm{L}(\mathbf{x}^{(\mathbf{j})}, y^{(j)}, \boldsymbol{\theta}),$$

where $\nabla_{\boldsymbol{\theta}}$ is the gradient with respect to $\boldsymbol{\theta}$.

We will use aa extension of the gradient decent method named stochastic gradient decent where we will be taking smaller batches of the sample size m, denoted m^s. The learning then becomes

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \epsilon g,$$

where ϵ is the learning rate and g is the new loss function with a smaller batch sample formalised as

$$g = \frac{1}{m^s} \sum_{j=1}^{m^s} \nabla_{\boldsymbol{\theta}} \mathrm{L}(\mathbf{x}^{(j)}, y^{(j)}, \boldsymbol{\theta}).$$

The learning rate $\epsilon \in [0, 1]$ controls the momentum, indicating the effect of the previous update should be included in the current update of θ. Batch updates have advantages when the sampling space is large and where taking the gradient of the m inputs can be time consuming.

Often a one does not need to take the distribution of $p(\mathbf{y}|\mathbf{x}; \boldsymbol{\theta})$, but instead take the some statistic of $\mathbf{y}$ conditioned on $\mathbf{x}$.

The log-likelihood loss function are usually used along with the softmax function as it outputs are in the form of a probability distribution. The softmax transforms logits values into the probability space $(0, 1)$. The cross-entropy then finds the error of the prediction output after transforming the logits. We have that the softmax function is given as follows:

$$\sigma(z_j) = \frac{e^{z_j}}{\sum_{k=1}^{m} e^{z_k}},$$

normalised over the K number of inputs. Together with the cross-entropy, we we have a loss function

$$L = -\sum_j y_j \ \log\sigma(z_j),$$

further averaging over number of classes we get

$$J = -\frac{1}{m} \sum_{j=1}^{m} (y_j \cdot \log\sigma(z_j)).$$

As the cross-entropy loss function as been revealed, we will need to consider to derive the gradients of the loss function. This is done in the next section with backpropagating.

3.3.2 Backpropagation

In order to derive the weights through back propagating, we need to be able to calculate the gradients of the loss function with softmax cross-entropy. The key is to use chain-derivatives as our main tool. We first begin with the softmax function defined in previous section, then move on to the loss function. We have that

$$\begin{cases} \frac{\partial \sigma(z_j)}{\partial z_j} = \sigma(z_j)(1-\sigma(z_j)) & \text{if } j = k \\ \frac{\partial \sigma(z_j)}{\partial z_k} = -\sigma(z_j)\sigma(z_k) & \text{if } j \neq k \end{cases}$$

With the cross-entropy loss we have

$$\frac{\partial J}{\partial z_j} = \sigma(z_j) - y_j$$

The calculation can be looked up in the appendix. The backpropagation method was first introduced in 1986 by [Rumelhart et al., 1986], making it possible for neural networks to solve problems related to learning task and classification problems that was previously was considered complicated. The training of deep learning models was also meet with obstacles such as the vanishing gradient problem. For more detailed explanation see [Hochreiter, 1998].

3.3.3 Activation Function

The choice of activation function used in the hidden units can be a mysterious and often found by trying different functions. We already mentioned one type of activation function, the softmax, and their usage. Another common activation functions are the rectified linear function (ReLU) defined as

$$f(z) = \max\{0, z\}.$$

This function is easy to optimise due to being closely related to linear units. The derivates remains large over the active states since the output of this particular function is zero across half its domain. In general the ReLU function is easy to optimise when the models is related to a linear model [Goodfellow et al., 2016]. The ReLU function is not differentiable at $x = 0$, which might be an issue for the gradient learning. This might not seem to work here in this case, but this method does work in practise due to the fact that derivates are close to zero, still is defined.

The sigmoid function is

$$\sigma(z) = \frac{1}{1 + \exp(-z)}.$$

This function suffers from saturation of high values if the input $\mathbf{x}$ have high values, and the opposite for high negative values of $\mathbf{x}$. Sigmoid works best as the values of input is close to zero. The sigmoid function works best when used as the output function mapping values to a probability space.

Since the introduction of the ReLU function, the use sigmoid function has been left behind in the feedforward network. Another activation function that we will be including is the exponential linear units (ELU). This activation function is given as:

$$\sigma(z, \alpha) = \begin{cases} \alpha(e^x - 1) & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases}$$

3.3.4 Regularisation Methods

Training large neural networks have been known to be overfitting. We will give short introduction to the L_2 norm regularisation and its ideas. Further bringing in dropout methods that covers high level data that we will be using.

The L_2 regularisation aims to reduce unimportant weights to zero by adding a scaling term with the loss function that does the weight decaying operation. The loss function is then:

$$J\hat{(\theta)} = J(\theta) + \lambda_{\text{reg}} \sum_{w} (w_t^{(j)})^2$$

where λ_{reg} is the strength controller of the regularisation.

The common regularisation method now is the dropout method [Hinton et al., 2012] assess a upper bound on the L_2 norm regularisation for each hidden units in the network. This method work alongs the mini batch sampling, where randomly chosen units are being removed under the training process with some given probability. As the weights under the hidden layers overcome the upper bound threshold, the weights are then renormalised.

The dropout methods is used in a way of reducing the errors in the training data set. The training is dividend into mini batches. Training on each of these mini batches implies running a separate network on each batch, which then are averaged. For a large sample size training data, this could take time to compute when not using dropout, but the method allows for an optimised method without adding any covariates and excess time during training. In practice, the dropout used in conjunction with the L_2 norm regularisation, in which these are both independent ways of reducing the training risk error.

3.4 Recurrent Neural Network

Stock market trends are often somewhat correlated to the previous information of stock movements. The recurrent network are a network architecture that takes into the account for such series in the financial market. This means also that an RNN also can do general sequential datas, such as audio, natural language processing or sentiment analysis. We will cover some fundamental ideas behind the RNN framework and discuss some properties with our problem task in mind. Then we will see that the basic RNN also struggles with the vanishing or exploding gradient problem, which are solved by using a Long Short Term Memory network.

The simples RNN takes the one neuron as the input, producing an output, where the networks sends the input signal backwards to the first or input layer. Denoting the input x_t at time step t, the network also takins the output from the previous iteration step, denoted y_{t-1}. The parameters are then then x_t and y_{t-1}, which are both vectors. Denoting the weights w_x and w_y the output of a single recurrent neuron can be computed as some function $\phi(\cdot)$ where the output is

$$y_t = \phi\left(x_t \cdot w_x + y_{t-1} \cdot w_y + b\right).$$

This seems similar to the feed forward network from our introduction, which is correct. Further a import concept is the memory cell and forget cell. Such cells contains the data from previously states, and the network are stacked with such layers of cells. The cells are also some sense hidden in the network as the feedforward hidden layers. The input of the cell at time step t depends on the previous state $t-1$, so denoting a cell $h(\cdot)$, we have that $h_t = f(h_{t-1}, x_t)$. For a more complex network, in a sense where we use more past data steps, the network output will be different from our debut network structure.

In connection with our problem, we will take to predict the $t+1$ step ahead price of the stock, where we input the information at time step t. We will also look at the values the network suggests from the beginning of the trading, so at $t=0$ up to $t=t$, to access some possible information of the prediction accuracy and even possibly deduct some sort of risk of the predictions.

When considering the training, the process follows much of the same ideas as the feed forward network. The forward pass from each cell, and then the gradients of the loss function are computed using backpropagation. Assuming a RNN network with some loss function $L(y_t, y_{t-1}, y_{t-2})$, meaning the loss function only depends on the the last known steps of the stock prices. Then backpropagation computes it's following three gradients and ignores the rest. The cell layers allow for sharing parameter under training.

3.4.1 LSTM cell

The LSTM cells as introduced above have some common factors with the basic cells in RNN networks. The popularity rises due to it's performance during training, in which the converges happens faster and detects long term dependencies in the data. The LSTM structure takes two states, the input of a short term state h_t and a long term state c_t. The network learnings what should be remembered as a long term property or a short term property. The long term states c_{t-1} traverse the network from left to right, firstly the data goes through a forget gate, dropping some memories, and adds some new memories via the addition operation. c_t

sends straight out, without transformation. At each steps, some memories are dropped, and some memories are added to. Results are then passed through a tanh function, and filtering by some output gate. The short term state h_t, remembers information for a short period of time. See [Hochreiter and Schmidhuber, 1997] for more information on LSTM model.

3.5 Convolutional Network

The name "convolutional" has it's roots in the convolutional operations in the network. In a general 2-dimensional discrete finite space a convolutional can be defined as

$$(I * K)(x, y) = \sum_{m=-M}^{M} \sum_{n=-N}^{N} I(x+m, y+n)K(m, n),$$

where I is a input image with black and white colour represented with an array of size $n_1 \times n_2$, and the K is the convolutional kernel function with size $(2M+1) \times (2N+1)$. The kernel can be then given as the matrix:

$$K = \begin{pmatrix} K(-M,-N) & \dots & K(-M,N) \\ \vdots & K(0,0) & \vdots \\ K(M,N) & \dots & K(M,N) \end{pmatrix}$$

The discrete case, the kernel takes the multiplication of a smaller window sizes of the image, trying to extract features such as edges of image objective. Further expanding to a multicoloured picture, the image then takes a new layer of depth describing colours where the image array tuns into $(width) \times (height) \times (depth)$. We can notice that using a feedforward network would easily lead to a high number of weights, which also increases the computational difficulty. Introducing a convolutional network will allows us to perform calculations without compensation result with computing time. This method is often used in imagine recognition or patter recognition [Krizhevsky et al., 2012].

3.6 Reinforcement Learning

The reinforcement learning (RL) framework takes the idea of learning from some experience. As humans, we often encounter the need to complete a certain task. Assuming these task are done by some action that maps to a reward for this task. The learning does not know in the beginning what the outcome of such task is. By taking one action, we look at the reward. Was is well done or did it go horribly. The second action is then perhaps something better then the first try, which leads to a better out come. After completing multiple trails, the human has learned that doing certain actions is more efficient then other actions. The learners has then found some optimal method of solving such task. This section we look at how this can be done in a computational way where learning is done through interaction with the environment. We use the same notations in the book of [Sutton and Barto, 2018].

3.6.1 Return and Reward

The RL agents learns to maximise the cumulative future reward of some task. We denote the return $R_t, t \geq 0$, where the return depends on the time t. The most simple cumulative return is

$$\begin{aligned} G_t &= R_{t+1} + R_{t+1} + \cdots = \sum_{i=1}^{\infty} R_{t+i+1} \\ &= R_{t+1} + G_{t+1}, \end{aligned}$$

this return does give equal weights to the present return and the future return. We introduce a shrinkage constant $\gamma \in [0, 1]$, where we give less weighs to the returns future in time then the present

$$\begin{aligned} G_t &= \gamma^0 R_{t+1} + \gamma^1 R_{t+1} + \gamma^2 R_{t+1} + \cdots = \sum_{i=1}^{\infty} \gamma^i R_{t+i+1} \\ &= R_{t+1} + \gamma G_{t+1}. \end{aligned}$$

We can now consider reward not only based on the instantaneous feedback, by also reward for next situation, and through that, all subsequent rewards.

3.6.2 Value Functions

Learning the optimal policy, requires the use of value functions. Mainly the value functions are of the state value function or action value function. The primer describes the value of a state when following a policy. We have the following equation for state value function $V^{\pi}(s)$

$$V^{\pi}(s) = \mathbb{E}_{\pi}[R_t|S_t = s],$$

where the return R_t is depends on the state s in time t. The action value function denoted $Q^{\pi}(s, a)$, and can be formalised as

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}[R_t|S_t = s, A_t = a],$$

where the return depends on the state s and action a at time t. Expectation are taken due to the randomness in future returns of both value functions.

3.6.3 Bellman Equation

We first define the following probability

$$p(s', r \ |s, a) = Pr[s_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a],$$

where $s', s \in \mathcal{S}, r \in \mathcal{R}$, and $a \in \mathcal{A}(s)$. $\mathcal{S}, \mathcal{R}$ and $\mathcal{A}(s)$ is the state space, return space and action space respectively. Note that p specifies a probability distribution for each choice of s, a, in a way that

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a) = 1.$$

In a Markov decision framework, the probabilities p completely characterise the environment's dynamics, meaning each possible value for S_t and R_t depends only on the previously states S_{t-1} and A_{t-1}. Further the state transition probabilities from state s to s' given the action a

$$p(s'|s, a) = Pr[S_t = s'|S_{t-1} = s, A_{t-1} = a] = \sum_{r \in \mathcal{R}} p(s', r|s, a).$$

The expected reward for the same specification is

$$r(s,a) = \mathbb{E}[R_t | S_{t-1} = s, A_{t-1} = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r | s, a).$$

A policy denoted $\pi(a|s)$ describes the agents actions. The policy function maps states to probabilities of selecting each possible action. This means that at time t, the agent follows the policy $\pi(a|s)$ is the probability that $A_t = a$ if $S_t = s$.

We can now derive the Bellman equation for the state value action

$$V_\pi(s) = \mathbb{E}_\pi[\sum_{i=0}^{\infty} \gamma^i R_{t+i+1} | S_t = s].$$

The expectation $\mathbb{E}_\pi$ denotes the value of a random variable under the agent policy π. Further we have that

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_\pi[R_t | S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r | s, a)[r + \gamma \mathbb{E}_\pi[R_{t+1} | S_{t+1} = s']] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r | s, a)[r + \gamma V_\pi(s')]. \end{aligned}$$

The last equation is the Bellman equation for state value equation.

3.6.4 Optimal Policies and Optimal Value Functions

The goal for reinforcement agent is to find some optimal policy that can achieve good result over a long run. In the finite MDPs, we can define the optimal policy π_* is the policy $\pi_*(s) \geq \pi(s)$ for all $s \in \mathcal{S}$. The optimal policy is always at least better or equally good as other policies. Then the problem is to find the optimal state value function

$$V_*(s) = \max_\pi V_\pi(s), \quad \text{for all } s \in \mathcal{S}.$$

The optimal Bellman equation can be derived as follow

$$\begin{aligned} V_*(s) &= \max_a Q_{\pi*}(s,a) \\ &= \max_a \mathbb{E}_{\pi*}[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a] \\ &= \max_a \mathbb{E}_{\pi*}[R_{t+1} + \gamma V_*(S_{t+1}) | S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r|s,a)[r + \gamma V_*(s')]. \end{aligned}$$

For the optimal action-state value function

$$Q_*(s,a) = \max_\pi Q_\pi(s,a).$$

With the optimal state value function, we have the following

$$\begin{aligned} Q_*(s,a) &= \mathbb{E}[R_{t+1} + \gamma V_*[S_{t+1} | S_t = s, A_t = a] \\ &= \sum_{s',r} p(s', r|s,a)[r + \gamma \max_{a'} Q_*(s', a')]. \end{aligned}$$

When in a MDP framework, the optimal Bellman equation for V_* have a unique solution. The system of equations, one for each state, with n states we have n equations with n unknowns. If the dynamics p of the environment are known, then we can in principle solve this system of equations for V_*. The optimal policy relies on finding V_*. For each state s, there will be one or more actions at which the maximum is obtained in the optimal Bellman equation. After one step, the optimal equation, is the best actions that results in best value function V_*.

3.6.5 Dynamic Programming

We begin this section by noting that the optimal state value function V_π for an arbitrary policy π can be evaluated using the Dynamic programming principle. Then the prediction of V_π for $s \in \mathcal{S}$

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_\pi[G_t | S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma G_{t+1} | S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma V_\pi(S_{t+1}) | S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s,a)[r + \gamma V_\pi(s')], \end{aligned}$$

where $\pi(a|s)$ is the usual policy probability of taking action a in state s. The existence of V_π are guaranteed as long as $\gamma < 1$. We now assume that the sequence of approximate value functions are $\{V_t\}_{t=0}^{\infty}$ maps from $\mathcal{S}^+$(real positive numbers) to $\mathbb{R}$. Letting V_0 be arbitrary chosen. Then the update rule for optimal Bellman equation is

$$\begin{aligned} V_{k+1}(s) &= \mathbb{E}_\pi[R_{t+1} + \gamma V_k(S_{t+1})|S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma V_k(s')], \end{aligned}$$

where all the updates are based on the expected future returns over a finite time horizon defined by the MDP framework.

We now seek to find a policy decision π that generates $V_\pi(s)$ for some state s. Will this current policy we follow through be better then a new policy we can find? This can be answered by using the optimal action function. We now have that

$$\begin{aligned} Q_\pi(s, a) &= [R_{t+1} + \gamma V_\pi(S_{t+1})|S_t = s, A_t = a] \\ &= \sum_{s',r} p(s', r\ |s, a)[r + \gamma V_\pi(s')]. \end{aligned}$$

Comparing a only state based return with respect to some policy denoted $V_\pi(s)$ or using a deterministic action a chosen in state s, then the policy is decided $Q_\pi(s, a)$. The update is done if we have the following

$$Q_\pi(s, \pi'(s)) \geq V_\pi(s),$$

for a change policy π' compared to π. Further considering the changes made to all states and to all possible actions taking. Selecting at each state the action that appears best according to $Q_\pi(s, a)$, and is also known as the greedy policy decision, we have

$$\begin{aligned} \pi'(s) &= \arg\max_a Q_\pi(s, a) \\ &= \arg\max_a \mathbb{E}[R_{t+1} + \gamma V_\pi(S_{t+1})|S_t = s, A_t = a] \\ &= \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V_\pi(s')]. \end{aligned}$$

With the greedy policy, it can be shown that the best policy is also the optimal Bellman equation. Assume that the new policy π' is as good as, but note better then the old policy π. This yields $V_{\pi'} = V_\pi$, where

$$\begin{aligned} V_{\pi'}(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma V_{\pi'}(S_{t+1}) | S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V_{\pi'}(s')]. \end{aligned}$$

The last equation is the optimal Bellman equation, meaning that $V_{\pi'} = V_*$, and π, π' are both optimal policies. The MDP for a finite horizon leads to that there must exits a optimal policy as convergence is guaranteed.

3.6.6 Value Function Approximation

Selecting policies to be evaluated are either a on-policy or a off-policy method. The former method attempts to evaluate or improve the policy that is used to make decisions. The latter method evaluate or improves a policy different from that used to generate the data. We introduce the method of function approximation in RL by considering its use in estimating the state-value function from on-policy data. We try to approximate V_π from experience generated using a known policy π. We denote the parameterised functional from with weights $\mathbf{w} \in \mathbb{R}^d$ and write $\hat{V}(s, \mathbf{w}) \approx V_\pi(s)$ is the approximate value for state s, with $\mathbf{W}$ as the feature weights. It is natural to interpret each update as specifying an example of the desired input-output behaviour of the value function. This means that the update, $s \mapsto u$ where the estimated value function for state s should be more like the update target u. Permitting arbitrarily complex and sophisticated methods to implement and update s, generalising so that the estimated values of many other states are changed as well.

We now turn to the the predictive objective. In general, an change in the state space affect other states. This is troublesome due to not being able to find the exact value of all states in the space. When working in this framework, we often have more states then weights. We can specify a state distribution to measure how much we care about certain states. We denote the the state distribution $\mu(s)$, which must satisfy $\sum_{s\in\mathcal{S}}\mu(s)=1$. This then indicate how much we care about a the error in each state s. Now the error can be computed with the common mean square of the difference between the approximated value $\hat{v}(s,\mathbf{w})$ and the true value $v_\pi(s)$. Weighting this over the state space by μ we the the following error measure

$$\overline{\mathrm{VE}}(\mathbf{w}) = \sum_{s\in\mathcal{S}} \mu(s)\left[v_\pi(s) - \hat{v}(s,\mathbf{w})\right]^2,$$

is the Mean Squared Value Error. Depending on the task of optimisation, the error function can be different [Sutton and Barto, 2018].

3.6.7 Stochastic Gradient Decent

The most common method for solving function approximation problems by value prediction is the Stochastic Gradient Decent (SGD) method. Let the weight vector be $\mathbf{w}\in\mathbb{R}^d$ with the approximate value function $\hat{V}(s,\mathbf{w})$ is a differentiable function of $\mathbf{w}$ for all the states $s\in\mathcal{S}$. The update are done for each time steps t, where $\mathbf{w}_t$ is the weight vector for each time step. The observed value is $S_t \mapsto V_\pi(S_t)$, for some state S_t and the true value under the policy π. The SGD method then updates the weights $\mathbf{w}_t$ by the rule given as follow

$$\begin{aligned}\mathbf{w}_{t+1} &= \mathbf{w}_t - \frac{1}{2}\alpha\nabla\left[V_\pi(S_t) - \hat{V}(S_t,\mathbf{w}_t)\right]^2\\ &= \mathbf{w}_t - \alpha\nabla\left[V_\pi(S_t) - \hat{V}(S_t,\mathbf{w}_t)\right]\nabla\hat{V}(S_t,\mathbf{w}_t),\end{aligned}$$

where α is the step size parameter, and ∇ is the gradient of $\mathbf{w}$. In general the true value of $V_\pi(S_t)$ is not known, and there it should be approximated by some value U_t, then the last equation above yields

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha\nabla\left[U_t - \hat{V}(S_t,\mathbf{w}_t)\right]\nabla\hat{V}(S_t,\mathbf{w}_t),$$

where U_t is unbiased estimate, that is $\mathbb{E}[U_t|S_t=s] = V_\pi(S_t)$. When using a bootstrap estimate of $V_\pi(S_t)$ as target of U_t, the convergence is not guaranteed. Methods such as the return function used in DP, where the return are dependent on successive estimates. These dependencies affect the weight vector $\mathbf{w}_t$, which leads to biased weights. The gradient method works better when the weights are independent of the target prediction. The bootstrapping estimates gradient only effect the change of weights $\mathbf{w}_t$ on the estimate, but not on the target itself, these are then called semi-gradient methods.

These methods does have advantage that the learning can be continual and online with no need to wait for the end of an episode. For further reading concerning reinforcement learning, the book [Sutton and Barto, 2018] explains theory and includes examples.

Chapter 4

Methodology

This chapter describes methods used to for our implementation based on the previous chapters. We also provide plots in connection with simulations we do. We begin by looking at the market data we have. Then we move with the model for rough stochastic volatility, introduced in section 2.7 for the market data and simulated stock prices. The next section then shows the volatility prediction. Using the predicted volatilities, we design a stock price model based on the Geometric Brownian motion with rough stochastic volatility. In the last two sections, we find the optimal portfolio based on simulated stock prices, and finish the chapter by looking at the Geometric Brownian motion with rough volatility.

4.1 Stock Market Data

As mentioned in the introduction, we will be using real stock market data, and also stock prices driven by a Geometric Brownian motion, see subsection 2.5.1, and including volatility driven by a fractional noise, see subsection 2.5.4. From the stock market data, we will be considering the Oslo Børs Index. This dataset consists of open, close, high, low and 10 minutes realised variance for daily trades from January, 2000 up to March, 2019. The currency exchange NOK to EUR is from January 2018 up to December 2018, intraday with 10 minutes window. The stock data consist 9 months period, intraday 10 minutes sampling from the end of April 2018 up to the middle of November 2018.

We make sure that the dataset does not have empty data or values of "not a number" (NaN). We also remove holidays and weekends where the stock market is closed. For the currency data, we have trading all day long from Monday through Friday. For the index and stock, we only consider data when the market is open. This leads to jumps in close and open between two days. In general these market data exhibit these problems. By looking at the trading volume for intraday data, the periods leading up to the closing hour, show a stronger volatility than during the daytime. This is due to the high volume trades in the last hour of market closing.

4.1.1 Oslo Børs Index Data

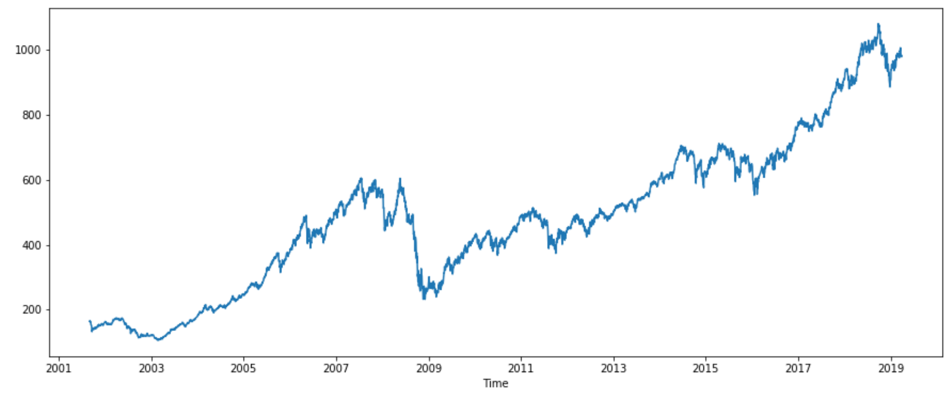

Figure 4.1: Oslo Børs index daily closing price.

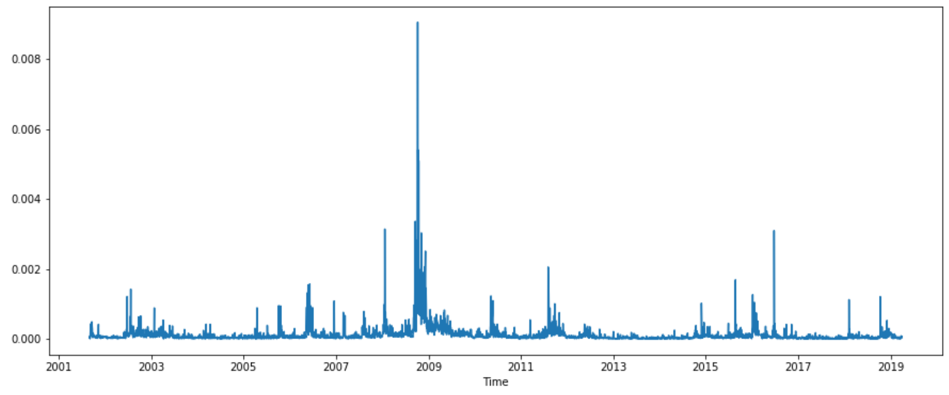

Figure 4.2: Oslo Børs index daily realised variance.

The large variance peeks are due to the financial crisis in 2008.

4.1.2 EUR to NOK Currency

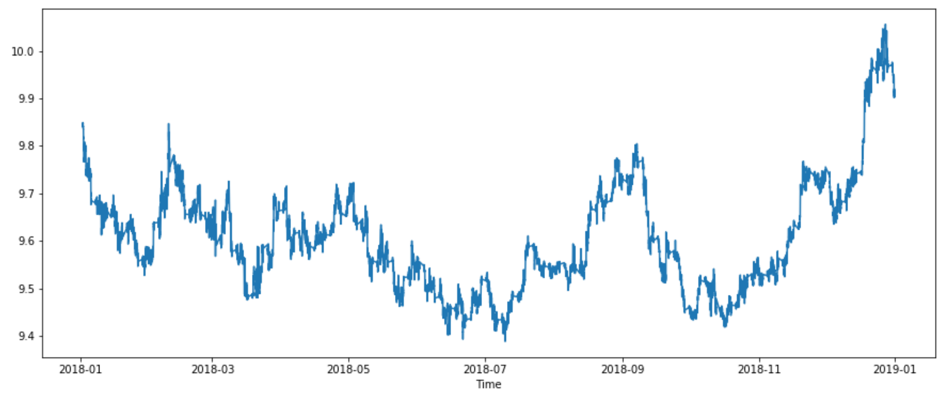

Figure 4.3: EUR to NOK currency 10 minutes closing price.

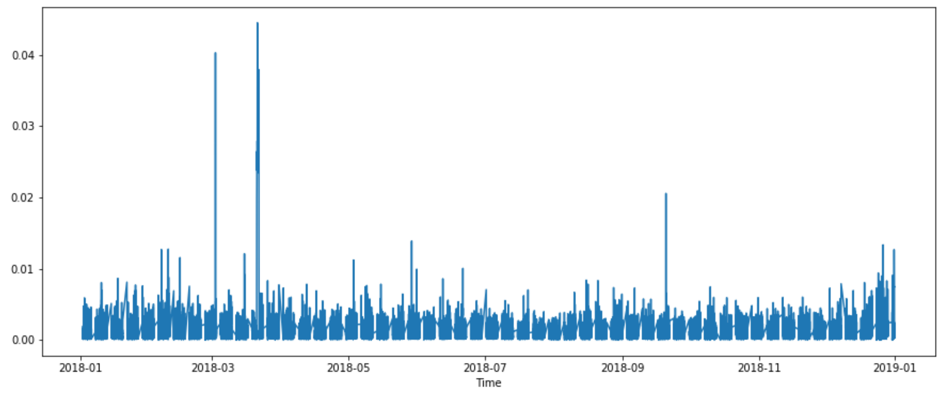

Figure 4.4: EUR to NOK currency 10 minutes standard deviation.

Here the large peaks in the volatility is due to high amounts of trading at the end of the weeks where the market is about to close.

4.1.3 Norwegian Stocks

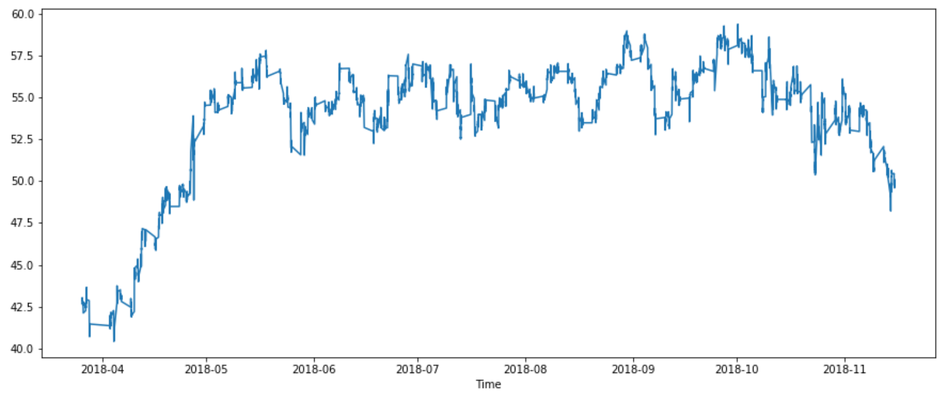

Figure 4.5: AKSONO 10 minutes closing price.

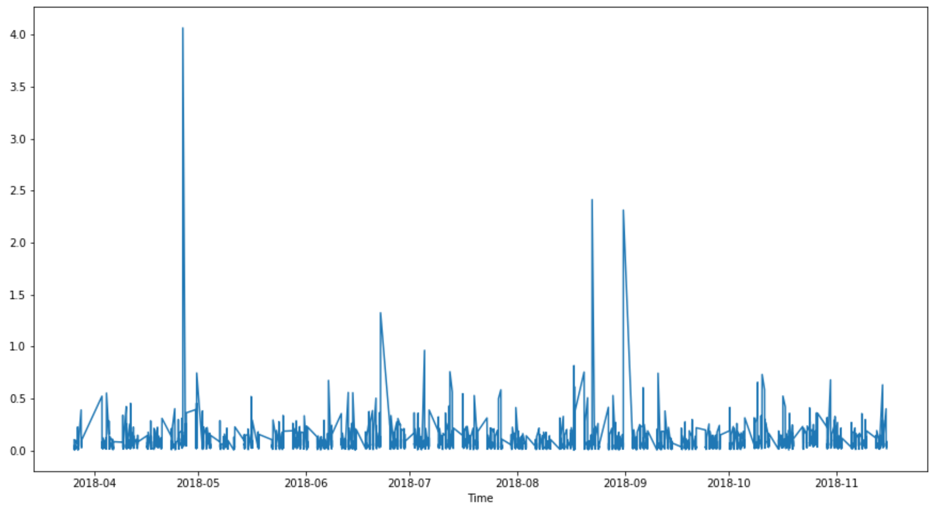

Figure 4.6: AKSONO 10 minutes standard deviation.

The stock has even more periods where no trading are happening. This is shown clearly in these two plots made.

4.2 Fractional Brownian Motion Volatility Model Smoothness

In this section we will introduce a volatility model to our Markowitz portfolio optimisation to model stock volatility defined as in the first chapter. [Gatheral et al., 2014] discovered the fact that increments of log-volatility are approximately Gaussian distributed, which also has been done by other prior studies. Further a mono-fractal scaling relationship (2.7.1) was given for the underlying volatility processes. This then led to a the log-volatility could be modelled using a fractional Brownian motion. We use then apply the volatility model to out stochastic pricing model (2.7.1), with the volatility being the stochastic process driven by the fractional Brownian motion.

4.2.1 Market Data

In the application, this will make a more realistic model compared with the Black Scholes stock price model. We estimate the smoothness of the volatility process for the Oslo Børs Index (.OSEAX), where data are from the Oxford-Man Institute of Quantitative Finance. We use the precomputed 10-min window variance given by the data set. We calculate the smoothness parameter from $m(q, \Delta)$ for different values of q and Δ. Then we do a linear regression of log $m(q, \Delta)$ against Δ. As we mentioned earlier, the scaling can also be found by (2.7.2) given

$$\mathbb{E}[|\log(\sigma_\Delta) - \log(\sigma_0)|^q] = C_q \Delta^{\xi_q}.$$

We shall also be looking at the different distribution of increments of $\log -$volatility.

The regression coefficient form C_q against q gives the Hurst parameter H.

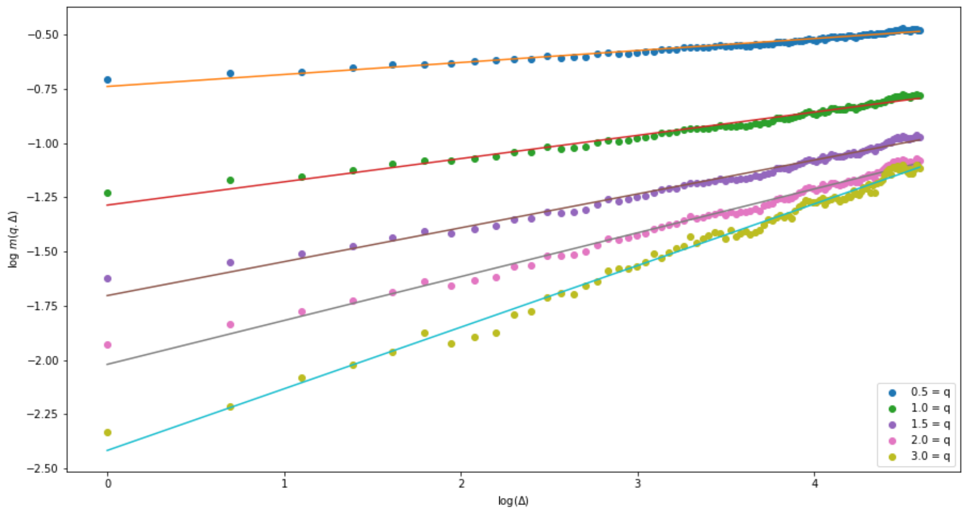

Figure 4.7: Plotting log $m(q,\Delta)$ against $\log(\Delta)$. (.OSEAX)

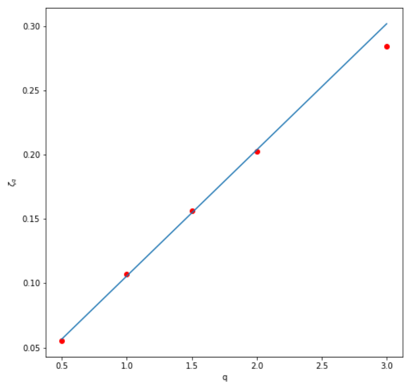

Figure 4.8: Scaling of C_q with q. (.OSEAX)

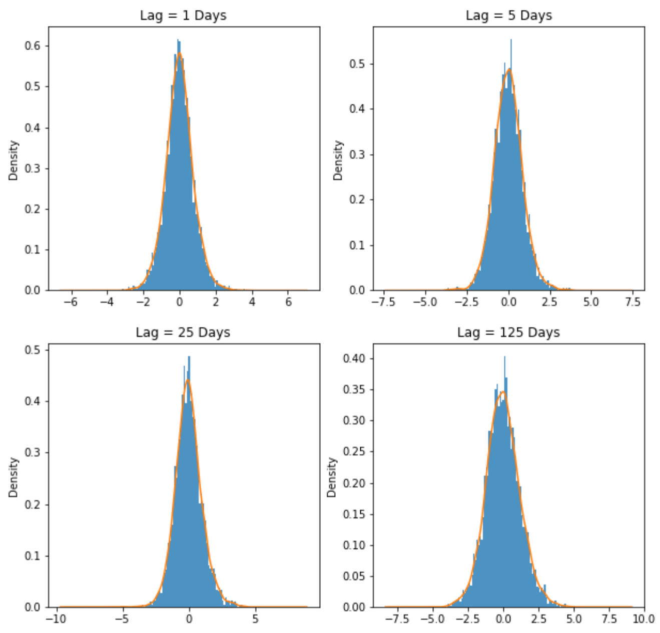

Figure 4.9: Log-increments for different lags Δ. (.OSEAX)

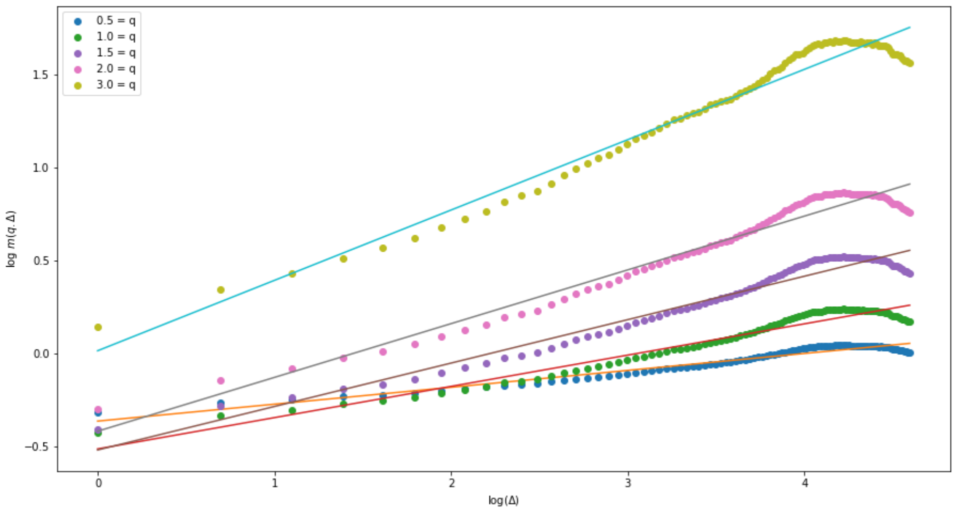

Figure 4.10: Plotting log $m(q, \Delta)$ against $\log(\Delta)$. (.EURNOK)

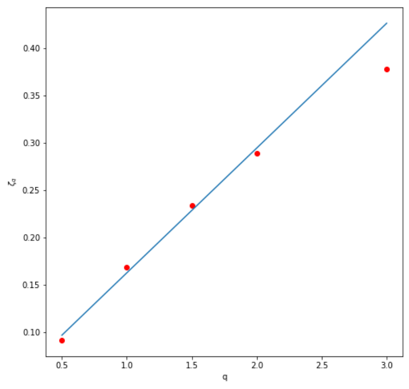

Figure 4.11: Scaling of C_q with q. (.EURNOK)

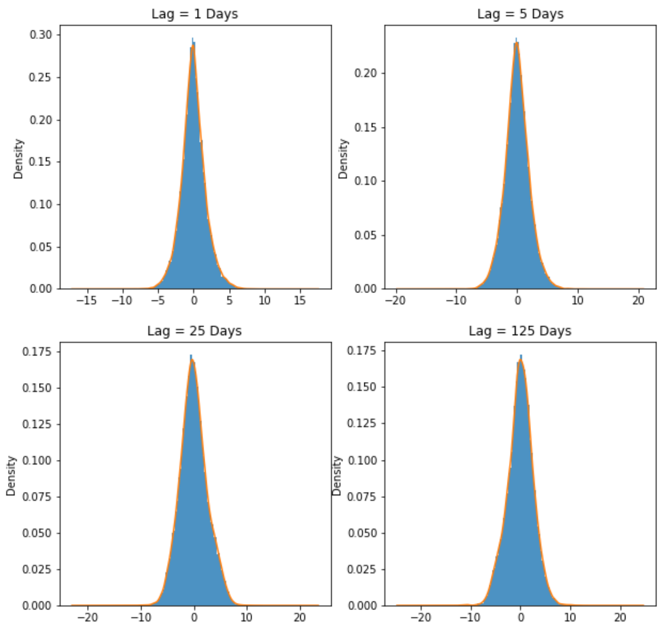

Figure 4.12: Log-increments for different lags Δ. (.EURNOK)

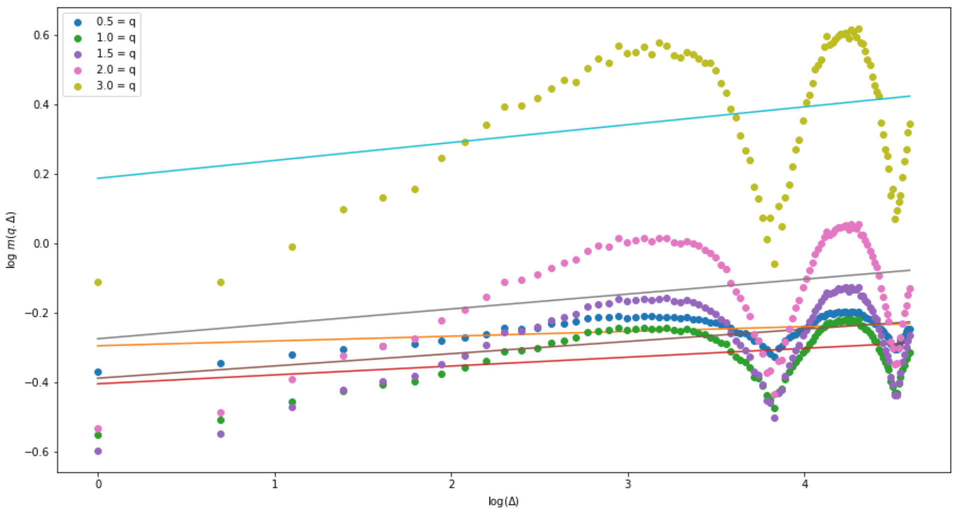

Figure 4.13: Plotting log $m(q, \Delta)$ against $\log(\Delta)$. (.AKSONO)

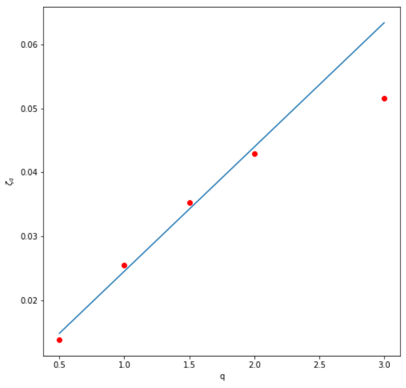

Figure 4.14: Scaling of C_q with q. (.AKSONO)

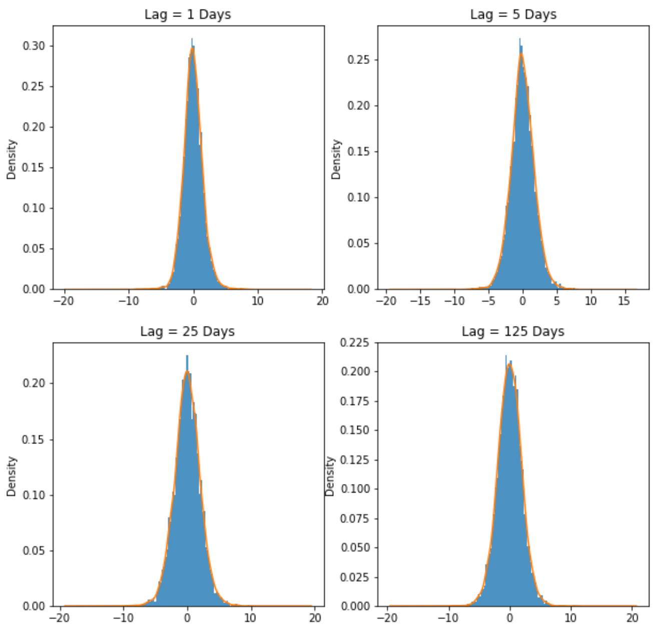

Figure 4.15: Log-increments for different lags Δ. (.AKSONO)

Market	**Hurst Parameter**
.OSAEX	0.0979
.EURNOK	0.0132
.AKSONO	0.0194

Table 4.1: Hurst parameter for different market.

We note that the Hurst parameters are very small, meaning we have rough paths. The smoothness fitting for .OSEAX and .EURNOK shows better result due to the fact that the dataset contains more points then .AKSONO. Both the first mentioned markets is related to what [Gatheral et al., 2014] calls the mono-fractional scaling property. We also note that the distribution of the log-increments are close to normal distributed by comparing the empirical data to the normal fitted curve in yellow. Further the log-log plot tells us that $m(q, \Delta)$ is related to $C_q\Delta^{\xi_q}$, where the latter is directly related to q in a proportion sense. From the plot for the first two markets, the scaling can be expressed as

$$m(q, \Delta) \propto \Delta^{\xi q},$$

from figure 4.7 and 4.10. Also we have the relationship

$$C^q = qH,$$

from figure 4.8 and 4.11. [Gatheral et al., 2014] also mentioned that the estimated parameter H for different time interval, have different values of H. This means that for some time interval, some volatility are more or less rough then certain intervals, e.g. the volatility is very high at the year of 2008, which was the financial crisis.

The log-normal plots indicate out initial assumption about the stylised fact, also reported from [Andersen et al., 2001], where the histogram is Gaussian distributed

4.2.2 Simulated Stock Price

We simulate the stock price movement driven by a Geometric Brownian motion process with stochastic volatility from section 2.7. The simulation procedure can be summarised with the following algorithm

Algorithm 1: Simulating Stock Price with Fractional Volatility.

```
Set initial values $S0, \mu, H, N, T$;
for $N$, simulation numbers do
    Simulate the Fractional Brownian motion;
    Simulate the Geometric Brownian Motion with the
      Fractional Brownian motion volatility;
    Calculate the standard deviation of the GBMRV stock price;
    Calculate the log of the standard deviation;
    Calculate the stock prices $S_0, S_1, \dots$;
end
return N number of stock prices $S_0, S_1, \dots S_N$, standard
  deviation and log-standard deviation;
```

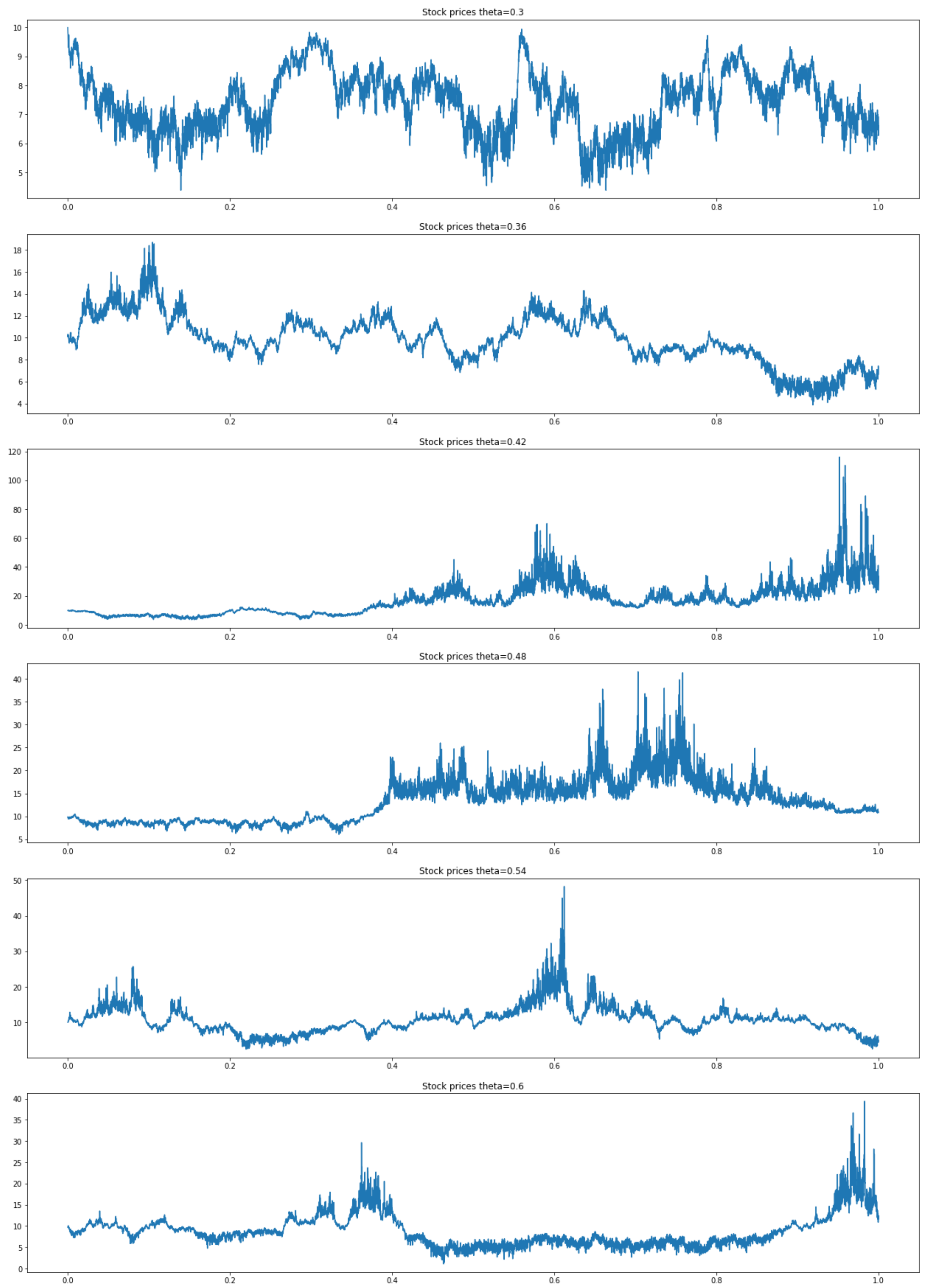

Figure 4.16: Simulated stock prices with different θ values.

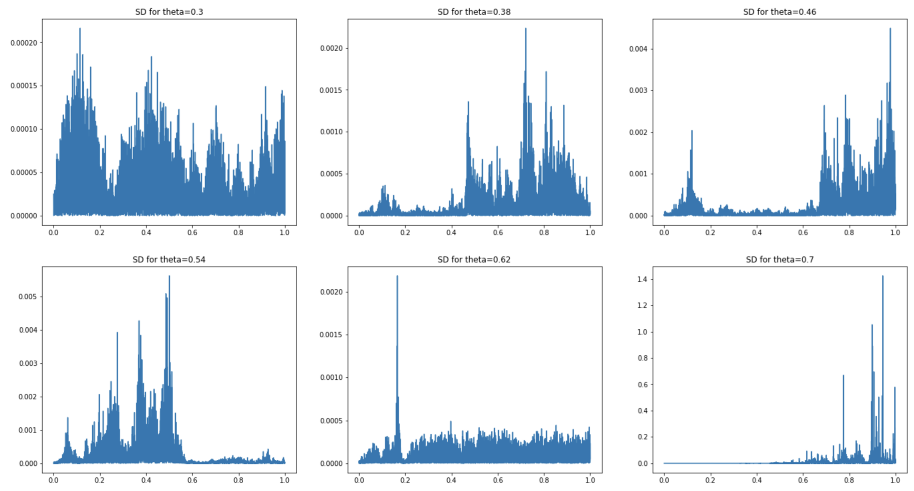

Figure 4.17: Volatility from simulated stock prices with different θ values.

We see that the volatility have the same roughness as our index and currency marked, having low volatility on some points, and high volatility peeks as well.

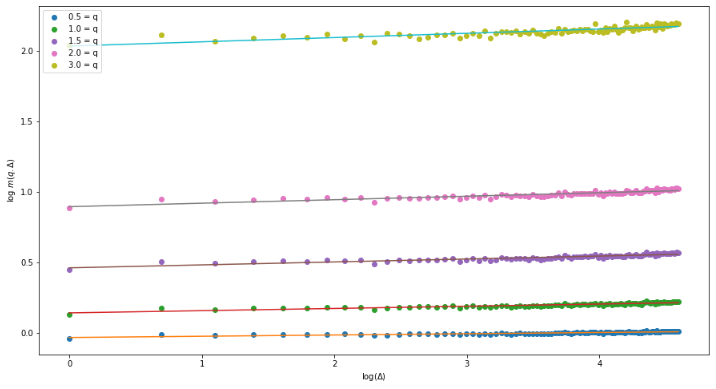

Figure 4.18: Plotting log $m(q, \Delta)$ against $\log(\Delta)$. (Simulated stock prices)

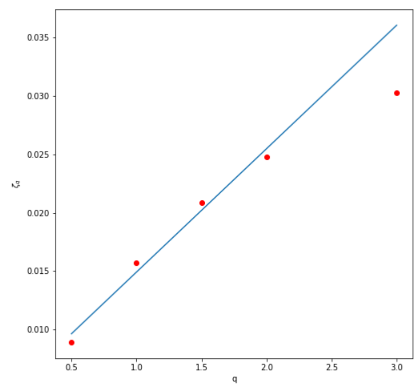

Figure 4.19: Scaling of C_q with q. (Simulated stock prices)

Simulates stock	**Hurst Parameter**
$\theta = 0.3$	0.009997
$\theta = 0.36$	0.007309
$\theta = 0.42$	0.016343
$\theta = 0.48$	0.017463
$\theta = 0.54$	0.012656
$\theta = 0.6$	0.018069

Table 4.2: Hurst parameter for simulates stock prices with different θ values.

4.3 Prediction Stochastic Volatility

In this section the main focus will be on fitting the rough volatility paths using the Fractional Brownian motion given in 2.7. We use the forecasting equation (2.5.10). We measure the forecast accuracy by using the mean squared error function of our prediction and the log-variance

$$P = \frac{\sum_{k=500}^{N-\Delta}\left(\log(\sigma^2_{k+\Delta}) - \widehat{\log(\sigma^2_{k+\Delta})}\right)^2}{\sum_{k=500}^{N-\Delta}\left(\log(\sigma^2_{t+\Delta}) - \mathbb{E}[\log(\sigma^2_{t+\Delta})]\right)^2},$$

where we begin from $k = 500$ and $\mathbb{E}[\log(\sigma^2_{t+\Delta})]$ is the empirical mean of the log-variance over the time period.

4.3.1 Market Data

We plot the actual volatility against the predicted volatility

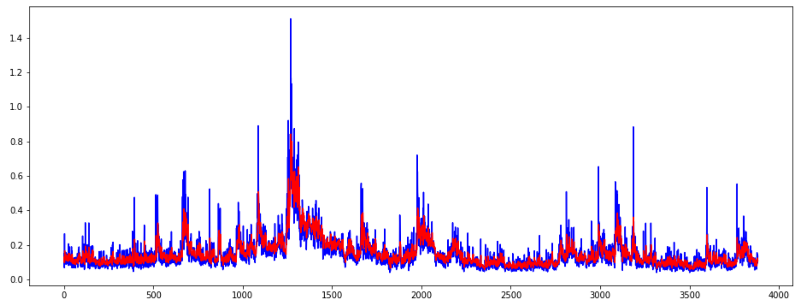

Figure 4.20: Predicted volatility against actual volatility, $\Delta = 1$. (.OSEAX)

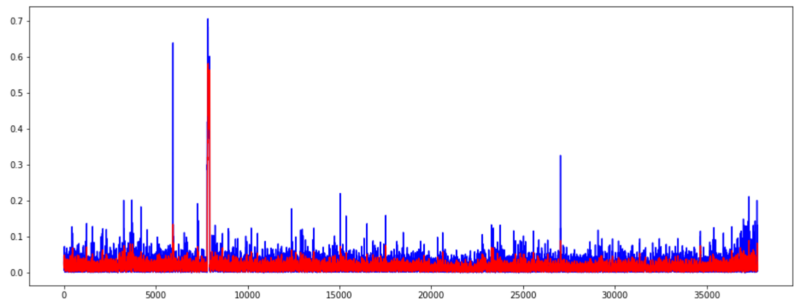

Figure 4.21: Predicted volatility against actual volatility,$\Delta = 1$. (.EURNOK)

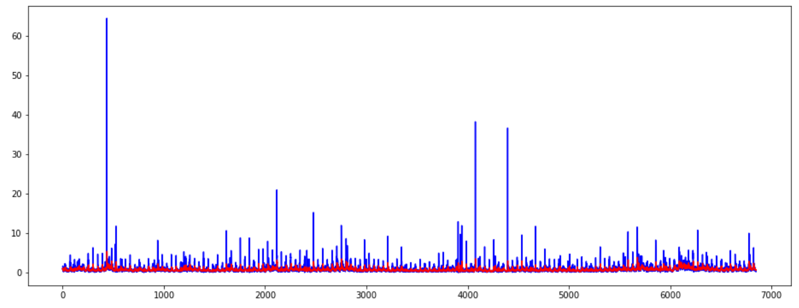

Figure 4.22: Predicted volatility against actual volatility, $\Delta = 1$. (.AKSONO)

Predication from figure 4.20 and 4.21 have in general good fit for $\Delta = 1$. The AKSONO stock prices struggles to find the roughness compared to the former predications. This indicate that the model fits bad when coming to lack of data. The error measure is shown in this table below.

Market	**Error**
.AKSONO $\Delta = 1$	0.0001576
.AKSONO $\Delta = 5$	0.0001578
.AKSONO $\Delta = 25$	0.0001588
.AKSONO $\Delta = 125$	0.0001640
.EURNOK $\Delta = 1$	0.0000268
.EURNOK $\Delta = 5$	0.0000269
.EURNOK $\Delta = 25$	0.0000269
.EURNOK $\Delta = 125$	0.0000270
.OSEAX $\Delta = 1$	0.0002957
.OSEAX $\Delta = 5$	0.0002964
.OSEAX $\Delta = 25$	0.0002999
.OSEAX $\Delta = 125$	0.0003191

Table 4.3: Prediction error for different market.

The error is relative to the stock market, in general when Δ increases, the prediction error also increases. [Gatheral et al., 2014] compared predications with an Autoregressive models and an HAR models. The results showed that the rough volatility model outperformed latter models.

4.3.2 Simulated Data

We plot the predicted volatility for out Geometric Brownian motion with stochastic volatility model.

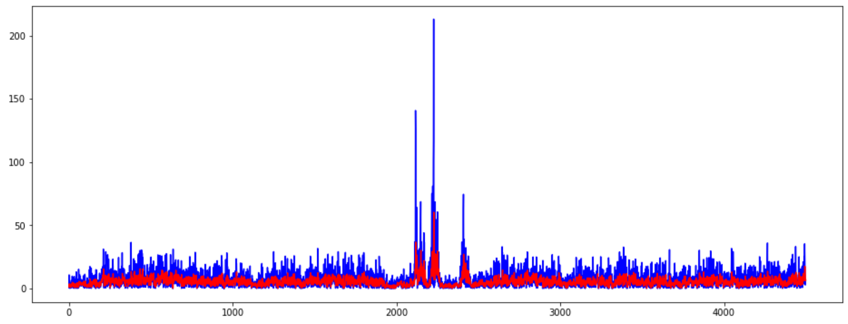

Figure 4.23: Predicted volatility against actual volatility, $\Delta = 1$. (GBMFBM)

Simulated stock	**Error**
$\Delta = 1$	0.0002501
$\Delta = 5$	0.0002506
$\Delta = 25$	0.0002532
$\Delta = 125$	0.0002667

Table 4.4: Prediction error for simulated stock.

4.4 Portfolio Optimisation

In chapter 2, we obtained the explicit formula for the optimal portfolio given some self-financing strategy (2.6.2). We first assume the stock prices follow a Geometric Brownian motion, and we optimise the portfolio allocation by controlling the optimal weights (ω_0, ω_1) with using a simple network. We assume that the weights (ω_0, ω_1) are randomly sampled from $[0, 1]$ and the μ and σ given constant. The following portfolio returns are given with these plots:

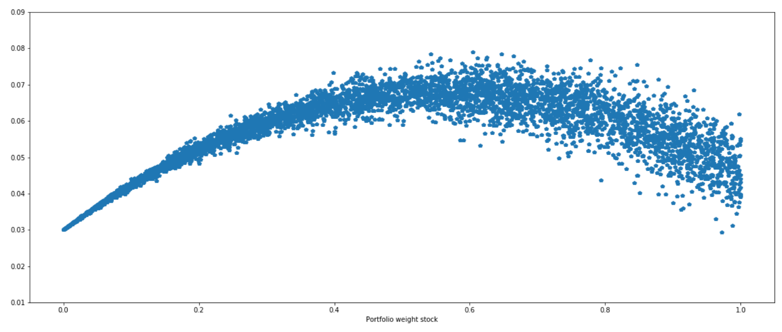

Figure 4.24: Geometric Brownian motion portfolio return with $\mu = 0.03,\ \sigma = 0.316$.

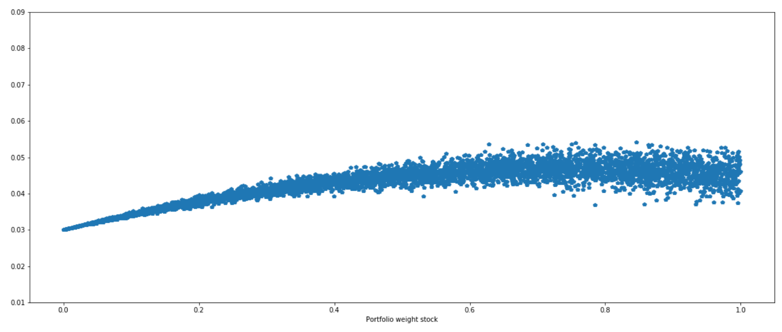

Figure 4.25: Geometric Brownian motion portfolio return with $\mu = 0.03,\ \sigma = 0.116$.

Both plots shares the same mean μ, but different standard deviations σ. The latter result shows a significant lower spread and a less obvious curve compared to the former plot. The sampling size made with these plots are 5000 sample means. This runs fairly quickly on a laptop computer with only CPU power.

Network Structure

We will use the PyTorch package in python for building our model. The PyTorch library is an easy starting point for machine learning compared to TensorFlow. We can define dynamic computational graphs, which makes changes to the model easy during runtime, compared to TensorFlow that does not include runtime options. The network parameters is then given as follows:

- Input shape: 3
- Hidden layer size: 7
- Output shape: 1
- Batch size: 500
- Learning rate = 0.001
- Number of epoch: 40,000
- Criterion: Mean squared error
- Optimiser: Stochastic gradient decent
- Train sample size: 5000
- Test sample size: 200
- $\mu = 0.03$, $\sigma = 0.2$

The network structure consists of

1. Input layer: Linear layer (input, hidden)
2. Hidden layer: Exponential linear activation function on Linear layer (hidden, hidden)
3. Output Layer: Linear layer (hidden, output)

Model Fitting
We fit the model using the network described above.

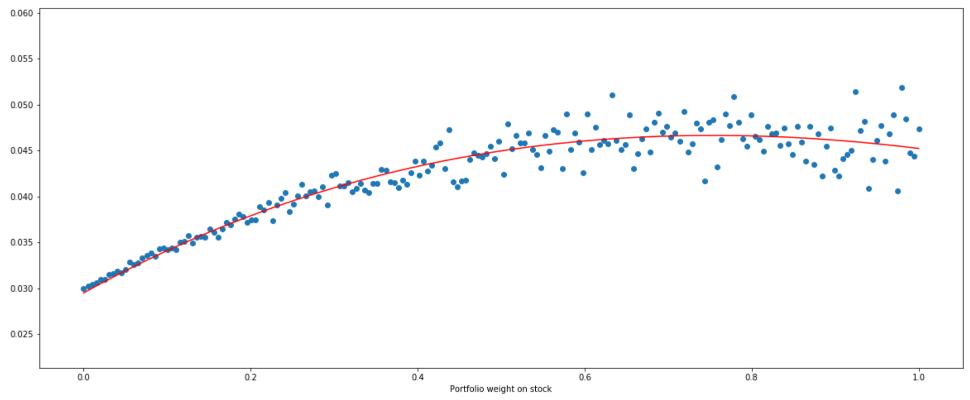

Figure 4.26: Network fitting the GBM stock price for weight ω_1. ($\mu = 0.03, \sigma = 0.316$)

The loss is visualised using tensorboard, which is a tool from Tensorflow, and working with PyTorch package.

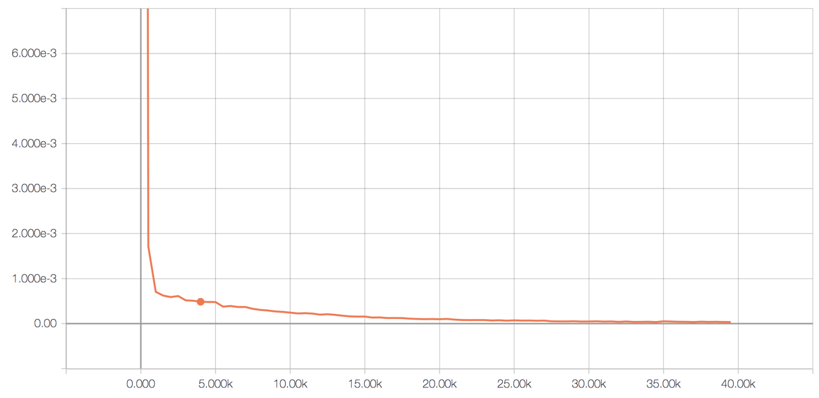

Figure 4.27: Network loss the GBM stock price for weight ω_1.

The loss after 15,000 iterations does not really improve much for our model. We obtain the optimal weigh (ω_0, ω_1) for the test data set containing 200 random sample from the portfolio given

$$\max\left(\omega_0 r + \omega_1 \mu - \frac{1}{2}\omega_1 \sigma^2 - \frac{1}{2}(\omega_1 \sigma)^2\right), \text{ s.t. } \omega_0 + \omega_1 = 1$$

where $\hat{\omega_1} = 0.73$, and $\hat{\omega_0} = 1 - \hat{\omega_1} = 0.23$. We compare the result to the exact solution from equation (2.6.3), where we have that $\omega_1 = 0.6$ and $\omega_0 = 0.4$.

4.4.1 Market Data

Now we turn our attention to the real market data. We optimise the portfolio using a direct method of allocation. Let us make some assumption for our optimisation scheme, firstly the log returns of investments (2.3.1) are independent, identically distributed random sequence. In particular, each return r_t is distributed accordingly to the same and possibly unknown probability distribution $\mathbb{P}$. This assumption is compatible with the assumption about EMH from section 2.1.1. Secondly we assume that we restrict our trading to not include any short positions or borrowing money, meaning that we exclude $\omega_0, \omega_1 < 0$.

4.4.2 Simulated Data

We simulate the portfolio returns from Geometric Brownian motion with stochastic volatility. We modify our network structure to include a deeper network consisting of these layers

1. Input: Rectified linear unit activation function on linear layer (input, hidden)
2. Hidden: Exponential linear activation function on linear layer (hidden, hidden)
3. Output: Linear layer (hidden, output),

with these parameters

- Input shape: 3
- Hidden layer size 10
- output shape: 1

- Batch size: 500
- Learning rage = 0.001
- Number of epoch: 30,000
- Criterion: Mean squared error
- Optimiser: Stochastic gradient decent
- Train sample size: 5000
- Test sample size: 200
- $\mu = 0.03, \theta = 0.13, H = 0.09$

The fitted model then can be plotted

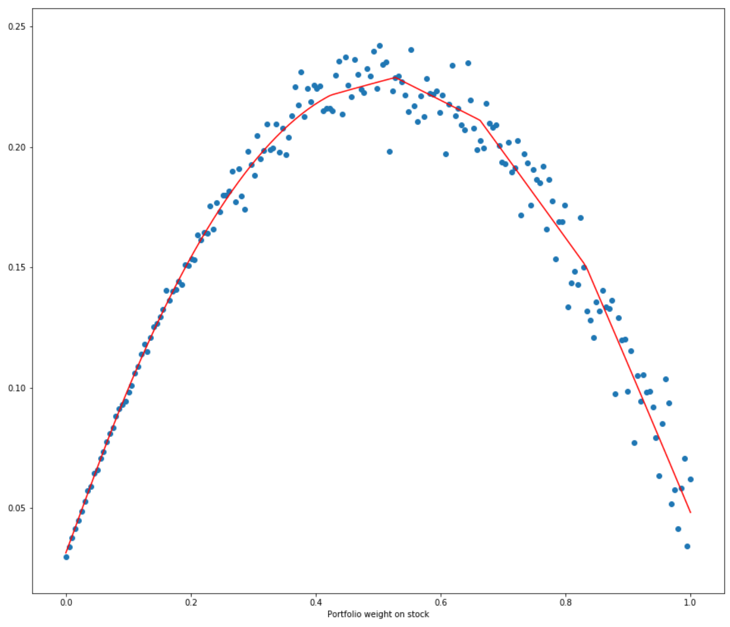

Figure 4.28: Fitting Geometric Brownian motion with fractional volatility.

These data points seems to be more spread then the Geometric Brownian motion test data set. The shape of the optimal allocation seems to be more closely to 0.5 for ω_1.

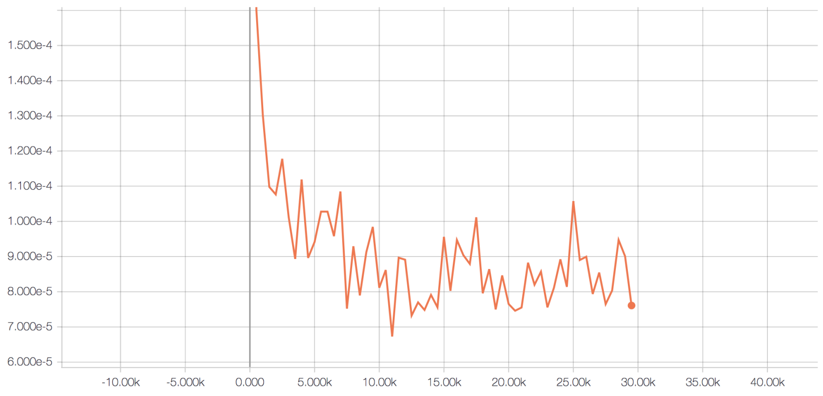

Figure 4.29: Training loss Geometric Brownian motion with fractional volatility.

Here the loss function reaches the lowest level after 10,000 iterations of training. The loss accuracy after that have slightly variations.

4.5 Evaluation of the Geometric Brownian Motion with Rough Volatility

We compare our GBMRV against the Oslo Børs index. The following table shows the estimated yearly return and volatility of the .OSEAX and GBMRV. The table below are specified by each year from 2013 to 2019. The second column is the estimated μ based on historical log-returns. Columns 4 and 5 are the volatility for the index with actual and predicted σ. Columns 6 corresponds to the expected stock price of the GBMRV model. The actual index price are in the last column.

Period	μ	**Annual** μ	**Actual** σ	**Predicted** σ	**Simulated GBM** S_t	**Actual** S_t
2003	0.00187	0.1344	0.116	0.111	157.81	157.73
2004	0.00128	0.3212	0.111	0.108	180.65	180.42
2005	0.00167	0.4175	0.128	0.130	249.20	248.78
2006	0.00120	0.2989	0.158	0.159	378.14	377.68
2007	0.00045	0.1128	0.155	0.155	509.53	509.28
2008	-0.00275	-0.0693	0.305	0.316	568.68	570.14
2009	0.00163	0.4095	0.252	0.236	285.68	285.18
2010	0.00052	0.132	0.169	0.164	429.75	429.51
2011	-0.00037	-0.0927	0.181	0.182	490.01	490.16
2012	0.00046	0.1155	0.130	0.125	446.98	446.76
2013	0.00073	0.1809	0.089	0.087	501.83	501.47
2014	0.00014	0.0353	0.110	0.112	601.02	600.92
2015	0.00012	0.0290	0.145	0.143	622.62	622.52
2016	0.00078	0.1874	0.154	0.154	641.35	640.84
2017	0.00065	0.1621	0.089	0.087	773.42	772.91
2018	0	-0.0022	0.121	0.124	908.90	908.88
2019	0.00126	0.0077	0.122	0.112	907.20	906.89

Table 4.5: Simulated GBM compared against .OSEAX index.

The simulated GBM paths under these parameters are then plotted

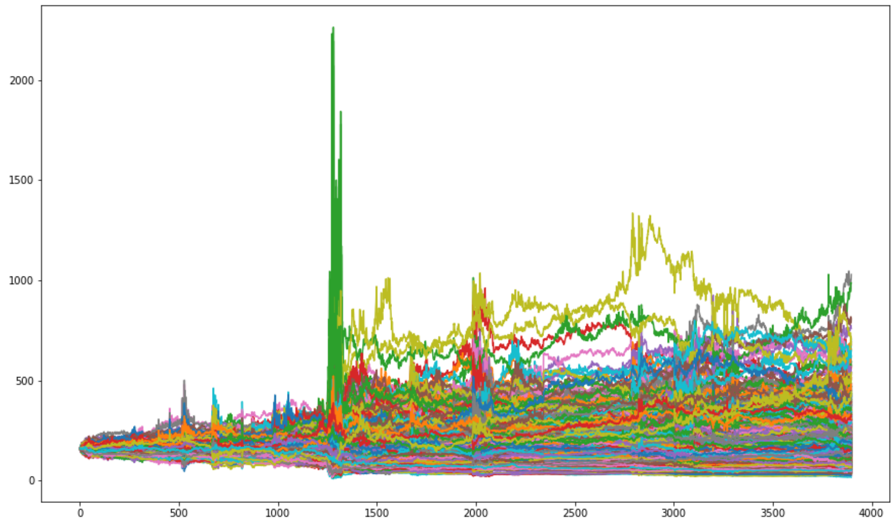

Figure 4.30: Simulated Geometric Brownian Motion with estimated parameter.

From table 4.5 we note that the estimated stock prices are close to the actual prices. Figure 4.30 are made of 1000 samples where each period of year are sampled with their respective parameters of μ and σ_t is changed over the whole period. The high peek is due to high volatility during financial crisis. The estimated stock price $\mathbb{E}(S(t))$ is found by equation (2.5.8)

Stock Price Prediction

Now that we have verified the GBMRV model does a good job of predicting the index prices, we can simulate stock prices using this model.

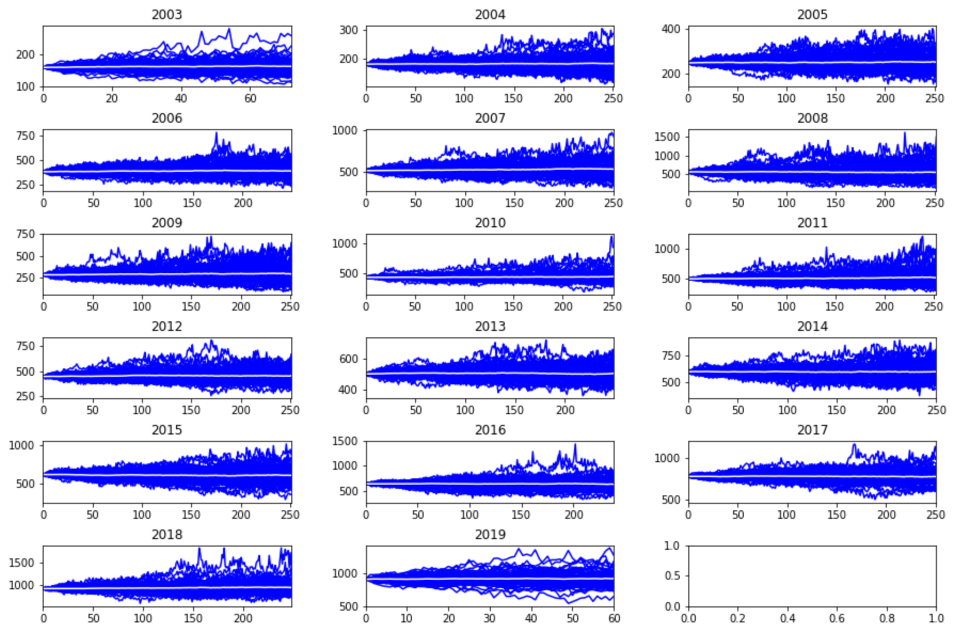

Figure 4.31: Simulation of GBMRV paths from different years.

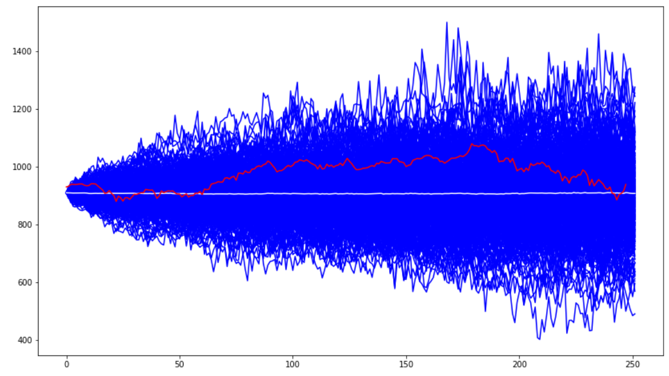

Figure 4.32: Simulation of GBMRV paths from 2018.

A closer look at the most recently year from the .OSEAX data, the plot shows the simulated stock prices in blue. The red line is the actual price, and the white line is average price. The average price is not able to capture the movements of the price, this is due to the simulations being stochastic by model. We note that at some points the average prices are in the same line as the actual prices. For the year of 2018, the starting price ends up in the same category as the ending price. At both these points, the simulated model are able in some sense to capture the right price. We also observe that the actual prices are covered by the simulation with 500 samples. This is a good sigh that the model can be used in practice to say something about the future prices.

Chapter 5

Conclusion

To summarise this thesis, we explored the optimal portfolio allocation in a market with a risk-free asset and a risky asset in sense of the Oslo Børs index, EURO to NOK currency and the Aker Solution stock prices. We found that in the first two markets, there were indication of the volatility process being rough in sense of the Hurst parameter $H < 0.1$. Due to this, we introduced a simple model for the price dynamics based on the Black Scholes model, where we afterwards introduced a stochastic volatility model based on the rough volatility. We then began to simulate Geometric Brownian motion with rough volatility by replacing σ with a stochastic σ_t, which was predicted using the Fractional Brownian motion variance based on the actual stock market. We used this to simulate a GBMRV with different μ and σ_t based on the estimated values from the .OSEAX close prices. We found that the price model with rough volatility model manages to find the peeks where the stock prices had high volatility. When implementing the volatility in the simulated model, we find that the average volatility consists with the actual volatility in the market. This also leads to that the prices are close to the actual market prices.

Under our optimisation, we assumed that there were some optimal strategy to allocate between the risk-free asset and the risky security. We found that using the simple Geometric Brownian motion requires a large data set, which is often a problem, when working with finance data. By making a GBM with rough volatility, we replace σ with a volatility process based on the fractional Brownian motion. The latter process requires the Hurst parameter to be estimated, in general these markets have rough stochastic volatility embedded.

As for future work, we could extend the model by including a different environment in connection with portfolio allocation. This means that we could include information such as a portfolio return based on future returns and as well as include different methods of machine learning for optimisation. We could also collect more data points from the stock market, i.e. include more trading days, which we say had effect for being able to find the rough volatility property. The framework of this thesis, was build on the face that our stock dynamics under the Geometric Brownian motion could be extended to simulate more stocks. This would imply that we need to measure the correlation between these stocks. The theoretical foundation as been introduced, and for the numerical simulation we need to extend our model to include correlation. Other methods of optimising optimising could also be used, such as having risk constraints based on the investors risk preference.

Code

Some codes are not included due to repetition of the same calculations.

1 Importing libraries

```
In [ ]: import numpy as np
        import matplotlib.pyplot as plt
        import pandas as pd
        import os
        import sys
        from fbm import FBM
        import feather

        import matplotlib.cm as cm
        colours = iter(cm.rainbow(np.linspace(0, 1, 4)))

        import torch
        import torch.nn as nn
        import numpy as np
        import matplotlib.pyplot as plt
        import torchvision
        import torchvision.transforms as transforms
        import torch.nn.functional as F
        from logger import Logger

        from sklearn.metrics import mean_squared_error
        from scipy.special import gamma
```

2 Functions

```
In [ ]: def GBM(S0,mu,sigma,N,T):
            """
            Simulate geometric Browmina motion
            """
            dt = T/N
            t = np.linspace(0.,T,N+1)
            W = np.zeros(N+1)
            W[1:] = np.cumsum((np.random.normal(0., 1., N)*np.sqrt(dt)))
            S = S0*np.exp((mu-(0.5*sigma**2))*dt+sigma*W)
            return S,t

        def Portfolio(w1,mu,sigma,N,T,sample):
            """
            Simulate portfolio
            """
            t = np.linspace(0.,T,N)
            dt = t[1]-t[0]
            w0 = 1-w1
            r = 0.02
            zt = (w0*r + w1*(mu-0*sigma**2)-0.5*(w1*sigma)**2)*dt
            +w1*sigma*np.random.randn(sample,N)*np.sqrt(dt)
    pt = np.exp(np.cumsum(zt,axis=1))
    pt = np.mean(np.log(pt))
    return pt,w0,w1

def Batch(datapd,sample_size,shuffle):
    """
    Making batches for PyTorch
    """
    if shuffle==True:
        df = datapd.sample(n=sample_size,replace=True)
        mu_b = torch.tensor(np.vstack(df['mu'].values).astype(np.float32))
        sigma_b = torch.tensor(np.vstack(df['sigma'].values).astype(np.float32))
        w1_b = torch.tensor(np.vstack(df['w1'].values).astype(np.float32))
        pT_b = torch.tensor(np.vstack(df['pT'].values).astype(np.float32))
        x_b = torch.cat((mu_b,sigma_b,w1_b),1)
    else:
        df = datapd
        mu_b = torch.tensor(np.vstack(df['mu'].values).astype(np.float32))
        sigma_b = torch.tensor(np.vstack(df['sigma'].values).astype(np.float32))
        w1_b = torch.tensor(np.vstack(df['w1'].values).astype(np.float32))
        pT_b = torch.tensor(np.vstack(df['pT'].values).astype(np.float32))
        x_b = torch.cat((mu_b,sigma_b,w1_b),1)
    return x_b, pT_b

def MakeTraindata(mu,sigma,N,steps,T,samples):
    """
    Make dataframe for train data
    """
    testlist = []
    for i in range(N):
        w1 = np.asscalar(np.random.uniform(0.0,1.,1))
        pt,w0,w1 = Portfolio(w1,mu,sigma,steps,T,samples)
        testlist.append([mu,sigma,w0,w1,pt])
    df = pd.DataFrame(data = testlist, columns = ["mu","sigma","w0","w1","pT"])
    return df


def MakeTestdata(mu,sigma,N,steps,T,samples):
    """
    Make dataframe for test data
    """
    trainlist = []
    w1_list = np.linspace(0.,1.,N2)
    for i in range(N):
        pt,w0,w1 = Portfolio(np.asscalar(w1_list[i]),mu,sigma,steps,T,samples)
        trainlist.append([mu,sigma,w0,w1,pt])
    df = pd.DataFrame(data = trainlist, columns = ["mu","sigma","w0","w1","pT"])
```

```
            return df

        def logTransfer(time_series):
            """
            Log-return
            """
            ts = []
            for i in range(1,len(time_series)):
                ts.append(np.log(time_series[i]/time_series[i-1]))
            return ts

        def Stock_price(S0,mu,sigma,dt):
            return S0*np.exp((mu+(0.5*sigma**2))*dt)

In [ ]: def fbms(sample,T,h):
            """
            Makes a matrix of fractional Brownian motion with N samples
            """
            fbm_list = [[FBM(T-1,h).fbm()] for _ in range(sample)]
            return np.array(fbm_list).reshape(sample,T)

        def Hurst(ts):
            """
            Estimating Hurst parameter
            """
            lags = range(2, 100)
            # Calculate the array of the variances of the lagged differences
            tau = [np.sqrt(np.std(np.subtract(ts[lag:], ts[:-lag]))) for lag in lags]

            # Use a linear fit to estimate the Hurst Exponent
            poly = np.polyfit(np.log(lags), np.log(tau), 1)

            # Return the Hurst exponent from the polyfit output
            return poly[0]*2.0

        def GBM_fbm(S0,mu,sigma0,h,N,T):
            """
            Simulate geometric Brownian motion with fractional Brownian motion
            as volatility
            """
            theta = np.sqrt(h)
            dt = 1/N
            t = np.linspace(0.,T,N+1)
            W = np.zeros(N+1)
            fbm = FBM(N,h).fbm()
            sigma = sigma0*np.exp(theta*fbm)
            W[1:] = np.cumsum((np.random.normal(0., 1., N)*np.sqrt(dt)))
            S = S0*np.exp((mu-(0.5*sigma**2))*dt+sigma*W)
            return S,t,sigma


        def Portfolio_fbm(w1,mu,theta,N,T,sample,fbm):
            """
            Simulate portfolio from geometric Brownian motion with fractional
            Brownian motion as volatility
            """

            t = np.linspace(0.,T,N)
            dt = t[1]-t[0]
            w0 = 1-w1
            r = 0.02
            sigma = np.exp(theta*fbm)
            zt = (w0*r + w1*(mu-(0.5*sigma**2)) - 0.5*(w1*sigma)**2)*dt
                    + w1*sigma*np.random.randn(sample,T)*np.sqrt(dt)
            pt = np.exp(np.cumsum(zt,axis=1))
            pt = np.mean(np.log(pt))
            return pt,w0,w1

        def logTransferIncrement(time_series):
            """
            Log-transformation of increments
            """
            ts = []
            for i in range(1,len(time_series)):
                ts.append(np.log(time_series[i])-np.log(time_series[i-1]))
            return ts

In [ ]: def shift_delta(log_vol,q, x):
            return [np.mean(np.abs(log_vol
                - log_vol.shift(lag)) ** q) for lag in x]

        def plot_zeta(vol):
            """
            Plotting scaling of log(m(q,Delta)) and log(Delta)
            """
            fig, ax = plt.subplots()
            plt.xlabel('log$(\Delta)$')
            plt.ylabel('log $m(q.\Delta)$')
            plt.ylim=(-4, -1)
            zeta_q = list()
            q_list = np.array([.5, 1, 1.5, 2, 3])
            x = np.arange(1, 100)
            for q in q_list:
                ax.plot(np.log(x), np.log(shift_delta(vol, q, x)), 'o', label=str(q) + ' = q')
                model = np.polyfit(np.log(x), np.log(shift_delta(vol, q, x)), 1)
                ax.plot(np.log(x), np.log(x) * model[0] + model[1])
                zeta_q.append(model[0])
            plt.legend()
            return zeta_q, q_list

        def c_tilde(h):
            """
            Calculate the constant c from equation (2.5.9)
            """
            return gamma(3. / 2. - h) / gamma(h + 1. / 2.) * gamma(2. - 2. * h)

        def forecast(rvdata, h, date, nLags, delta, nu):
            """
            Forecast rough volatility
            """
            i = np.arange(nLags)
            cf = 1./((i + 1. / 2.) ** (h + 1. / 2.) * (i + 1. / 2. + delta))
            ldata = rvdata.truncate(after=date)
            l = len(ldata)
            ldata = np.log(ldata.iloc[l - nLags:])
            ldata['cf'] = np.fliplr([cf])[0]
            ldata = ldata.dropna()
            fcst = (ldata.iloc[:, 0] * ldata['cf']).sum() / sum(ldata['cf'])
            return np.exp(fcst + 2 * nu**2 * c_tilde(h) * delta**(2 * h))

        def prediction(vol,delta=1):
            """
            Volatility prediction
            """
            h = Hurst(vol)
            nu = np.sqrt(h)
            rvdata = vol
            n = len(rvdata)
            delta = delta
            nLags= 500
            dates = rvdata.iloc[nLags:n-delta].index
            rv_predict = [forecast(rvdata, h=h, date=d, nLags=nLags,
                                   delta=delta, nu=nu) for d in dates]
            rv_actual = rvdata.iloc[nLags+delta:n].values
            vol_actual = np.sqrt(np.multiply(rv_actual,252))
            vol_predict = np.sqrt(np.multiply(rv_predict,252))
            fig = plt.figure(figsize=(16, 6))
            plt.plot(vol_actual,color='b')
            plt.plot(vol_predict,color='r')
            T = vol_actual.shape[0]-delta
            mse = 0
            for i in range(nLags,T):
                mse = mse + (((np.log(vol_actual[i+delta]) - np.log(vol_predict[i+delta]))**2)/
                             (np.log(vol_actual[i+delta]) - np.mean(np.log(vol_actual)))**2)
```

```
            mse = 1/(T-nLags)
            return vol_actual, vol_predict, np.round(mse,decimals=8)

        def plot_q_zeta(q,zeta):
            """
            Fitting Hurst
            """
            plt.figure(figsize=(8,8))
            plt.xlabel('q')
            plt.ylabel('$\zeta_{q}$')
            plt.plot(q, zeta, 'or')
            line = np.polyfit(q[:4], zeta[:4],1)
            plt.plot(q, line[0] * q + line[1])
            h_est= line[0]
            return h_est

        def plot_log_increments(scale, vol):
            """
            Plot log-normal plot of the increments
            """
            def xDel(x, lag):
                return x-x.shift(lag)

            def sdl(lag):
                return (xDel(np.log(vol), lag)).std()

            sd1 = (xDel(np.log(vol), 1)).std()
            f, ax = plt.subplots(2,2,sharex=False, sharey=False, figsize=(10, 10))

            for i_0 in range(0, 2):
                for i_1 in range(0, 2):
                    la = scale ** (i_1*1+i_0*2)

                    hist_val = xDel(np.log(vol), la).dropna()
                    std = hist_val.std()
                    mean = hist_val.mean()

                    ax[i_0][i_1].set_title('Lag = %s Days' %la)
                    n, bins, patches = ax[i_0][i_1].hist(hist_val.values,
                                                 bins=80, normed=1, alpha=0.8)
                    hist_val.plot.density(ax=ax[i_0][i_1],legend=None)


In [ ]: def GBM_fbm_paths(S0,mu,h,N,T):
            """
            Sample mutiple fractional Brownian motions
            """
            theta = np.round(np.linspace(0.3,0.6,6),decimals=3)
```

```
    df = pd.DataFrame()
    for i in range(theta.shape[0]):
        s,t = GBM_fbm(S0,mu,h,N,T,theta[i])
        temp = pd.DataFrame(data = s, index = t, columns = ["s_"+str(i)])
        temp["std_"+str(i)] = temp["s_"+str(i)].rolling(2).std()
        temp["log_std_"+str(i)] = np.log(temp["std_"+str(i)])
        df = pd.concat([df, temp],axis = 1)
    df = df.dropna(axis=0)
    return df,theta
```

3 Fractional Brownian motion

```
In [ ]: """
        Simulate some fractional Brownian paths
        """
        S0 = 10
        mu = 0.03
        h = 0.097
        N = 5000
        T = 1
        fbm_df, theta = GBM_fbm_paths(S0, mu, h, N, T)
```

```
In [ ]: """
        Simulate random portfolio allocation
        """
        theta = 0.13
        sample = 3000
        N = 3000
        pt_list = []
        w1_list = []
        w0_list = []
        for i in range(sample):
            fbm = fbms(sample,T,h)
            w1 = np.asscalar(np.random.uniform(0.,1.,1))
            pt,w0,w1 = Portfolio_fbm(w1,mu,theta,N,T,sample,fbm)
            pt_list.append(pt)
            w1_list.append(w1)
            w0_list.append(w0)
            if i%500==0:
                print(i)
        plt.plot(w1_list,pt_list, "p")
```

```
In [ ]: """
        Plotting smoothness and predicting volatility
        """
        df2 = df[["s_5","std_5","log_std_5"]]
        df2 = df2.reset_index()
        df2 = df2.drop("index",axis=1)
```

```
zeta, q = plot_zeta(df["log_std_5"])
h = plot_q_zeta(q,zeta)
rvdata = pd.DataFrame(df2['std_5']**2)
prediction(rvdata,delta=1)
```

4 Aker Solution stock

```
In [ ]: """
        Reading the data set, and cleaning
        """
        aksono = pd.read_csv('aksono_10min.csv')
        def format_data(stock):
            vol = pd.DataFrame(stock['std'])
            stock = stock[stock !=0 ]
            stock = stock[~stock.isin([np.nan, np.inf, -np.inf]).any(1)]
            stock = stock.reset_index()
            vol = vol[vol != 0]
            vol = vol[~vol.isin([np.nan, np.inf, -np.inf]).any(1)]
            vol['var'] = vol['std']**2
            vol['log_std'] = np.log(vol['std'])
            vol = vol.reset_index()
            log_vol = logTransferIncrement(vol['var'])
            return stock,vol,log_vol

        stock,vol,log_vol = format_data(aksono)

        stock['time'] = pd.to_datetime(stock['time'], utc=True)
```

```
In [ ]: """
        Plotting distribution of the volatility and prediction volatility
        """
        plot_log_increments(5, vol['var'])
        zeta, q = plot_zeta(vol['log_std'])
        h = plot_q_zeta(q,zeta)
        rvdata = pd.DataFrame(vol['var'])
        actual, predict, mse = prediction(rvdata,delta=1)
```

5 Oslo Børs index

```
In [ ]: """
        Read and clean the data set
        """
        stock = pd.read_csv('oxfordmanrealizedvolatilityindices.csv', sep=';')
        stock = stock.set_index('Symbol == .OB', drop = True)
        volatility_intra_oseax = stock.loc[".OSEAX","rv10"].values
        oseax_df = stock.loc[".OSEAX",:]
        oseax_df['Unnamed: 0'] = pd.to_datetime(oseax_df['Unnamed: 0'], utc = True)
        oseax_df = oseax_df.rename(columns={'Unnamed: 0':'time'})
        oseax_df = oseax_df.set_index("time")

In [ ]: """
        Plot log-normal distribution of the volatility and the volatility
        """
        plot_log_increments(5,oseax_df["rv10"])
        plt.plot(oseax_df.index,oseax_df['rv10'])

In [ ]: """
        Reducing the dataset
        """
        vol_oseax = oseax_df["rv10"][:-1].copy()
        simple_oseax_pd = pd.DataFrame(data = vol_oseax, columns=['rv10'])
        simple_oseax_pd['std'] = np.sqrt(simple_oseax_pd['rv10'])
        simple_oseax_pd['log_std'] = np.log(simple_oseax_pd['std'])
        simple_oseax_pd['close_price'] = oseax_df['close_price'][:-1].values
        simple_oseax_pd['log_st'] = logTransferIncrement(oseax_df["close_price"].values)
        years = list(simple_oseax_pd.index.year.unique())

In [ ]: """
        Plotting the smoothness and prediction volatility
        """
        zeta,q = plot_zeta(simple_oseax_pd['log_std'])
        h = plot_q_zeta(q,zeta)
        rvdata = pd.DataFrame(simple_oseax_pd['rv10'])
        actual, predict, mse = prediction(rvdata,delta=1)
```

6 Geometric Brownian motion with rough volatility model

```
In [ ]: """
        Calculate som usefull information about the stock such as
        mean return and sigma
        """
        df = simple_oseax_pd[500:-2]
        years = list(df.index.year.unique())
        vol = predict.reshape(3882)
        mean_return = {}
        mean_return_annualy = {}
        day_dict = {}
        start_prices = {}
        sigma_pred_dict = {}
        l = 0
        temp_len = 0
        for y in years:
            y = str(y)
            l = temp_len
            temp = df.loc[df.index.strftime("%Y") == y]
            temp_len2 = temp.shape[0]
            temp_len = temp_len + temp.shape[0]
            mu_hat = float(temp["log_st"].mean(axis=0))
            mu_annualy = float(temp["log_st"].sum(axis = 0))
            sigma = np.mean(vol[l:temp_len])
            mean_return[y] = mu_hat
            mean_return_annualy[y] = mu_annualy
            sigma_pred_dict[y] = sigma
            start_prices[y] = temp["close_price"][0]
            day_dict[y] = temp_len2
            l = temp_len
```

```
In [ ]: """
        Simulation of GBM paths with predicted volatility
        """
        vol = predict[:3899].reshape(3882)
        gbm_df = pd.DataFrame()
        stock_price = []
        S0 = start_prices.get("2003")
        fig = plt.figure(figsize=(15, 9))
        for i in range(1000):
            stock_price = []
            S0 = start_prices.get("2003")
            N_teller = 0
            for key in sorted(day_dict):
                N = day_dict.get(key)
                T = 1
                mu = mean_return_annualy.get(key)
                sigma = vol[N_teller:(N_teller+N+1)]
                N_teller = N_teller + N
                st,t = GBM(S0,mu,sigma,N,T)
                S0 = st[-1]
                st = st.tolist()
                stock_price.extend(st)
            gbm_df = pd.concat([gbm_df,pd.DataFrame(stock_price)],axis = 1)
            plt.plot(stock_price)
        plt.savefig("gbm_est.png",bbox_inches = 'tight')
        gbm_df.columns = range(gbm_df.shape[1])
```

```
In [ ]: """
        Stock price calculation based on the predicted parameters
        """
        s = []
        for key in sorted(day_dict):
            dt = 1/252
            S0 = start_prices.get(key)
            mu = mean_return_annualy.get(key)
            sigma = sigma_pred_dict.get(key)
            st = Stock_price(S0,mu,sigma,dt)
            s.append(st)
```

```
In [ ]: """
        Simulate some fractional Brownian paths for different years
        """
        gbm_fbm_df = pd.DataFrame()
        fig, ((ax1, ax2, ax3), (ax4, ax5, ax6),(ax7, ax8, ax9),(ax10, ax11, ax12),
              (ax13, ax14, ax15),(ax16, ax17, ax18)) = plt.subplots(6, 3,figsize=(15,10))
        plt.subplots_adjust(wspace=0.3, hspace=0.7)
        h = 0.09800
        N = 252
        T = 252
        sample = 100
        c = 1
        for day in sorted(day_dict):
            S0 = start_prices.get(day)
            mu = mean_return_annualy.get(day)
            sigma0 = sigma_pred_dict.get(day)
            N = day_dict.get(day)
            gbm_fbm_df = pd.DataFrame()
            for i in range(sample):
                s, t, sig = GBM_fbm(S0,mu0,sigma0,h,N,T)
                gbm_fbm_df = pd.concat([gbm_fbm_df,pd.Series(s).rename(str(i))],axis=1)
            gbm_fbm_df.plot(color="blue",legend=False,ax = eval("ax"+str(c)), title=day)
            eval("ax"+str(c)).plot(gbm_fbm_df.mean(axis=1),"w")
            c+=1
        plt.savefig("gbmrv_sim.png",bbox_inches = 'tight')
```

```
In [ ]: """
        Sample paths from parameters from 2018
        """
        S0 = 908.88
        mu0 = -0.002191908442963353
        sigma0 = 0.1213797066759016
        h = 0.09800
        N = 252
        T = 252
        sample = 500
        gbm_fbm_df = pd.DataFrame()
        gbm_fbm_sigma_df = pd.DataFrame()
        for i in range(sample):
            s, t, sig = GBM_fbm(S0,mu0,sigma0,h,N,T)
            gbm_fbm_df = pd.concat([gbm_fbm_df,pd.Series(s).rename(str(i))],axis=1)
            gbm_fbm_sigma_df = pd.concat([gbm_fbm_sigma_df,pd.Series(sig).rename(str(i))],
                                         axis=1)
```

```
In [ ]: """
        Stock price calculation
        """
        dt=1/25
        price_df = pd.DataFrame()
        for col in gbm_fbm_df.columns:
            S0 = gbm_fbm_df.iloc[0:-1,int(col)]
            sigma0 = gbm_fbm_sigma_df.iloc[0,0]
            sigma = gbm_fbm_sigma_df.iloc[1:,int(col)]
            mu = logTransfer(gbm_fbm_df[col].values)
            s = Stock_price(S0,mu,sigma0,dt)
            price_df = pd.concat([price_df,pd.Series(s).rename(str(col))],axis = 1)
```

```
In [ ]: fig = plt.figure(figsize=(14, 8))
        df_last_m = df[(df.index > '2018-01-04') & (df.index <= '2019-01-04')]
        plt.plot(price_df, "b")
        plt.plot(price_df.mean(axis=1),"white", label = "Average price")
        plt.plot(df_last_m["close_price"].values, "red", label = "Actual price")
        plt.savefig("gbm_fbm_price.png")
        plt.legend()
        plt.show()
```

7 EURO to NOK currency

```
In [ ]: """
        Reading and cleaning data
        """
        data = feather.read_dataframe('EURNOKCurncy_10T.h5')
        data = data[data.trades != 0]
        data = data[~data.isin([np.nan, np.inf, -np.inf]).any(1)]
        data = data.reset_index()
        df = pd.DataFrame(data['std'])
        df = df[df['std'] != 0]
        df = df.reset_index()
        df['var'] = df['std']**2
        df['log_std'] = np.log(df['std'])
        plot_log_increments(5,df['var'])
```

```
In [ ]: """
        Plotting and prediction volatility
        """
        q, zeta = plot_zeta(df['log_std'])
        h = plot_q_zeta(zeta,q)
        rvdata = pd.DataFrame(df['var'])
        actual, predict, mse = prediction(rvdata,delta=1)
```

8 Neural Network

```
In [ ]: """
        Initiate network parameters
        """
        input_size = 3
        hidden_size = 7
        num_classes = 1
        num_epochs = 40000
        batch_size = 500
        learning_rate = 0.001
        logger = Logger('./logs')

        mu = 0.03
        sigma = 0.2
        N = 5000
        N2 = 200
        T = 1.
        steps = 2
        samples = 5000
        df_train = MakeTraindata(mu,sigma,N,steps,T,samples)
        df_test = MakeTestdata(mu,sigma,N2,steps,T,samples)
```

```
In [ ]: """
        Specify network structure and optimiser, loss function and learning rate
        """
        class Network(nn.Module):
            def __init__(self, input_size, hidden_size, num_classes):
                super().__init__()
                self.linear1 = nn.Linear(input_size, hidden_size)
                self.linear2 = nn.Linear(hidden_size,hidden_size)
                #self.linear3 = nn.Linear(hidden_size, num_classes)

            def forward(self, x):
                out = F.elu(self.linear1(x))
                #out = F.elu(self.linear2(out))
                out = self.linear2(out)
                return out

        model = Network(input_size, hidden_size, num_classes)
        criterion = nn.MSELoss(reduction='mean')
        optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
        logger = Logger('./logs')
```

```
In [ ]: """
        Training data with epoch
        """
        loss_list = []
        for epoch in range(num_epochs):
            x_b, pT_b = Batch(df_train, batch_size, shuffle=True)
            outputs = model(x_b)
            loss = criterion(pT_b, outputs)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            if epoch%batch_size == 0:
                print(epoch, loss.item())
                info = {'loss': loss.item()}

                for tag, value in info.items():
                    logger.scalar_summary(tag, value, epoch)

                for tag, value in model.named_parameters():
                    tag = tag.replace('.', '/')
                    logger.histo_summary(tag, value.data.cpu().numpy(), epoch+1)
                    logger.histo_summary(tag+'/grad', value.grad.data.cpu().numpy(),
                                         epoch+1)
```

```
In [ ]: """
        Make test data, and plot the test data
        """
        df_test = MakeTestdata(mu,sigma,N2,steps,T,samples)
        test_batch, pT = Batch(df_test,N2,shuffle=False)
        w1_list = np.linspace(0.,1.,N2)
        with torch.no_grad():
            model_reward = np.reshape(model(test_batch).numpy(),N2)
            plt.scatter(w1_list, pT)
            plt.plot(w1_list,model_reward,color = "r")
            plt.xlabel("Portfolio weight on stock")
            plt.savefig('gbm_fit1.png')
            plt.show()
```

Bibliography

[Algoet et al., 1988] Algoet, P. H., Cover, T. M., et al. (1988). Asymptotic optimality and asymptotic equipartition properties of log-optimum investment. *The Annals of Probability*, 16(2):876–898.

[Andersen et al., 2001] Andersen, T. G., Bollerslev, T., Diebold, F. X., and Ebens, H. (2001). The distribution of realized stock return volatility. *Journal of financial economics*, 61(1):43–76.

[Artzner et al., 1999] Artzner, P., Delbaen, F., Eber, J.-M., and Heath, D. (1999). Coherent measures of risk. *Mathematical finance*, 9(3):203–228.

[Benth, 2003] Benth, F. E. (2003). *Option theory with stochastic analysis: an introduction to mathematical finance*. Springer Science & Business Media.

[Berman, 1973] Berman, S. M. (1973). Local nondeterminism and local times of gaussian processes. *Bull. Amer. Math. Soc.*, 79(2):475–477.

[Bishop, 2006] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

[Black and Litterman, 1992] Black, F. and Litterman, R. (1992). Global portfolio optimization. *Financial Analysts Journal*, pages 28–43.

[Black and Scholes, 1973] Black, F. and Scholes, M. (1973). The pricing of options and corporate liabilities. *The Journal of Political Economy*, pages 637–654.

[Cont, 2001] Cont, R. (2001). Empirical properties of asset returns: stylized facts and statistical issues. *Quantitative Finance*, 1:223–236.

[Deng et al., 2009] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee.

[Devore and Berk, 2007] Devore, J. L. and Berk, K. N. (2007). *Modern mathematical statistics with applications*. Cengage Learning.

[Duffie and Pan, 1997] Duffie, D. and Pan, J. (1997). An overview of value at risk. *Journal of derivatives*, 4(3):7–49.

[Gatheral et al., 2014] Gatheral, J., Jaisson, T., and Rosenbaum, M. (2014). Volatility is rough. *arXiv preprint arXiv:1410.3394*.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

[Hagan et al., 1996] Hagan, M. T., Demuth, H. B., Beale, M. H., et al. (1996). *Neural network design*. Pws Pub. Boston.

[Han et al., 2016] Han, J. et al. (2016). Deep learning approximation for stochastic control problems. *arXiv preprint arXiv:1611.07422*.

[Hinton et al., 2012] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

[Hochreiter, 1998] Hochreiter, S. (1998). The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

[Jiang et al., 2017] Jiang, Z., Xu, D., and Liang, J. (2017). A deep reinforcement learning framework for the financial portfolio management problem. *arXiv preprint arXiv:1706.10059*.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

[Malkiel and Fama, 1970] Malkiel, B. G. and Fama, E. F. (1970). Efficient capital markets: A review of theory and empirical work. *The journal of Finance*, 25(2):383–417.

[Markowitz, 1952] Markowitz, H. (1952). Portfolio selection. *The journal of finance*, 7(1):77–91.

[McCulloch and Pitts, 1943] McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.

[Merton, 1969] Merton, R. C. (1969). Lifetime portfolio selection under uncertainty: The continuous-time case. *The review of Economics and Statistics*, 51(3):247–257.

[Mikosch, 1998] Mikosch, T. (1998). *Elementary stochastic calculus with finance in view.* World Scientific Pub Co Inc.

[Minsky and Papert, 2017] Minsky, M. and Papert, S. A. (2017). *Perceptrons: An introduction to computational geometry.* MIT press.

[Mitchell, 1997] Mitchell, T. M. (1997). *Machine learning. 1997*, volume 45.

[Nourdin, 2012] Nourdin, I. (2012). *Selected aspects of fractional Brownian motion*, volume 4. Springer.

[Rosenblatt, 1958] Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

[Rumelhart et al., 1986] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, 323(6088):533.

[Shevchenko, 2014] Shevchenko, G. (2014). Fractional brownian motion in a nutshell. *arXiv preprint arXiv:1406.1956*.

[Shreve, 2004] Shreve, S. E. (2004). *Stochastic Calculus for Finance Vol.2*. Springer.

[Smith, 2002] Smith, L. I. (2002). A tutorial on principal components analysis. Technical report.

[Sutton and Barto, 2018] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

[Walczak, 2001] Walczak, S. (2001). An empirical analysis of data requirements for financial forecasting with neural networks. *J. of Management Information Systems*, 17:203–222.

[Zhang et al., 1998] Zhang, G., Eddy Patuwo, B., and Y Hu, M. (1998). Forecasting with artificial neural networks:: The state of the art. *International journal of forecasting*, 14(1):35–62.