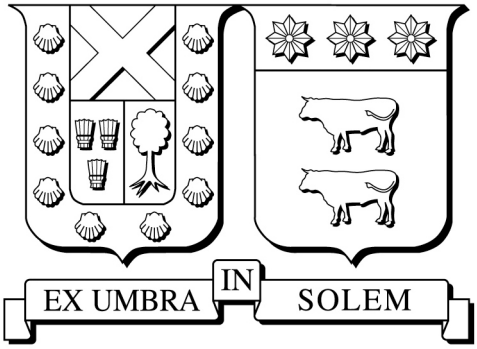
Universidad Técnica Federico Santa María
Departamento de Informática



Online learning methods for financial time series forecasting

DISSERTATION

Submitted in partial satisfaction of the requirements
for the degree of

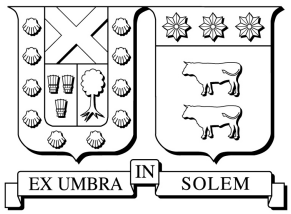
DOCTOR EN INGENIERÍA INFORMÁTICA

by

Paola Soledad Arce Azócar

Advisor
Luis Salinas Carrasco

In Valparaíso, Chile
June 2016.



Examining Committee

Prof. Dr. Luis Salinas
Universidad Técnica Federico Santa María
Chile

Advisor

Prof. Dr.
Universidad Técnica Federico Santa María
Chile

Committee Member

Prof. Dr.
University
Chile

External Peer Reviewer

To my beloved family

Abstract

Financial time series are known for their non-stationary behaviour and this has motivated its study using different techniques. One relevant observation is that sometimes time series exhibit some stationary linear combinations. When this happens, it is said that those time series are cointegrated. Cointegration has been then one of the main features studied. Vector error correction model (VECM) is an econometric model which characterises the joint dynamic behaviour of a set of cointegrated variables in terms of forces pulling towards equilibrium.

VECM parameters are obtained using the ordinary least squares (OLS) method. Even though OLS is extensively used, it has over-fitting issues and is computationally expensive. Ridge regression is commonly used instead of OLS since they include a regularisation parameter which could improve prediction error.

In this thesis, financial time series features are studied and different algorithms were developed in order to optimise parameters and increase performance. In particular, I propose an online algorithm based on VECM which optimises how model parameters are obtained and reduces execution times. This is achieved considering only a sliding window of the last historical data and using machine learning techniques to solve the model. Moreover, the long-run relationship between the time series is used in order to make optimisations and obtain improved execution times. Due to the large amount of financial data available and the need of quick response, the algorithm presented was optimised using high performance computing. Our experiments were tested using synthetic data and foreign exchange rates. Results show that cointegration and high performance computing allow to obtain models with better performance accuracy and reduced execution times.

Keywords: *Computational finance - Cointegration - Time Series - Online algorithms - Regression*

Contents

List of Tables

List of Figures

Chapter 0

Introduction

"An individual economic variable, views as a time series, can wander extensively and yet some pairs of series may be expected to move so that they do not drift far apart."-Robert F. Engle and Clive W.J. Granger [**EG87b**].
In this introduction we present the scope, objectives, hypothesis and organization of this thesis.

0.1. Scope of this research

The stochastic behaviour of financial time series, its incrementing amount of data available and the need of performing accurate forecasting in short periods of time has motivated researchers to create efficient and fast algorithms. This study involves interdisciplinary knowledge such as: finance, scientific computing, high performance computing, machine learning among others.

Forecasting financial time series have been modelled using classical statistical approaches. More recently, machine learning models have been extensively used in forecasting. However, their main disadvantage is that getting model parameters is a computational challenge. The computational complexity of machine learning algorithms has become a limiting factor for problems that require processing large volumes of data and where response time is crucial.

Therefore, algorithms that process large amount of data in a short periods of time are required. Recently, online learning algorithms have been developed to solve large-scale problems since they process an instance at a time, updating the model at each step incrementally. This is opposed to the batch algorithms where the forecast model is built using a large collection of historical data in a training phase.

The specific scope of this study is to use financial time series features in order to design a forecasting algorithm which ensures accuracy and low response times. Cointegration is the main feature studied and it refers that one or more linear combinations of these time series are stationary even though individually they are not [**EG87a**]. Some models, such as the Vector Error Correction Model (VECM), take advantage of this property and describe the joint behaviour of several cointegrated variables.

In this thesis, an online formulation of the VECM called Online VECM (OVECM) is proposed. OVECM is based on the consideration of a sliding window of the most recent data. The algorithm introduces matrix optimisations in order to reduce the number of operations and also takes into account the fact that cointegration vector space doesn't experience large changes with small changes in the input data. Moreover, VECM parameters are obtained using machine learning methods. Our method is later tested using four currency rates from the foreign exchange market with different frequencies. On the other hand, in order to improve VECM parameters, an adaptive VECM algorithm is presented called AVECM. AVECM allows VECM parameters to be found by maximising the number of cointegration relations for a given set of parameters on a grid search. This grid search is done in parallel.

Models effectiveness were focused on the out-of-sample forecast rather than on the in-sample fitting. This criteria allows the OVECM and AVECM prediction capability to be expressed rather than just explaining data history. Our method performance is compared with ARIMA which is the most widely used algorithm for modelling a multivariate time series.

0.2. Research Objectives

The main motivation for this research is the development of efficient methods for forecasting financial time series.

The specific objectives of this research are,

- $\mathcal{O}_1$: *A review of the literature on time series analysis models including machine learning techniques.*
- $\mathcal{O}_2$: *Development of a set of known features of the studied time series and applied them to improve forecasting.*
- $\mathcal{O}_3$: *Development of parallel and efficient algorithms to ensure quick response times .*
- $\mathcal{O}_4$: *Deep mathematical analysis of the proposal and financial concepts involved.*
- $\mathcal{O}_5$: *Design and implement representative set of experiments in order to show when and why proposal performs better.*

0.3. Research Hypothesis

Cointegration concept was introduced by Engle and Granger in 1987 [**EG87b**] and implies that one or more linear combinations of non-stationary variables are stationary even though individually they are not. Moreover Stock and Watson in 1988 [**SW88**] observed that cointegration reflects the common stochastic trends providing a useful way to understand cointegration relationships. These common stochastic trends can be also interpreted as a long-run equilibrium relationships.

Vector error correction model (VECM) introduces this long-run relationship among a set of cointegrated variables as an error correction term. VECM is a special case of the vector autoregresive model (VAR) model. VAR model expresses future values as a linear combination of variables past values. However, VAR model cannot be used with non-stationary variables. VECM is a linear model but in terms of variable differences. If cointegration exists, variable differences are stationary and they introduce an error correction term which adjusts coefficients to bring the variables back to equilibrium. In finance, many economic time series turn to be stationary when they are differentiated and cointegration restrictions often improves forecasting [**DT98**]. Therefore, VECM has been widely adopted.

Both VECM and VAR model parameters are obtained using ordinary least squares (OLS) method. OLS has two main problems: is sensitive to errors on input data and involves many calculations. The former problem is commonly solved using Ridge Regression (RR) [**HK70**] which introduces a regularization parameter that leads to an unbiased estimation with better generalisation capability. The second problem of computational complexity depends on the number of past values and observations considered. Recently, online learning algorithms have been proposed to solve problems with large data sets because of their simplicity and their ability to update the model when new data is available.

The main research hypothesis explored in this dissertation is the following:

An online learning algorithms based on cointegration and high performance computing will allow faster forecasting algorithms for financial time series to be obtained while maintaining good accuracy levels.

0.4. Organisation of this Thesis

Chapter 1 contains relevant finance concepts required to understand financial time series models such as market hypothesis, frequency, order book generation, market microstructure. Chapter 2 discuss main time series models including classic ones such as ARMA, GARCH and more recent ones such as VECM, VAR and volatility models. Chapter 3 gives a brief introduction to machine learning and its variant online learning.

Chapter 4 presents different algorithms propose to forecast financial time series using machine learning techniques. In Chapter 5 a deep study of foreign exchange market features is presented and experiments of cointegration are carried out in order to show its dynamic. Chapter 6 presents a discussion of found results and summarise the main conclusions of this thesis. It also presents some research directions for future studied.

Chapter 1

High Frequency Trading

High frequency trading (HFT) strategies are based on buy and sell assets in short periods of time gaining a small profit in every transaction despite the transaction costs. The key is the amount of transactions that HFT algorithms are capable to execute. In 2012 more than 50% of US market trades were HFT. HFT has motivated computer-driven strategies capable of processing large amount of data in short periods of time.

1.1. High Frequency Trading Definition

HFT is not a strategy but a technology that implements different trading strategies. The aim of HFT is to benefit from market short-term pricing inefficiencies [**CSKM11**]. HFT is characterised for the use of high-speed and sophisticated quantitative and algorithmic computer applications for modelling and executing orders efficiently. In order to make fast decisions, HFT firms require speed access to trading servers and sometimes they are physically near so they can minimise network latencies.

High frequency trades are short-term positions that commonly end the trading day avoiding leaving opened positions to the next business day. HFT is frequently associated to algorithm trading strategies, but the former is focused in to reduce the market impact of large institutional positions and the later refers to trade execution strategies that are typically used by fund managers to buy or sell large amounts of assets. The duration of the positions in HFT it is not well-defined. Figure 1.1 shows a survey done to traders about the holding time of a position opened. Most of them agreed that HFT refers to positions between 1-second and 10-minutes. Overnight positions are avoided since they are riskier and also have more expensive fees, HFT firms end the trading day closing all opened positions.

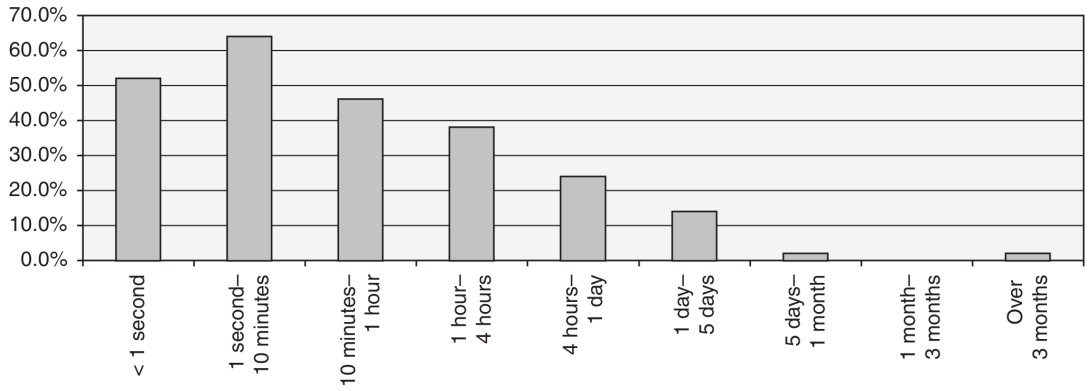


FIGURE 1.1. Holding time of an opened position of a high frequency trade.

In 2014 HFT represented nearly 50% of equity trades in the US and more than 20% in Europe showing a consistent fall since 2009 which was its best year. US market revenues have fallen dramatically. Figure 1.2 on the left shows HFT trades percentage of US and Europe equity shares. The right figure shows how revenues in US stocks have fallen the last years.

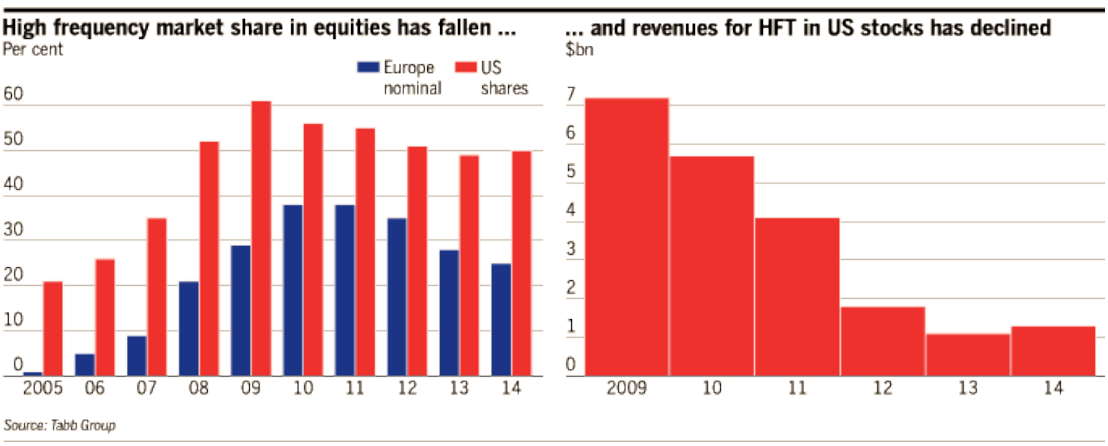


FIGURE 1.2. Left figure shows HFT market share in the US and Europe. Right figure shows revenue in the US.

One of the reasons of this fall is that exchange markets have adapted, having now faster, more transparent and efficient market structures than before. This has been possible due to its investment in technology enhancing reliability and stability of transactions. Even though this fall, HFT is still a major component of regulated markets and will probably remain as a topic of interest for researchers in the near future.

On the other hand, HFT has been criticised on qualitative issues concerning fairness and systemic risk. However, empirical research shows that HFT has lead to beneficial

impacts such as reducing spreads (difference between buyers and sellers prices), increased liquidity, allowing more efficient price formation, reduced transaction costs and lower market volatility. HFT makes the stock market more efficient and helps small investors who trade at random times over the day.

1.2. Financial Markets

A financial market is any marketplace where buyers and sellers participate trading different assets such as equities, bonds, currencies and derivatives (future or options). One of the main objectives of financial markets is to set prices for global trade. A financial market has many components but the most commonly used are money markets and capital markets. Money markets are used for short-term basis, usually for assets up to one year, for greater periods capital markets are used. Capital markets include the stock or equity market and the bond or debt market and their movements are the most widely followed.

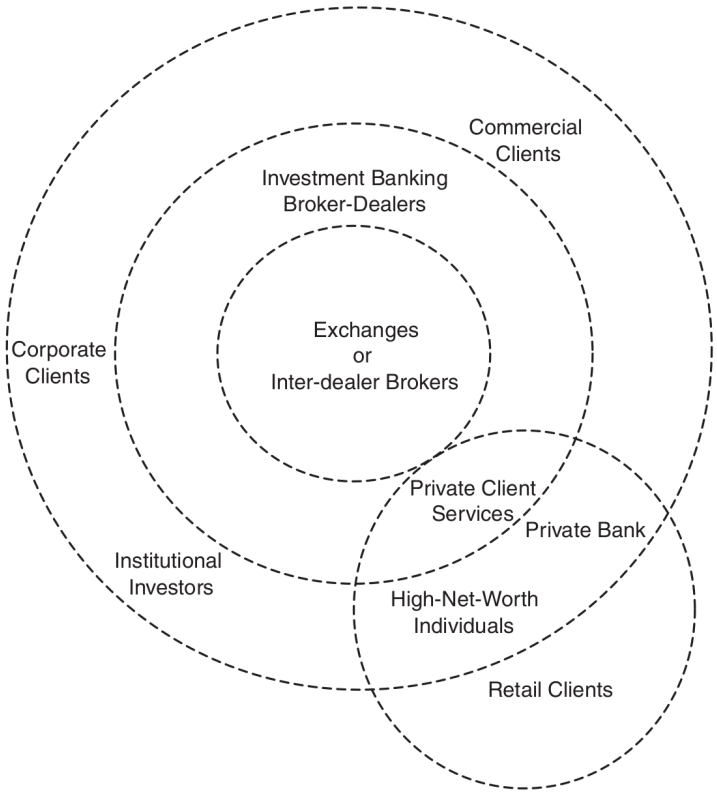


Figure 1.3. Old structure of capital markets.

Figure 1.3 shows the typical structure of capital markets existed from the early 1929s through much of the 1990s where the broker-dealers played the central and most profitable role. At the core are the exchanges or inter-dealer networks (foreign exchange trading). Exchanges are the centralised marketplaces for transacting. Broker-dealers perform two functions: trading for their own accounts and transacting for their customers. Broker-dealers use inter-dealer brokers to quickly find the best price for a particular asset among the network of other broker-dealers. Occasionally, broker-dealers also deal directly with other broker-dealers, particularly for less liquid instruments. Broker-dealers clients are institutional investors, corporate clients, commercial clients, and high-net-worth individuals.

This centralised structure existed until computer technology allowed a better communication structure. Today financial markets are more decentralised providing more liquidity. Exchanges and inter-dealer brokers were replaced by liquidity pools or Electronic Communication Networks (ECNs) which are able to transmit order quickly matching buyers and sellers optimally. There are also dark liquidity pools where trader identity and orders remain anonymous.

Figure 1.4 shows current structure of capital markets including ECNs and dark pools. In this structure, ECNs, Exchanges, dark pools, broker-dealers and retail brokerages can execute orders. However, there are some institutional clients that have also become broke-dealers.

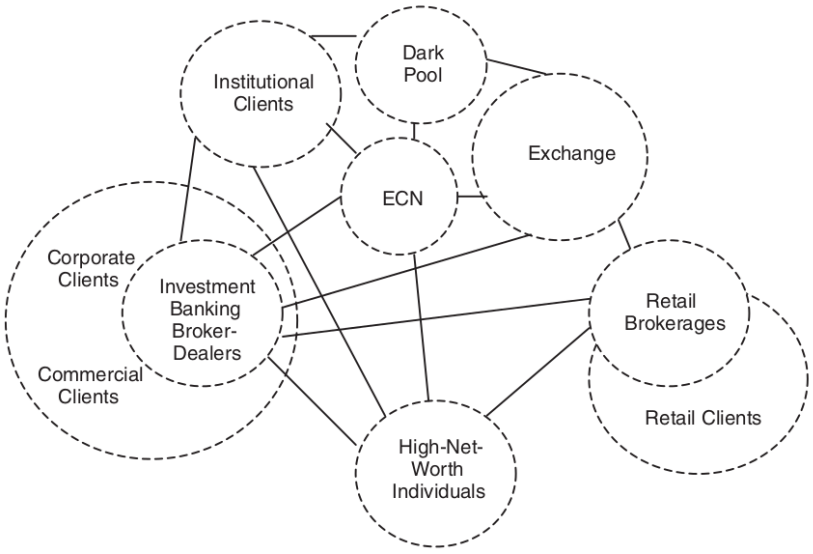


FIGURE 1.4. Actual structure of capital markets.

Equity market and foreign exchange market (Forex) are the most popular markets for high frequency trading strategies. In the Equity market, stocks such as futures and options, exchange-traded funds (ETFs) among others financial instruments can be

traded. Additionally, in the Forex market, interest rates denominated pair currencies such as EURUSD (Euro to UD dollar) or USDJPY (US dollar to Japanese yen) are traded. Traders of this market are diverse, some of them are high frequency traders, long-term investors and corporations. The Forex market used to be centralised, only commercial banks had exclusive access to inter-dealer networks. Today, Forex market is decentralised and has become the biggest market in terms of volume of trading. The main participants are international banks geographically dispersed with continuous 24 hours operations excepting weekends. Figure 1.5 shows different market trading hours in GMT for London, New York, Sydney and Tokyo.

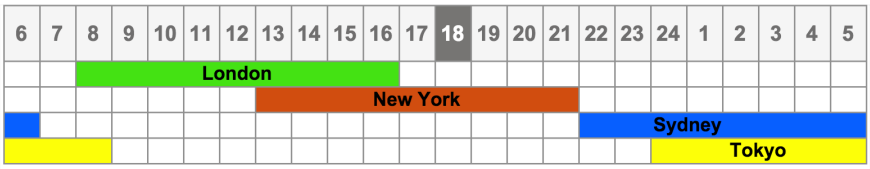


FIGURE 1.5. Forex market trading hours (GMT).

There are three periods of overlaps between different trading times:

- New York and London from 13:00 GMT until 17:00 GMT
- Tokyo and London from 8:00 GMT until 9:00 GMT
- Sydney and Tokyo from 23:00 GMT to 7:00 GMT

These overlap hours are very important since the highest volume of trades are expected. Therefore, most HFT strategies are executed in these hours.

1.3. Price formation process

A trade consists at least of two orders: one to enter the market (buy order) and exit the market (sell order). Traders have different types of order they can execute:

Market order: allows to buy or sell an asset at the current best price whatever the price is. The transaction price is determined by the market considering previous executed orders and volume requested.

Limit order: allow to specify the price you want to buy or sell an asset. Depending on the market conditions, this type of orders could never been executed.

Stop order: are suitable for investors who are unable to monitor their investments for a period of time. Stop order allows to specify the price that a position should be closed called stop price. When the stop price is reached a market order is executed, this means that market price could not be exactly the same specified in the stop order.

In terms of commissions, limit and stop order are more expensive than market orders and there is no certainty of execution.

Moreover, all orders can specify other parameters such as:

Fill or Kill: is an order that must be executed immediately or being cancelled, no partial fulfillments are allowed.

Day: the order is only valid during the day.

Good til canceled: a order is active until the investor decides to cancel it or the trade is executed.

In order to determine the execution price, buyers and sellers orders are placed in an order book which help to determine which order can be fulfilled. Figure 1.6 illustrates how buyers and sellers are ordered. The ask or offer price is the current lower price a seller is willing to accept for a good. The bid price corresponds to the current highest price a buyer is willing to pay for a good. Orders with the same price are prioritised by arrival time and placed on top of the book. The difference between current ask and bid price is called spread and their average is called mid-price.

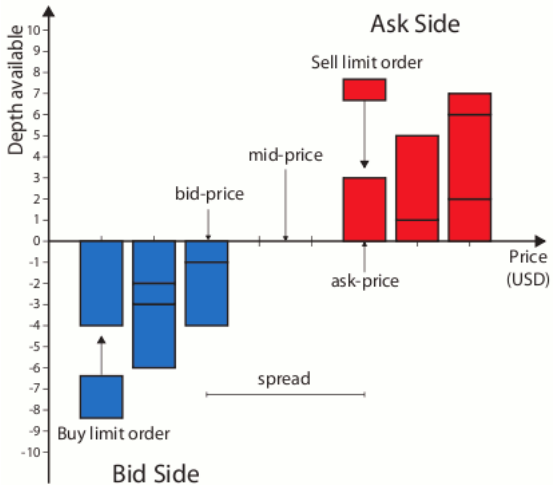


FIGURE 1.6. Order book. Buyers and sellers are ordered according to the bid or ask price and the market determines the mid-price and transacted volume.

Today, order books are available and they are a popular source of study among researchers. The mid-price and spread modelling are some of the problems related with this area.

1.4. EFFICIENT MARKET HYPOTHESIS

Efficient Market Hypothesis (EMH) was developed independently by Paul Samuelson and Eugene Fama in the 1960s. EMH also known as the random walk theory states that current stock prices fully reflect available information related to its value and there is no way to earn excess profits [**Fam70**]. EMH requires that all the participants have rational expectations, and investors reactions be random and follow a normal distribution pattern. Thus, any one can be wrong about the market, but the market is always right as a whole. There are three common forms of EMH: weak-form efficiency, semi-strong efficiency and strong-form efficiency.

The weak form claims that prices already reflect all past publicly available information. Therefore, future prices cannot be predicted based on analysis of historical data. This implies that future prices movements are determined entirely by information not contained in the past prices and participants are unable to systematically profit from market inefficiencies. However, many studies have shown a marked tendency for the stock markets to trend over time periods. Various explanations for such large and apparently non-random price movements have been promulgated. Even Fama has accepted price anomalies which do not follow the weak-form efficiency hypothesis [**FF08**].

The semi-strong form of the EMH claims that prices instantly change to reflect new public (not private) information such that no excess return can be obtained.

The strong form of the EMH additionally claims that prices instantly reflect even hidden, private or "insider" information.

EMH is related with two approaches to investment analysis: fundamental and technical analysis. Fundamental analysts base their predictions of stock price behaviour on fundamental factors such as internal information of a company, its industry or the economy. Technical analysts, by contrast, consider that all this financial information is already included in the prices and believe that future stock prices can be predicted studying the historical market behaviour. A market technician bases its predictions on historical patterns of prices of volume changes.

Under the weak form of EMH, strategies based on technical analysis will not be able to produce excess returns. However, the weak form accepts that some forms of fundamental analysis may still provide excess return. Similarly, the semi-strong form of EMH neither technical or fundamental analysis can produce excess return.

If the stock market efficiently digests all available information, there is little justification for seeking excess returns gains from investing. However, EMH doesn't lessen the importance of investing only change its philosophy. The only way an investor can possibly obtain higher returns is by purchasing riskier investments.

Researches can determine now how efficient a financial market is, i.e how efficiently information is processed.

EMH is based on rational human behaviour and its validity has been criticised by psychologists and behavioural economists who argue that the EMH is based on counterfactual assumptions regarding human behaviour, that is, rationality. Recent advances in evolutionary psychology and the cognitive neurosciences may be able to reconcile the

EMH with behavioural anomalies. On the other hand, if the EMH is true, the market really walks randomly and therefore there shouldn't be any difference between experienced and novice traders. Kim Man Lui proved the contrary in a controlled experiment [**mLC13**].

Chapter 2

Financial Time Series

A time series is a collection of observations in time (discrete or continuous). The analysis of time series main objective is to find possible internal structure in the data such as autocorrelation, trend or seasonal variation. Some of application and uses of time series analysis are data compression, explanatory variables (relationships with other variables, seasonal factors, etc.), signal processing and forecasting (predict future values), which is the focus of this thesis. There are two main approaches to study financial time series: to study directions of financial rates and to explain financial rate volatility. This chapter reviews the most relevant techniques in the rich and rapidly growing field of time series analysis considering these two approaches.

2.1. Characteristics of Financial Time Series

There are several characteristics and concepts of financial time series and here we will discuss the more important ones.

2.1.1. Stylized facts of asset returns. There are several known features exhibited by financial instruments called stylized facts, which have been empirically studied and some of them have been documented only recently and accepted as truth. Stylized facts are usually formulated in terms of qualitative properties of asset returns and may not be precise enough to distinguish among different parametric models. [**Con01**].

Some stylized facts which are common to a wide set of financial assets [**Sew11**] are:

Dependence: autocorrelation function (ACF) in returns is largely insignificant. Returns of a time series y_t is defined as $r_t = y_t - y_{t-1}$. The ACF measures the linear predictability of the time series y_t at time t using only the value at time

s. ACF is defined as:

$$\rho(s,t) = \frac{\gamma(s,t)}{\sqrt{\gamma(s,s)\gamma(t,t)}}$$

where $\gamma(s,t)$ is the auto-covariance function that measures the linear dependence between two points on the same series observed at different times. It is defined as the second moment product:

$$\gamma_y(s,t) = E[(y_s - \mu_s)(y_t - \mu_t)]$$

The ACF in the absolute and squared returns is always positive, significant and decays slowly. In addition, the ACF in the absolute returns is generally higher than the ACF in the corresponding squared returns.

Figure 2.1 shows the ACF of the returns of SPY stock where the correlations are significant even for very long lags, this implies a long-memory process.

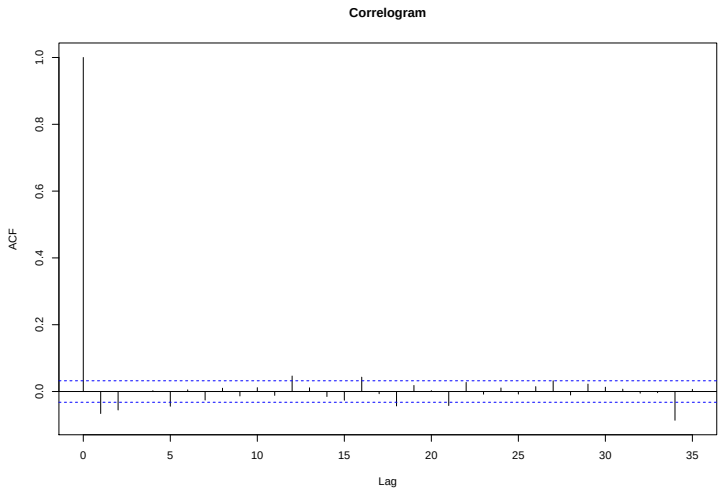


FIGURE 2.1. SPY returns ACF

Distribution: The distribution of returns is approximately symmetric and has high kurtosis (i.e fat tails and a peaked centre compared with the normal distribution). However, distribution of returns whose were obtained from higher frequencies looks more like a normal distribution. The returns distribution tails are larger than what is hypothesised by common data generation process (generally normal distribution assumption). In the markets, fat tails are an undesirable feature because of the additional risk they imply. In the figure 2.2 is shown the SPY returns distribution based on daily dates from the period 1st July 1998 to 4th April 2013. The distribution was compared against the normal distribution which clearly doesn't fit the data.

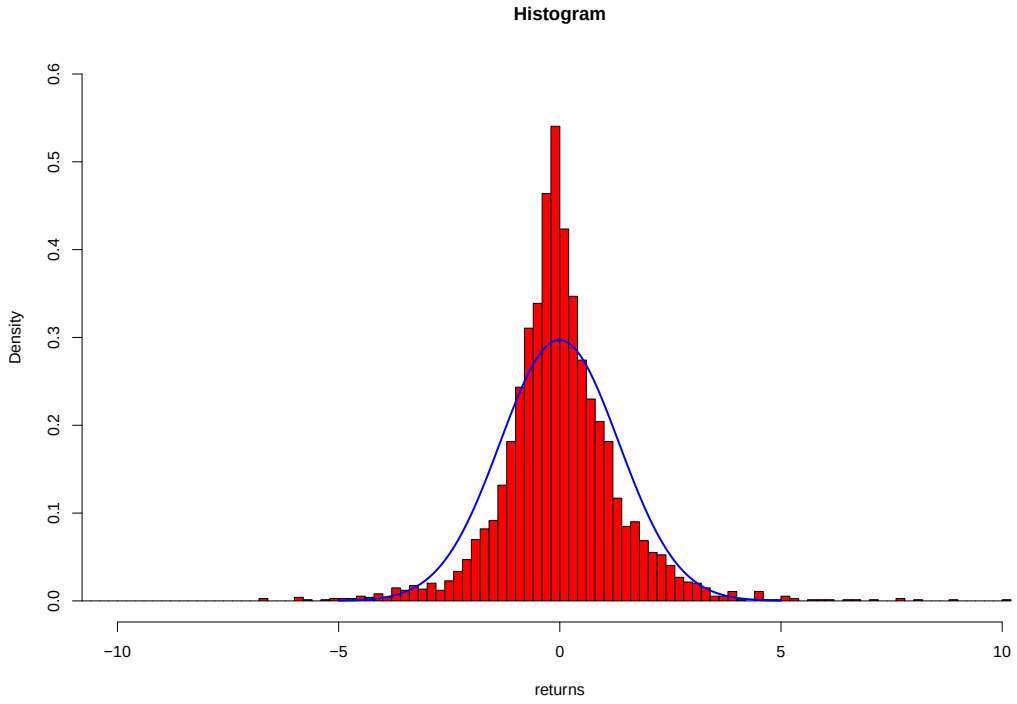


FIGURE 2.2. SPY returns distribution

Heterogeneity: despite the fact that financial returns are non-stationary, stationary periods can be observed. Economist speaks in terms of a structural break [**Sto94**], in machine learning this is known as drift which means that the statistical properties of the target variable change over time [**WK96**], [**Tsy04**].

Non-Linearity: financial returns may be non-linear in mean and/or non-linear in variance.

Calendar effects: are also called seasonal effects and they are cyclical anomalies in returns where the cycle is based on the calendar. Some of known calendar effects are: intraday effect, weekend effect, Monday effect, intramonth effect, the January effect and Holiday effect. The most important calendar anomalies are the January effect and the weekend effect.

2.1.2. Stationary. A strictly stationary times series y_t is one for which the probabilistic behaviour of every collection of values $\{y_{t_1}, y_{t_2}, \ldots, y_{t_L}\}$ is identical to that of the time shifted set, more precisely:

$$P\{y_{t_1} \leq c_1, \ldots, y_{t_L} \leq c_L\} = P\{y_{t_1+h} \leq c_1, \ldots, y_{t_L+h} \leq c_L\} \quad \forall L \in \mathbb{N}, \forall h \in \mathbb{Z}$$

where $c_1, \ldots, c_L$ are constants. This definition is too strong and difficult to assess it from a single data set. The weak version of this definition imposes conditions only on the two first two moments.

A weakly stationary time series is a process which mean, variance and auto covariance do not change over time:

$$\begin{aligned} E(y_t) &= \mu \quad \forall t \in \mathbb{N} \\ E(y_t^2) &= \sigma^2 \quad \forall t \in \mathbb{N} \\ \lambda(s,t) &= \lambda(s+h, t+h) \quad \forall s,t \in \mathbb{N}, \forall h \in \mathbb{Z} \end{aligned}$$

with $\lambda(s,t) = E[(y_s - \mu)(y_t - \mu)]$

In finance, a shock represents an unexpected change in a variable or a in its error term in a particular time period. For stationary time series, shocks to the system will gradually die away. That is, a shock during time t will have a smaller effect in time $t+1$, a smaller effect on $t+2$ and so on. For non-stationary data, the persistence of shocks will always be infinite.

2.1.3. Non-stationary processes. There are different types of non-stationary time series models often found in economics:

(a) **Deterministic trend:** Deterministic trend or trend stationary processes have the following form:

$$y_t = f(t) + \epsilon_t \quad ,$$

where t is the time trend and ϵ_t represents a stationary error term (with mean 0 and variance σ^2) and $f(t)$ is a deterministic function of time:

- If $f(t) = \alpha + \beta t$ we have a linear trend model which is widely used.
- If $f(t) = \alpha \exp^r t$ we have an exponential growth curve.
- If $f(t) = c_1 + c_2 t + c_3 t^2$ we have a quadratic trend model
- If $f(t) = \frac{1}{k + \alpha \beta^t}$ we have a logistic curve

(b) **Stochastic trend:** Stochastic trend processes are also called unit root or difference stationarity processes and have the following form:

$$y_t = \mu + y_{t-1} + \epsilon_t$$

where ϵ_t is a stationary process. When $\mu = 0$ the process is called pure random walk and when $\mu \neq 0$ the process is called random walk with drift.

Alternatively this process can be expressed using the lag operator L such as:

$$(1 - L) y_t = \mu + \epsilon_t$$

This process is also called unit root because the root of the characteristic equation $(1 - z = 0)$ is the unity.

A random walk with or without drift can be transformed to a stationary process by differencing the time serie once. The disadvantage of differencing is that the process loses one observation each time the time series is differentiated.

Apart from a stochastic trend, many economic financial time series seem to involve an exponential trend, this is the reason why researchers often take the logarithmic transformation before doing analysis.

Other less common forms of non-stationarity are structural break in mean and structural break in variance.

2.1.4. Spurious regression. The use of non-stationary data can lead to spurious regressions. Spurious regression dates back to Yule in 1926 [**Yul26**]. If two stationary variables are generated as independent random series and are trending over time, when one of those variables is regressed on the other, they could have a high R^2 even if the two are totally unrelated. So, if standard regression techniques are applied to non-stationary data, it could look good under standard measures but valueless [**Bro02**].

Example of spurious regression An example is shown below. Two random walks series $x_{1,t}$ and $x_{2,t}$ with a small drift ($\alpha = 0.09$) are generated and then regressed one on the other:

$$x_{i,t} = \alpha + x_{i,t-1} + \epsilon_i, t \qquad \text{where} \qquad \mathcal{N}(0,1), \quad i = 1,2 \tag{2.1}$$

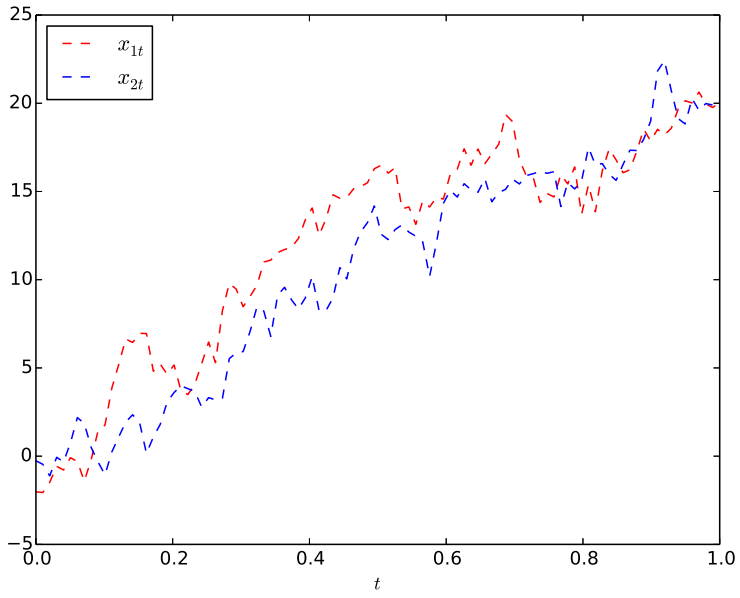


FIGURE 2.3. Two random walks time series $x_{1,t}$ and $x_{2,t}$

These two random walks, which are independent by construction, appear to be related with a $R^2 = 0.883$. However, what it is happening is that they are drifting in the

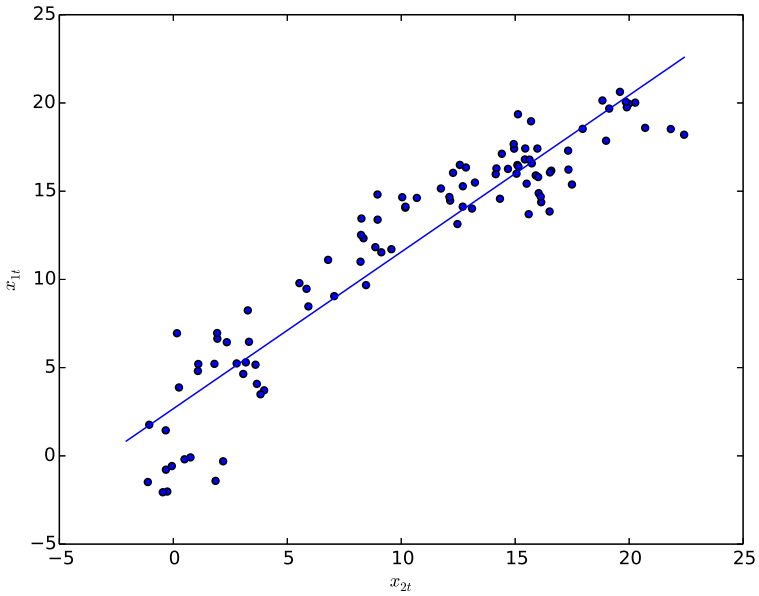


FIGURE 2.4. Regression between two random walks time series

same direction. Drift can be removed using returns or first differences. If after taking returns or first differences the regression fit is still good we will say that the variables are cointegrated, this means that both series may drift but their residuals will not. A more formal definition of cointegration is given in the section 2.1.6.

2.1.5. Integration. Following Johansen [**Joh95**] we shall say that a stochastic process Y_t which satisfies $Y_t - E(Y_t) = \sum_{i=0}^{\infty} C_i\, \varepsilon_{t-i}$ is called $I(0)$, and then we shall write $Y_t \sim I(0)$, whenever $\sum_{i=0}^{\infty} C_i \neq 0$ and $\sum_{i=0}^{\infty} C_i\, z^i$ converges for $z \in \mathbb{C}$ with $|z| < 1$. It is understood that the condition $\varepsilon_t \sim iid(0, \sigma^2)$ holds.

A (vector) time series $\mathbf{y}_t$ is said to be *integrated of order* d, and then we shall write $\mathbf{y}_t \sim I(d)$, whenever after d times (discrete) differentiation a stationary process is obtained [**Ban93**]; more precisely, whenever $(1-L)^d\, \mathbf{y}_t \sim \mathrm{I}(0)$, where L is the usual lag operator: $(1-L)\, \mathbf{y}_t = \Delta \mathbf{y}_t = \mathbf{y}_t - \mathbf{y}_{t-1}$ for all t.

Note that this definition includes the scalar case as time series of vectors of dimension 1; in this scalar case we will write the time series in non-bold format.

2.1.6. Cointegration. Cointegration concept was introduced by Engle in 1987 [**EG87b**] and implies that one or more linear combinations of non-stationary variables are stationary even though individually they are not.

Let $\mathbf{y}_t^\nu$, $\nu = 1, \ldots, p$, be a set of p vector time series of order $I(1)$. They are said to be *cointegrated* if a vector $\beta = [\beta(1), \ldots, \beta(p)]^\top \in \mathbb{R}^p$ exists, such that the time series,

$$\mathbf{Z}_t := \sum_{\nu=1}^{p} \beta(\nu)\, \mathbf{y}_t^\nu \sim \mathrm{I}(0)\,. \tag{2.2}$$

In other words, a set of $I(1)$ time series is said to be cointegrated if a linear combination of them exists, which is I(0).

Stock and Watson in 1988 [**SW88**] observed that cointegration reflects the common stochastic trends providing a useful way to understand cointegration relationships. These common stochastic trends can be also interpreted as a long-run equilibrium relationship.

The idea of cointegration was immediately adopted in finance since it could represent their long-run relationship implied by economic theory [**LL91**], [**LF05**]. Economic theory suggest that economic time series are mean-reverting process and therefore, it reflects the idea of that some set of variables cannot wander too far from each other.

On the other hand, the efficient markets hypothesis, also known as the random walk theory states that current stock prices fully reflect available information related to its value and there is no way to earn excess profits [**Fam70**]. This means that if we have stock prices from a jointly efficient market, they cannot be cointegrated [**Gra86**], [**DJW92**]. However, [**Ric95**] claims that cointegration is directly at odds with market efficiency, even though, there is no evidence that cointegration among asset prices have implications about market efficiency [**LF05**].

Despite the fact that cointegration on closing daily rates of currency pairs has not been found [**Col90**], [**Cop91**], different time series frequencies can have different behaviours [**Ald09**]. Pair trading is a very common example of cointegration application [**Her03**] but cointegration can also be extended to a larger set of variables [**MN95**],[**EP04**].

2.1.7. Johansen method. Johansen in 1988 [**Joh88**] suggests a method for determining cointegration vectors.

There are two statistics for cointegration under Johansen approach: the trace (shown in equation 2.3) and the maximum-eigenvalue statistic (shown in equation 2.4):

$$\lambda_{\mathrm{trace}}(r) = -T \sum_{i=r+1}^{g} \ln(1 - \hat{\lambda}_i) \tag{2.3}$$

$$\lambda_{\max}(r, r+1) = -T \ln(1 - \hat{\lambda}_{r+1}) \tag{2.4}$$

where r is the number of cointegration vectors and $\hat{]}\lambda_i$ is the estimated value for the ith ordered eigenvalue. λ_{trace} null hypothesis is that the number of cointegration vectors is less than or equal to r against that there are more than r. $\lambda_{\max}$ has the null hypothesis that the number of cointegration vectors is r against an alternative of $r + 1$.

Cointegration example

If we have two-dimensional process $\mathbf{y}_t$, $t = 1, \ldots, T$ by:

$$\begin{aligned} \mathbf{y}_{1t} &= \sum_{i=1}^{t} \epsilon_{1i} + \epsilon_{2t} \\ \mathbf{y}_{2t} &= a \sum_{i=1}^{t} \epsilon_{1i} + \epsilon_{3t} \end{aligned}$$

Since $\mathbf{y}_{1t}$ and $\mathbf{y}_{2t}$ are I(1) processes and there exist a vector $\beta = [a \; -1]$ such that:

$$\beta^\top \mathbf{y}_t = a\mathbf{y}_{1t} - \mathbf{y}_{2t} = a\epsilon_{2t} - \epsilon_{3t} \sim \text{I}(0)$$

then, both processes are said to be cointegrated.

If we add a I(0) process $\mathbf{y}_{3t} = \epsilon_{4t}$ we find that there exists two cointegration vectors now: $\begin{bmatrix} a & -1 & 0 \end{bmatrix}$ and $\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$ since:

$$\beta^\top \mathbf{y}_t = \begin{bmatrix} a & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{y}_{1t} \\ \mathbf{y}_{2t} \\ \mathbf{y}_{3t} \end{bmatrix} = \begin{bmatrix} a\epsilon_{2t} - \epsilon_{3t} \\ \epsilon_{4t} \end{bmatrix}$$

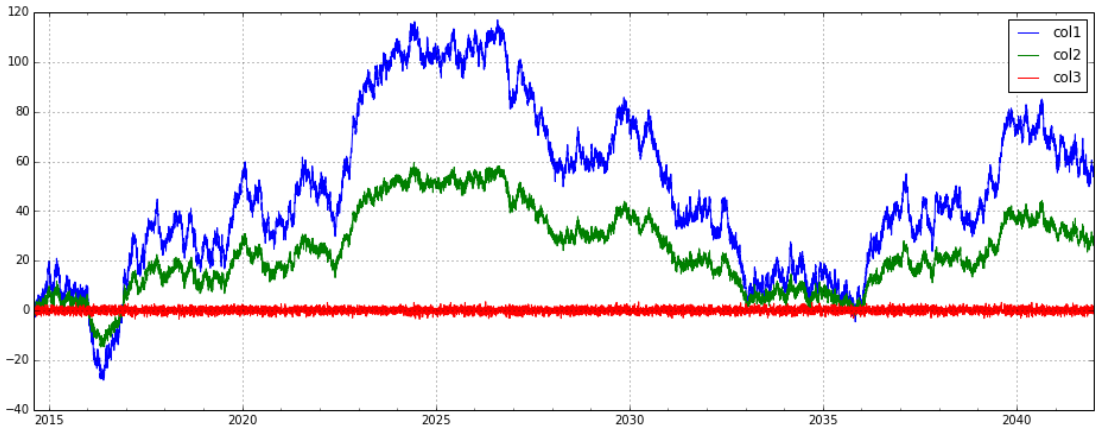


FIGURE 2.5. Cointegration example: two I(1) processes (blue and green) and one I(0) process (red).

This example shows how cointegration vectors describes the stable relations between the processes by linear relations that are more stationary than the original process. Cointegration vectors can be obtained using the Johansen method which gives with a certain probability the number of significant vectors. Figures 2.6, 2.7 and 2.8 shows the linear combinations obtained using cointegration vectors given by Johansen method. The method confirm that only two vectors are found and the third one given doesn't result in a I(0) process.

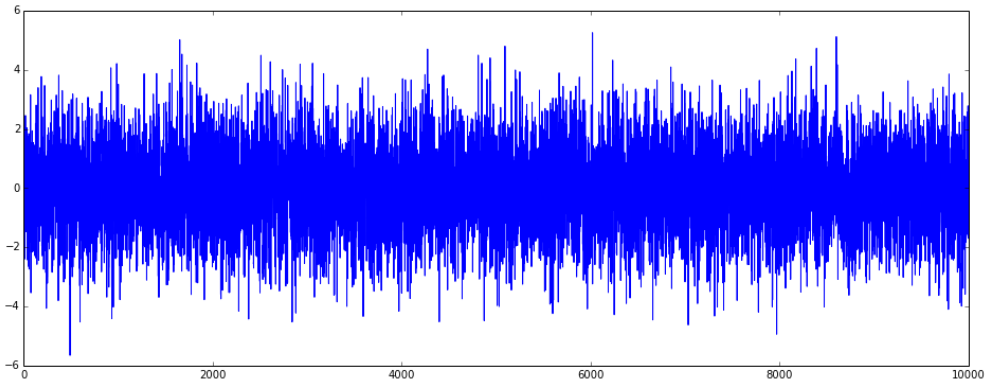


FIGURE 2.6. Time series linear combination using the first cointegration vector

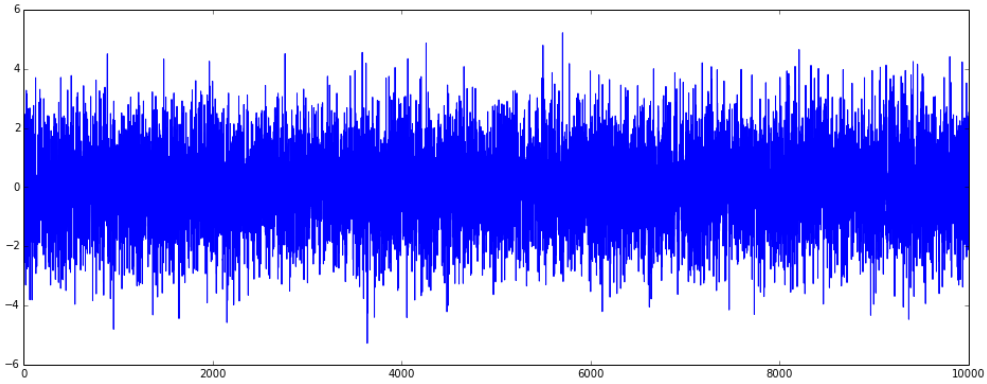


FIGURE 2.7. Time series linear combination using the second cointegration vector

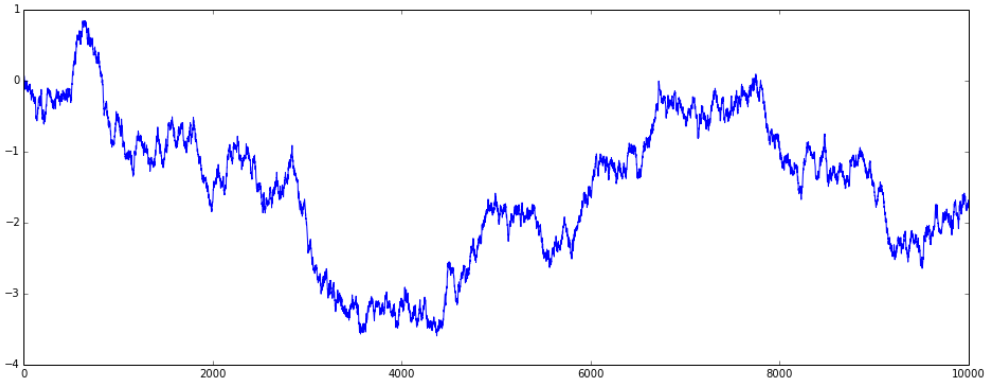


FIGURE 2.8. Time series linear combination using the third cointegration vector

2.2. Unit root tests

Many economic and financial time series exhibit trending behaviour or non-stationary behaviour in the mean or variance. An important econometric task is to determine the type of non-stationary. For trend stationary I(0) time series a time-trend regression is appropriate. For I(1) time series taking first differences transforms it into a stationary one. For cointegration to determine if time series are I(1) is mandatory to model their long-run relationship. In order to determine if a time series is an a I(1) or I(0) process unit root tests are used. Several tests have been developed but the most widely used is an extension of the Dickey and Fuller test [**DF79**] called the Augmented Dickey-Fuller (ADF) test developed by Said and Dickey in 1984[**SD84**].

The ADF test tests the null hypothesis that a time series y_t is I(1) against the alternative that it is I(0), assuming that the dynamics in the data have an ARMA structure. The ADF test is based on estimating the test regression:

$$y_t = \boldsymbol{\beta}^{\top}\mathbb{D}_t + \phi y_{t-1} + \sum_{j=1}^{p} \Psi_j \Delta y_{t-j} + \epsilon_t \tag{2.5}$$

where $\mathbb{D}_t$ is a vector of deterministic terms. Lag length p has to be determined when applying the test, Akaike Information Criteria (AIC) is commonly used to determine it. The p-lagged difference terms Δy_{t-j} are used to approximate an ARMA process and the value of p is set so that the error ϵ_t is serially uncorrelated. Under the null hypothesis, y_t is I(1) which implies that $\phi = 1$. The test statistic is the following:

$$\mathrm{ADF}_t = \frac{\hat{\phi} - 1}{SE(\phi)}$$

where SE is the standard error.

The ADF_t statistic is based on the least squares estimates of equation 2.5.

2.3. Volatility in the financial markets

In financial markets, volatility is one of the key elements to model the stochastic dynamic behaviour of financial assets. This is mainly because volatility gives a measure of uncertainty about future returns and it is often viewed as an indicator of the vulnerability of financial markets. It is also important as an input parameter in problems like derivative pricing, hedging and portfolio management. For example, in option pricing we need to known first the volatility of the underlying asset before pricing an option.

Besides, volatility provides important information for studying asset returns because the latter are largely uncorrelated and nonlinearly dependent. However, it has been

found that volatility exhibits significant autocorrelation and predictable patterns [**PG03**] and many models have been proposed to forecast its behaviour.

The volatility of the data is due to a large number of factors affecting the market, with some of them directly measurable such as historical prices, trends, supply and demand. However, others such as monetary policies and news are not directly measurable.

Despite the fact variance and volatility are related, they are not the same concept. Variance is a measure of distribution of returns and is not necessarily bound by any time period. Volatility is a measure of the standard deviation (square root of the variance) over a certain time interval. In finance, variance and volatility both gives you a sense of an asset's risk. Variance gives you a sense of the risk in the asset over its lifetime, while volatility gives you a sense of the movement of the asset in, for example, the past month or the past year. The main underlying difference is in their definition. Variance has a fixed mathematical definition, however volatility does not as such. Volatility is said to be the measure of fluctuations of a process.

Volatility is a subjective term, whereas variance is an objective term i.e. given the data you can definitely find the variance, while you can't find volatility just having the data. Volatility is associated with the process, and not with the data. In order to know the volatility you need to have an idea of the process i.e you need to have an observation of the dispersion of the process. All the different processes will have different methods to compute volatilities based on the underlying assumptions of the process.

2.3.1. Types of volatility. The volatility of a stock is not directly observable [**Tsa05, Eng93**]. For example, daily volatility is not directly observable from only daily returns because there is only one observation in a trading day. If intraday data is available, then volatility could be estimated. However, intraday returns are not the only explanatory variables for volatility and several estimators have been proposed. These estimators are observable variables that are related to the latent variable of interest called volatility proxies [**DVV07**]. Examples of volatility proxies are the following:

2.3.1.1. *Realized volatility.* is also known as historic volatility and it is the actual variance in the price of a stock over time. Realized volatility is measured in terms of the standard deviation using the historical stock prices. It is commonly calculated based on intraday price returns:

$$r_{t,n} = 100(\ln(p_{t,n}) - \ln(p_{t,n-1})) \tag{2.6}$$

where $p_{t,n}$ is the price observed at day $t = 1, \ldots, T$ and intraday sample $n = 2, \ldots, N$. Realized volatility is defined as:

$$\hat{\sigma}(t) = \sum_{n=1}^{N} r_{t,n}^2 \,, \tag{2.7}$$

where N is the number of intraday samples and T is the number of days. In order to include overnight returns, Hansen and Lunde [**LH05a**] introduced a scaling version of

realized volatility using the following definitions:

$$r_t = 100(\ln(p_{t,N}) - \ln(p_{t-1,N})) \tag{2.8}$$

$$\bar{\rho}(t) = \sum_{t=1}^{T} r_t^2 \,. \tag{2.9}$$

where equation (2.8) represents overnight returns and the volatility as equation (2.9), where $p_{t,N}$ is the last intraday sample at day t. The scaled realized volatility $\rho(t)$ is defined as:

$$\rho(t) = \gamma\hat{\rho}(t)\,, \qquad \gamma = \frac{\bar{\rho}(t)}{\sum_{t=1}^{T}\hat{\rho}(t)} \tag{2.10}$$

Realized volatility has also been defined as the absolute value return or as the mean of the sum of intraday squared returns at short intervals of time. The majority of research carried out in the literature obtain the daily volatility as the daily squared returns as is shown in equation (2.9). However, it has been proven that this measurement noise is too high for observing the true volatility process [**AB98**]. Hansen and Lunde [**HL06**] stated that the use of a noisy proxy could result in an inferior model being chosen as the best one. The realized volatility, as calculated by the cumulative sum of squared intraday returns and shown in equation (2.7), is less noisy and doesn't lead to choosing an inferior model.

2.3.1.2. *Implied volatility.* not only can be extracted from returns but it can also be derived from option or future pricing models. The volatility obtained corresponds to the market's prediction of future volatility. In finance, an option is a derivative, that is, a contract which gives the owner the right, but not the obligation to buy or sell an underlying asset at a given price called strike price. An option can be executed at any time before an expiration date previously defined no matter what price the underlying asset has. For example, the Black-Scholes model [**BS73**] determines the fair option value based on stock price, strike price, time to option expiration, the interest rate and volatility. These are known or can be easily obtained from the market, excepting by volatility which must be estimated. However, rather than assuming a volatility a priori and computing option prices from it, the model can be used to estimate volatility at given prices, time to expiration and strike price. This obtained volatility is called the implied volatility of an option. Additionally, some models obtain implied volatility from futures (other derivative from prices). For instance, the Barone-Adesi and Whaley futures option model [**BAW87**] is also used to determine future volatilities [**HI04**]. Higher implied volatility is indicative of greater price fluctuation in either direction. Implied volatility is found by determining the value which makes theoretical prices equal to market prices. In this way volatility is "implied" by the current market price of the stock.

In finance, an option is a derivative, that is, a contract which gives the owner the right, but not the obligation to buy or sell an underlying asset at a given price called

strike price. An option can be executed at any time before an expiration date previously defined no matter what price the underlying asset has.

The Black-Scholes formula [**BS73**], developed in the early 1970's, Myron Scholes, Robert Merton and Fisher Black, allows to determine an option value V based on the underlying asset price $S(t)$ at a time t and the following constant parameters:

σ**::** underlying asset price volatility which measures the standard deviation of the returns

μ**::** underlying asset drift which is a measure of the average rate of growth of the stock

E**::** option strike or excersice price

T**::** option date of expiry

t**::** current time

r**::** risk-free interest rate

The Black-Scholes model assume that the underlying price S follows a lognormal random walk:

$$dS = \mu S dt + \sigma S dB \tag{2.11}$$

where B is a Brownian motion. This stochastic differential equation has two components: a deterministic term given by $\mu S dt$ and a random term given by $\sigma S dX$.

A brownian motion B (also called a Wiener process) is a stochastic process characterized by the three following properties:

Continuity:: $B(t)$ is a continuos function

Normal increments:: $B(t) - B(s)$ has a normal distribution with mean 0 and variance $t - s$.

Independence of increments:: for every choice of nonnegative real numbers $0 \leq s_1 < t_1 \leq \cdots \leq s_n < t_n < \infty$, the increment random variables $W_{t_1} - W_{s_1}, \cdots, W_{t_n} - W_{s_n}$ are jointly independent.

An stochastic differential integral has the form:

$$W(T) = \int_0^T f(t) dB(t)\,. \tag{2.12}$$

This equations is also expressed in an abbreviate form:

$$dW = f(t) dB\,. \tag{2.13}$$

Therefore, the integral form of the stock price model shown in equation (2.11) is:

$$S(T) = \int_0^T \mu S(t) dt + \int_0^T \sigma S(t) dB(t) \tag{2.14}$$

Ito's lemma is used to find the differential of a time dependent function of a stochastic process. In option pricing we need to find the option price $V(S(t))$ which depends on a stochastic stock price model $S(t)$. $V(S,t)$ is required to be differentiable function of S and once differentiable function of t.

These are known or can be easily obtained from the market, excepting by volatility which must be estimated. However, rather than assuming a volatility a priori and computing option prices from it, the model can be used to estimate volatility at given prices, time to expiration and strike price. This obtained volatility is called the implied volatility of an option. Additionally, some models obtain implied volatility from futures (other derivative from prices).

For trading strategies, the interest is centred in forecasting realized volatility over the life of an option and to take advantage when this volatility differs from the implied volatility. This is called volatility arbitrage. For example, a trader will buy an option and hedge the underlying asset if the implied volatility is under the realized volatility.

2.3.2. Volatility methods. In the existing literature, there are four main classes of asset return volatility models: the general autoregressive conditional heteroskedasticity (GARCH) models, the stochastic volatility (SV) models, the realized volatility models and the machine learning based models. A comparison of the first three models can be found in [**Wei12**].

For many years the most popular methods for estimating financial volatility were the autoregressive conditional heteroskedasticity (ARCH) models [**Eng82**] and the general ARCH (GARCH) models [**Bol86**]. For instance, the GARCH(1,1) defines returns y_t and volatility σ_t as:

$$\begin{aligned} y_t &= \sigma_t \epsilon_t \\ \sigma_t^2 &= \alpha_0 + \alpha_1 \epsilon_{t-1}^2 + \beta_1 \sigma_{t-1}^2 \end{aligned}$$

where ϵ_t is standard Gaussian white noise, $\alpha_0, \alpha_1, \beta_1 \geq 0$ are required to ensure that the variance will never be negative and $\alpha_1 + \beta_1 < 1$ is needed to guarantee a weakly stationary process [**Nel**].

Since the introduction of the GARCH models, several extensions have been proposed, but none of them seems to beat the GARCH(1,1) model [**LH05b**]. Despite its popularity, GARCH models have several limitations: firstly, a time series model may be non-linear in mean and/or non-linear in variance, but ARCH and GARCH models are non-linear in variance, but not in mean. Besides, GARCH models often fail to capture highly irregular phenomena, like wild market fluctuations.

SV models explain how volatility varies in a random fashion. These models are useful because they explain why options with different strikes and expirations dates have different Black-Scholes implied volatilities, phenomenon known as the volatility smile. This is useful because the Black-Scholes model assumes that the volatility of the underlying asset is constant which is not always true. There are several SV models and

the most well-known and popular is the Heston model [**Hes93**]. Additional information about SV models can be found in [**She95**].

The realized volatility constructed from high frequency intraday returns gave rise to the realized volatility models mainly because the realized volatility series is much more homoskedastic and seems to be a long memory process [**ABDL03**]. For realized volatility, the autoregressive fractionally integrated moving average (ARFIMA) process emerged as a standard model [**CKP10**] and many variations have been studied, but all of them produce similar forecasting results to the ARFIMA(1,d,1) model [**KJH05**].

On the other hand, machine learning based models, especially artificial neural networks (ANN) and support vector machines (SVM) have arisen as an alternative to forecast volatility. ANN is a statistical technique inspired by biological neural networks which is capable of changing its structure based on external or internal information during a training phase [**SW11**]. SVM are supervised learning models for classification analysis which recognize patterns finding a separating hyperplane. An extension for regression analysis is known as support vector regression (SVR).

Since machine learning models and in particular ANN do not require assumptions about the data (gaussianity for example) and allow more explanatory variables than returns to be included, they have become widely used in solving financial problems, specially volatility forecasting [**HI04, DK97**]. There are also many works focused on the using of SVM in volatility forecasting [**CJH08, CHJ10, GB06, PPG12**].

However, just as with ANN, SVMs have scalability problems because their training process is computationally intensive and it is done in batch mode. The scalability problem worsens when new additional training data is available and a re-training process from scratch needs to be done. This problem can be avoided using online machine learning algorithms that allow one instance at a time to be processed with low computationally expensive calculations.

2.4. Univariate Time Series modeling

The random walk model, despite its simplicity is still difficult to outperform for standard econometric forecasting models [**LM11**] despite its simplicity. The random walk model is defined as:

$$\mathbf{y}_t = \mathbf{y}_{t-1} + \epsilon_t \tag{2.15}$$

The naive forecast of the time series difference $\hat{\mathbf{y}}_{t+1}$ for the random walk model is defined as:

$$\hat{\mathbf{y}}_{t+1} = \mathbf{y}_t + \hat{\epsilon}_{t+1} \tag{2.16}$$

where $\hat{\epsilon}_{t+1} = \epsilon_t$.

On the other hand, ARIMA is widely used to forecast returns in finance [**Tsa05**]. A process can be modelled as an ARIMA(p,d,q) model if $\mathbf{x}_t = \Delta^d \mathbf{y}_t$, i.e after differencing d times the time series $\mathbf{y}_t$, we get an ARMA(p,q). An ARMA(p,q) model is the following:

$$\mathbf{x}_t = \sum_{i=1}^{p} \phi_i \mathbf{x}_{t-i} + \sum_{j=1}^{q} \theta_j \epsilon_{t-j} \tag{2.17}$$

with coefficients $\phi_p \neq 0$, $\theta_q \neq 0$ and $\sigma_\epsilon^2 > 0$.

2.5. Multivariate Time Series modelling

The vector autoregressive VAR(p) model [**Sim80**] is one of the most easy to use, successful and flexible models for the analysis of multivariate time series. It is a natural extension of the univariate autoregressive (AR) model. The VAR model has proven to be useful for describing the dynamic behaviour of economic and financial time series. VAR is a general framework describing the behaviour of a set of l endogenous variables as a linear combination of their last p values, where $l, p \in \mathbb{N}$. In our case, each one of these l variables is a scalar time series $y_{i,t}$, $i = 1, \ldots, l$, and we represent them all together at time t by the the vector time series:

$$\mathbf{y}_t = \begin{bmatrix} y_{1,t} & y_{2,t} & \cdots & y_{l,t} \end{bmatrix}^\top .$$

Notice that the vector $\mathbf{y}_t$ is assumed to be l-dimensional.

The VAR(p) model describes the behaviour of a dependent variable in terms of its own lagged values and the lags of the others variables in the system. The model with p lags is formulated as the system of N:

$$\begin{aligned} \mathbf{y}_t = \mathbf{\Phi}_1 \mathbf{y}_{t-1} + \mathbf{\Phi}_2 \mathbf{y}_{t-2} + \cdots + \mathbf{\Phi}_p \mathbf{y}_{t-p} + \mathbf{c} + \boldsymbol{\epsilon}_t \\ t = p+1, \ldots, N, \end{aligned} \tag{2.18}$$

where $\mathbf{\Phi}_1, \mathbf{\Phi}_2, \ldots, \mathbf{\Phi}_p$ are $l \times l$-matrices of real coefficients, $\boldsymbol{\epsilon}_{p+1}, \boldsymbol{\epsilon}_{p+2}, \ldots, \boldsymbol{\epsilon}_N$ are error terms, $\mathbf{c}$ is a constant vector and N is the total number of samples.

Notice that, regarding our notation of section (2.1.6), we have here $\mathbf{y}_t^0 = \mathbf{y}_t$, $\mathbf{y}_t^\nu = \mathbf{y}_{t-\nu}$ and the i-th component of the vector time series $\mathbf{y}_t^\nu$ is the scalar time series $y_{i,t}^\nu$, where $\nu = 1, \ldots, p$ and $i = 1, \ldots, l$. Transposing each equation of the system (2.18) we can write the VAR(p) model in block-matrix form as:

$$\mathbf{Y} = \mathbf{A}\mathbf{X} + \mathbf{E}\,, \tag{2.19}$$

where:

$$\mathbf{Y} = \begin{bmatrix} \mathbf{y}_{p+1}^\top \\ \mathbf{y}_{p+2}^\top \\ \vdots \\ \mathbf{y}_N^\top \end{bmatrix}_{(N-p)\times l} \qquad \mathbf{A} = \left[\begin{array}{cccc:c} \mathbf{y}_p^\top & \mathbf{y}_{p-1}^\top & \cdots & \mathbf{y}_1^\top & 1 \\ \mathbf{y}_{p+1}^\top & \mathbf{y}_p^\top & \cdots & \mathbf{y}_2^\top & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{y}_{N-1}^\top & \mathbf{y}_{N-2}^\top & \cdots & \mathbf{y}_{N-p}^\top & 1 \end{array}\right]_{(N-p)\times(pl+1)}$$

$$\mathbf{X} = \begin{bmatrix} \boldsymbol{\Phi}_1^\top \\ \boldsymbol{\Phi}_2^\top \\ \vdots \\ \boldsymbol{\Phi}_p^\top \\ \mathbf{c}^\top \end{bmatrix}_{(pl+1)\times l} \qquad \mathbf{E} = \begin{bmatrix} \boldsymbol{\epsilon}_{p+1}^\top \\ \boldsymbol{\epsilon}_{p+2}^\top \\ \vdots \\ \boldsymbol{\epsilon}_N^\top \end{bmatrix}_{(N-p)\times l}$$

Taking into account the error term $\mathbf{E}$, equation 2.19 can be solved with respect to $\mathbf{X}$ using the ordinary least squares estimation (see section 2.6.1).

2.6. Vector error correction model

Vector error correction model (VECM) [**EG87a**] describe the joint behaviour of a set of cointegrated I(1) time series [**Ban93**] and can be derived from the simple VAR model.

It is obtained re-writing equation 2.18 in terms of the new variable $\Delta\mathbf{y}_t = \mathbf{y}_t - \mathbf{y}_{t-1}$. The VECM model, expressed in terms those differences, takes the form:

$$\Delta\mathbf{y}_t = \boldsymbol{\Omega}\,\mathbf{y}_{t-1} + \sum_{i=1}^{p-1} \boldsymbol{\Phi}_i^*\,\Delta\mathbf{y}_{t-i} + \mathbf{c} + \boldsymbol{\epsilon}_t\,, \tag{2.20}$$

where the coefficients matrices $\boldsymbol{\Phi}_i^*$ and $\boldsymbol{\Omega}$, expressed in terms of the matrices $\boldsymbol{\Phi}_i$ of the model VAR shown in equation 2.18, are:

$$\boldsymbol{\Phi}_i^* := -\sum_{j=i+1}^{p} \boldsymbol{\Phi}_j\,,$$
$$\boldsymbol{\Omega} := -\left(\mathbb{I} - \boldsymbol{\Phi}_1 - \cdots - \boldsymbol{\phi}_p\right).$$

The following well known properties of the matrix $\boldsymbol{\Omega}$ [**Joh95**] will be useful in the sequel:

- If $\boldsymbol{\Omega} = \mathbf{0}$, there is no cointegration.
- If $rank(\boldsymbol{\Omega}) = l$, i.e., if $\boldsymbol{\Omega}$ has full rank, then the time series are not I(1) but stationary.
- If $rank(\boldsymbol{\Omega}) = r$, $0 < r < l$, then there is cointegration and the matrix $\boldsymbol{\Omega}$ can be expressed as $\boldsymbol{\Omega} = \boldsymbol{\alpha}\boldsymbol{\beta}^\top$, where $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are $l \times r$ matrices and $\text{rank}(\boldsymbol{\alpha}) = \text{rank}(\boldsymbol{\beta}) = r$.

- The columns of $\boldsymbol{\beta}$ contains the cointegration vectors and the rows of $\boldsymbol{\alpha}$ correspond with the adjusted vectors. $\boldsymbol{\beta}$ is obtained by Johansen procedure [**Joh88**], whereas $\boldsymbol{\alpha}$ has to be determined as a variable in the VECM.

It is worth noticing that the factorization of the matrix $\boldsymbol{\Omega}$ is not unique, since for any $r \times r$ nonsingular matrix $\mathbf{H}$, $\boldsymbol{\alpha}^* := \boldsymbol{\alpha}\mathbf{H}$, and $\boldsymbol{\beta}^* = \boldsymbol{\beta}(\mathbf{H}^{-1})^\top$ we have $\boldsymbol{\alpha}\boldsymbol{\beta}^\top = \boldsymbol{\alpha}^*(\boldsymbol{\beta}^*)^\top$. If cointegration exists, then equation (2.20) can be written as follows:

$$\Delta \mathbf{y}_t = \boldsymbol{\alpha}\boldsymbol{\beta}^\top \mathbf{y}_{t-1} + \sum_{i=1}^{p-1} \boldsymbol{\Phi}_i^* \, \Delta \mathbf{y}_{t-i} + \mathbf{c} + \boldsymbol{\epsilon}_t \,, \tag{2.21}$$

which is a VAR model but for time series differences.

Transposing each equation of the system (6.1) we can write the VECM(p) model in block-matrix form as:

$$\mathbf{Y} = \mathbf{A}\mathbf{X} + \mathbf{E} \,, \tag{2.22}$$

where $\mathbf{Y}$ dimension is $((N-p) \times l)$, $\mathbf{A}$ dimension is $((N-p) \times (r + (p-1)l + 1))$, $\mathbf{X}$ dimension is $((r + (p-1)l + 1) \times l)$ and $\mathbf{E}$ dimension is $((N-p) \times l)$:

$$\mathbf{Y} = \begin{bmatrix} \Delta \mathbf{y}_{p+1}^\top \\ \Delta \mathbf{y}_{p+2}^\top \\ \vdots \\ \Delta \mathbf{y}_N^\top \end{bmatrix} \tag{2.23}$$

$$\mathbf{X} = \begin{bmatrix} \boldsymbol{\alpha}^\top \\ \boldsymbol{\Phi}_1^{*\top} \\ \boldsymbol{\Phi}_2^{*\top} \\ \vdots \\ \boldsymbol{\Phi}_{p-1}^{*\top} \\ \mathbf{c}^\top \end{bmatrix} \tag{2.24}$$

$$\mathbf{E} = \begin{bmatrix} \boldsymbol{\epsilon}_{p+1}^\top \\ \boldsymbol{\epsilon}_{p+2}^\top \\ \vdots \\ \boldsymbol{\epsilon}_N^\top \end{bmatrix} \tag{2.25}$$

and

$$\mathbf{A} = \left[\begin{array}{ccccc:c} \mathbf{y}_p^\top\boldsymbol{\beta} & \Delta \mathbf{y}_p^\top & \Delta \mathbf{y}_{p-1}^\top & \cdots & \Delta \mathbf{y}_2^\top & 1 \\ \mathbf{y}_{p+1}^\top\boldsymbol{\beta} & \Delta \mathbf{y}_{p+1}^\top & \Delta \mathbf{y}_p^\top & \cdots & \Delta \mathbf{y}_3^\top & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{y}_{N-1}^\top\boldsymbol{\beta} & \Delta \mathbf{y}_{N-1}^\top & \Delta \mathbf{y}_{N-2}^\top & \cdots & \Delta \mathbf{y}_{N-p-1}^\top & 1 \end{array}\right] . \tag{2.26}$$

Taking into account the error term $\mathbf{E}$, equation 2.22 can be solved with respect to $\mathbf{X}$ using the ordinary least squares estimation (see section 2.6.1).

2.6.1. Ordinary Least Squares method. When $\mathbf{A}$ is singular, solution to equation (6.2) is given by the ordinary least squares (OLS) method. OLS consists of minimizing the sum of squared errors or equivalently minimizing the following expression:

$$\min_{\mathbf{X}} \quad \|\mathbf{AX} - \mathbf{B}\|_2^2 \tag{2.27}$$

for which the solution $\hat{\mathbf{X}}$ is well-known:

$$\hat{\mathbf{X}} = \mathbf{A}^+ \mathbf{B} \tag{2.28}$$

where $\mathbf{A}^+$ is the Moore-Penrose pseudo-inverse which can be written as follows:

$$\mathbf{A}^+ = (\mathbf{A}^\top \mathbf{A})^{-1} \mathbf{A}^\top . \tag{2.29}$$

However, when $\mathbf{A}$ is not full rank, i.e $rank(\mathbf{A}) = k < n \leq m$, $\mathbf{A}^\top \mathbf{A}$ is always singular and equation (3.9) cannot be used. More generally, the pseudo-inverse is best computed using the compact singular value decomposition (SVD) of $\mathbf{A}$:

$$\underset{m\times n}{\mathbf{A}} = \underset{m\times k}{\mathbf{U_1}} \; \underset{k\times k}{\Sigma_1} \; \underset{k\times n}{\mathbf{V}_1^\top}, \tag{2.30}$$

as follows

$$\mathbf{A}^+ = \mathbf{V}_1 \Sigma_1^{-1} \mathbf{U}_1^\top . \tag{2.31}$$

Conventional regression estimators, including VARs, have good properties when applied to covariance-stationary time series, but encounter difficulties when applied to non stationary or integrated processes.

These difficulties were illustrated by Granger and Newbold in 1974 [**GN74**] when they introduced the concept of spurious regressions (see section 2.1.4). In 1982, Nelson and Plosser [**NP82**] showed that unit roots might be present in a wide variety of macroeconomic series in levels or logarithms. This finding gave rise to the industry of unit root testing, and the implication that variables should be rendered stationary by differencing before they are included in an econometric model. Further theoretical developments by Granger and Engle in 1987 [**EG87a**] raised the possibility that two or more integrated, non stationary time series might be cointegrated, so that some linear combination of these series could be stationary even though each series were not.

If two series are both integrated (of order one, or I(1)) we could model their interrelationship by taking first differences of each series and including the differences in a VAR or a structural model. However, this approach would be suboptimal if it was determined that these series are indeed cointegrated. In that case, the VAR only express the short-run responses of these series to innovations in each series. This implies that the simple regression in first differences is misspecified. If the series are cointegrated, they move together in the long-run. A VAR in first differences will not capture those long-run tendencies. VECM fit to the first differences of the non stationary variables, but a lagged error-correction term is added to the relationship representing the long-run relationship.

In finance, many economic time series are revealed to be stationary when they are differentiated and moreover cointegration restrictions often improve forecasting [**DT98**]. Therefore, VECM has been widely adopted in finance: [**MN95**], [**MK00**], [**ADL01**] and [**SAZ13**] to name a few. Pair trading is a very common example of a cointegration application [**Her03**] but also can also be extended to a larger set of variables [**MN95**], [**EP04**].

Chapter 3

Machine Learning Models

Machine learning is a scientific discipline focused on the development of algorithms based on learning from examples. This idea has become central to the design of search engines, robots systems and forecasts applications that process large data sets. Usually machines designed to forecast financial time series requires long training period and therefore they are not suitable to be updated with stream data. Online machine learning techniques tackle this problem by updating the model with new data or by computing a new model using less data so they can give a response in a short period of time.

3.1. Introduction

Machine Learning (ML) studies computer algorithms for learning something such as to complete a task, to make accurate predictions or to behave intelligently. The learning is always based on samples and the objective is about to do better in the future based on the past experiences in an automatic way. ML is a sub-area of artificial intelligence and broadly intersects with other fields such as statistics, mathematics, physics and computer science. There are many examples of machine learning problems: time series forecasting, image processing, face detection, spam filtering, weather prediction, search engines, among many others.

ML models are often more accurate than what can be created through direct programming. The reason for this is that ML models are data driven and are able to examine large amounts of data.

There are three types of ML classified depending on the nature of the learning input or output available to a learning system:

Supervised learning: the input data is a tuple which contains the example input and its desired outputs (also called labels). The list of tuples is called the training set. The concept of supervised learning comes from the supervisor, acting as a teacher in the learning process. The goal is to learn a general rule that maps inputs to outputs optimising a target function. There are two related problem types in supervised learning: classification and regression problems [**B⁺06**]. Its two mainstream approaches are: support vector machines (SVMs) [**Vap98**] and ensemble learning [**B⁺98**]. Furthermore, supervised learning can be categorised into offline or batch learning and online learning (see section 3.3).

Unsupervised learning: also known as clustering [**BDvLSTT05**]. In this type of learning no labels are given and the system has to find a structure on its own, discovering hidden patterns in data.

Reinforcement learning: is the problem faced by an agent that must learn through trial-and-error interactions with a dynamic environment. It is based on programming agents by reward and punishment without the need to specify how the task is to be achieved [**SB98**].

3.2. Statistical learning theory

All ML problems can be viewed as optimisation problems. The ML core task is to define a learning criterion, i.e the function to be optimised.

Supervised learning is most popular and most commonly used in modelling financial problems and the assessments of this method and their results in practice are fairly good. Therefore, in this thesis we will adhere to this trend and we will use supervised learning.

Supervised learning consists in finding a learning function $f : X \to Y$ which models a set of examples, called training set S, which have been drawn randomly and independently according to a unknown joint distribution function $p(x, y)$ on $X \times Y$:

$$S = \{(x_k, y_k) \in X \times Y : k = 1, ..., n\}$$

where $X \subseteq \mathbb{R}^m$ and $Y = \mathbb{R}$ in case of regression problems and $Y = \{1, -1\}$ if it is a classification problem.

A learning algorithm $\mathcal{A}$ takes as input a data set $S \in \mathcal{T}$ and selects from $\mathcal{H}$ a function f such that $f(x) \approx y$ in a predictive way:

$$\begin{aligned} \mathcal{A} : \mathcal{S} &\to \mathcal{H} \\ S &\to \mathcal{A}(S) = f \end{aligned}$$

where $\mathcal{H}$ is called the *hypothesis space*, is the space of functions that the algorithm is allowed to search. The selection of f is based on the minimization of the expected error of the loss function $V(f(x), y)$ defined as:

$$\begin{aligned} V : Y \times Y &\rightarrow Y \\ f(x), y &\rightarrow V(f(x), y) \end{aligned}$$

Therefore, given a function f a loss function V and a probability distribution p over $X \times Y$, the expected error of f is:

$$I[f] = E[V(f(x), y)] = \int_{X \times Y} V(f(x), y)\, dp(x, y) \tag{3.1}$$

$V(f(x), y)$ denote the price paid for mistakes. Therefore, $V(f(x), y) = 0$ if $f(x) = y$.

For regression, the most common loss function is square loss or L2 loss function:

$$V(f(x), y) = (f(x) - y)^2 \tag{3.2}$$

another option is the L1 loss

$$V(f(x), y) = |f(x) - y|\,. \tag{3.3}$$

the choice of loss function here gives rise to several well-known learning algorithms such as regularised least squares and support vector machines.

Since the true distribution is unknown and only training samples are available, the objective is to estimate a function $\hat{f}$ through empirical risk (training error) minimisation (ERM):

$$R_{\text{emp}}[f] = \frac{1}{n} \sum_{i=1}^{n} V(f(x), y) \tag{3.4}$$

3.2.1. Learning algorithms. In all learning algorithms that are trained from example data, there is a tradeoff [**Die03**] between three factors:

Complexity of the hypothesis class: for best generalisation, we should adjust the complexity of the learner model $\hat{f}$ to the complexity of the data f. The complexity of a learning problem mostly depends on the number of training examples and the size of the searched hypothesis space. In polynomial regression, the complexity parameter is the order of the fitted polynomial, and therefore we need to find a way to choose the best order that minimises the generalisation error, that is, tune the complexity of the model to best fit the complexity of the function inherent in the data.

Generalisation accuracy on new examples: is the capability of the algorithm to generate the right output for an input instance outside the training set. It is measured by the expected risk (equation 3.1).

The amount of training data: in most cases, generalisation accuracy increases as the amount of training data increases.

Very complex models will fit the training data better (low bias) but it could overfit it and generalise poorly (high variance), this is also called the bias-variance tradeoff. Generalisation accuracy can sometimes be improved with more training data. The figure 3.1 illustrates how generalisation accuracy depends on the complexity of the model and the amount of training data.

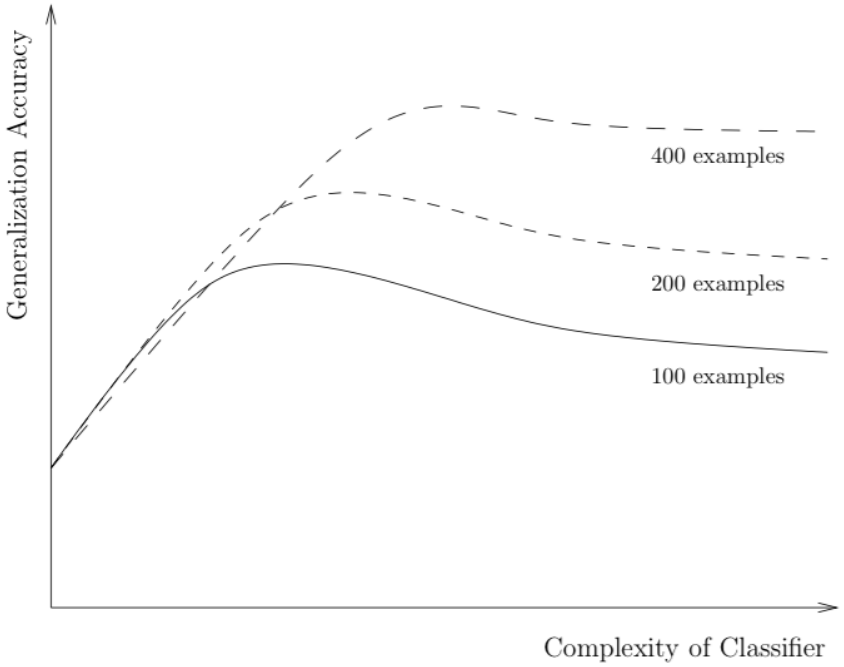


FIGURE 3.1. Tradeoff in empirical learning from [**Die03**]. The generalization accuracy is not affected by the amount of examples for low complex classifiers. However, for some more complex classifiers, the number of examples improves the generalization accuracy. The key is to find the best generalization accuracy and the lowest complex classifier as possible.

3.2.2. The bias-variance tradeoff. Models learning error can be split into two main components: error due to bias and error due to variance. Bias measures how far off in general these models' predictions are from the correct value. Variance is taken as the variability of a model prediction for a given data point.

If we have a learning model $\hat{f}$, its expected generalisation error on an unseen or testing sample x can be decomposed as follows:

$$\mathrm{E}\Big[\big(y - \hat{f}(x)\big)^2\Big] = \mathrm{Bias}(\hat{f}(x))^2 + \mathrm{Var}(\hat{f}(x)) + \sigma^2 \tag{3.5}$$

where:

$$\text{Bias}(\hat{f}(x)) = \text{E}\big[\hat{f}(x)\big] - f(x)$$
$$\text{Var}(\hat{f}(x)) = \text{E}\Big[\big(\hat{f}(x) - \text{E}[\hat{f}(x)]\big)^2\Big]$$

Demo

$$\begin{aligned} E[(y - \hat{f}(\mathbf{x}))^2] &= E[(y - f(\mathbf{x}) + f(\mathbf{x}) - \hat{f}(\mathbf{x}))^2] \\ &= E[(B - f(\mathbf{x}))^2] + E[(f(\mathbf{x}) - \hat{f}(\mathbf{x}))^2] + \dots \\ &\quad \cancelto{0}{2E[y - f(\mathbf{x})]E[f(\mathbf{x}) - \hat{f}(\mathbf{x})]} \\ &= \sigma^2 + MSE(\hat{f}(\mathbf{x})) \end{aligned}$$

where

$$\begin{aligned} MSE(\hat{f}(\mathbf{x})) &= E[f(\mathbf{x}) - \hat{f}(\mathbf{x}))]^2 \\ &= E[f(\mathbf{x}) - E[\hat{f}(\mathbf{x})] + E[\hat{f}(\mathbf{x})] - \hat{f}(\mathbf{x}))]^2 \\ &= E[(f(\mathbf{x}) - E[\hat{f}(\mathbf{x})])^2] + E[(\hat{f}(\mathbf{x}) - E[\hat{f}(\mathbf{x})])^2] + \dots \\ &\quad \cancelto{0}{E[(f(\mathbf{x}) - E[\hat{f}(\mathbf{x})])]E[(\hat{f}(\mathbf{x}) - E[\hat{f}(\mathbf{x})])]} \\ &= Bias(\hat{f}(\mathbf{x}))^2 + Var(\hat{f}(\mathbf{x})) \end{aligned}$$

■

Bias is introduced by the model selection. Therefore the model building process is repeated (through resampling) and substantially different averages of prediction values are obtained, bias will be high. The error due to variance is the amount by which the prediction, over one training set, differs from the expected predicted value, over all the training sets. Variance measures how inconsistent are the predictions from one another, over different training sets, not whether they are accurate or not.

This generalisation error is unknown, but it can be estimated using a training sample, a testing sample or by estimating the model complexity. Figure 3.2 shows prediction error for training and testing sample. In general, in the training sample prediction error is decreasing with more complex models but it overfit the data and it is not a good model for the test data. Figure 3.3 shows how testing error can be decomposed into bias and variance components. The best model will be the one has a balance between bias and variance.

The objective is to simultaneously reduce bias and variance as much as possible in order to obtain as accurate model as is feasible. However, there is a tradeoff to be made when selecting models. Models that exhibit small variance and high bias underfit the truth target. Models that exhibit high variance and low bias overfit the truth target.

This bias-variance tradeoff is the reason why in order to obtain the best model, data is usually broken in three subsets: training set, validation set and testing set. Training

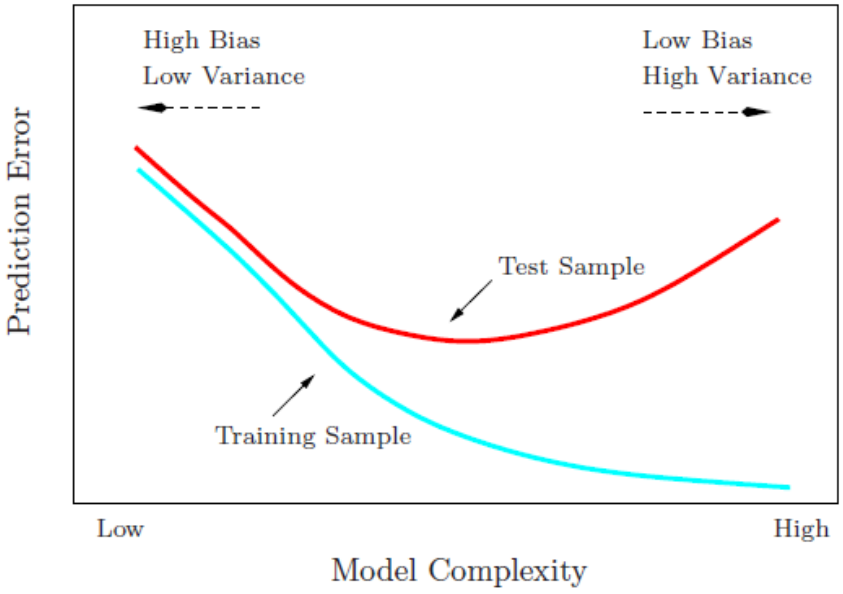


FIGURE 3.2. Training and test error

set is used to determine the model, validation set is used to estimate the generalisation error and finally testing set is used to estimate the accuracy of the model. Usually the partition is 50% for training set, 25% for validation and 25% for testing purposes. This procedure can be extended to a Cross Validation (CV) procedure [**Gei75**], usually used when the amount of data is limited. Various splitting strategies lead to various CV estimates. In K-fold cross-validation the training data is divided randomly into K distinct subsets, then the network is trained using K-1 subsets, and tested on the remaining subset. The process of training and testing is then repeated for each of the K possible choices of the subset omitted from the training. The average performance on the K omitted subsets is then our estimate of the generalisation performance.

Another way to control the complexity of the model is to include in the optimisation function not only the generalisation error but a penalisation of high complex models.

3.2.3. Regularization. Since ERM is an ill-posed problem, Tikhonov introduced a regularisation which ensures well-posedness and generalisation of ERM, i.e prevents overfit, by constraining the hypothesis space $\mathcal{H}$ usually called regularised ERM or Tikhonov regularisation.

Tikhonov regularisation minimised over the hypothesis space $\mathcal{H}$ for a fixed positive parameter λ the following:

$$R_{\text{emp}}[f] = \frac{1}{n}\sum_{i=1}^{n} V(f(x), y) + \lambda \mathcal{R}(f) \tag{3.6}$$

where $\mathcal{R}(f)$ is the regulariser, a penalisation on f.

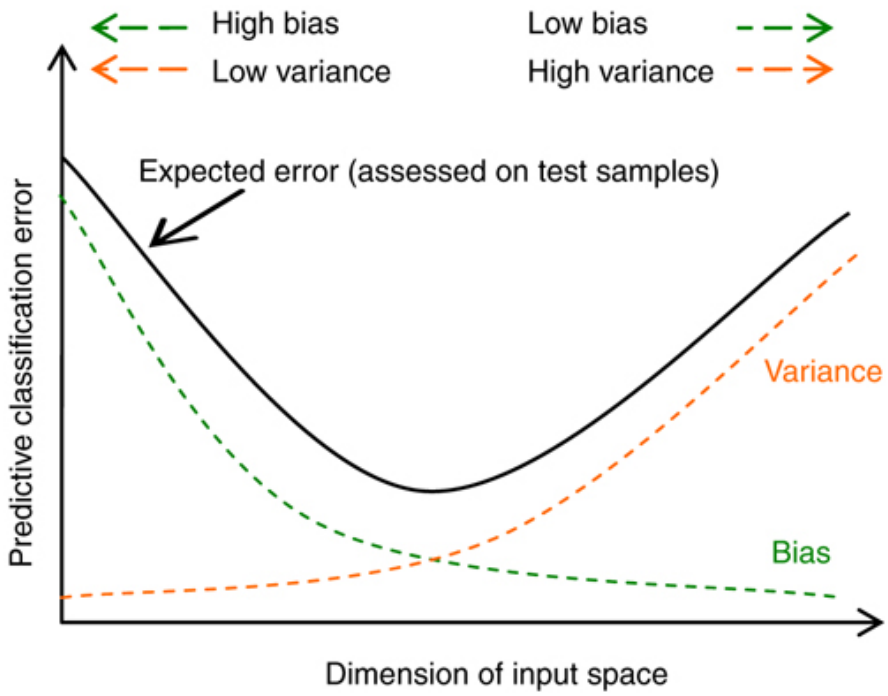


FIGURE 3.3. Bias variance tradeoff

3.2.4. Ridge Regression. One example of regularisation in linear models is Ridge Regression (RR), which is a regularised least squares method. The least squares (LS) method is a well known way to solve a regression problem. LS method consists of minimising the sum of squared errors:

$$\begin{aligned} J(\mathbf{X}) &= \sum_{t=1}^{N} (f(\mathbf{x}_t) - y_t)^2 \\ &= \sum_{t=1}^{N} (\mathbf{X}^\top \mathbf{x}_t - y_t)^2 \\ &= \|\mathbf{A}\mathbf{X} - \mathbf{Y}\|_2^2 \end{aligned}$$

which is equivalent to find a solution to

$$\underset{m\times n}{\mathbf{A}} \; \underset{n\times l}{\mathbf{X}} = \underset{m\times l}{\mathbf{Y}} \tag{3.7}$$

The optimal solution for $\mathbf{X}$ is:

$$\mathbf{X} = \mathbf{A}^{+}\mathbf{Y} \tag{3.8}$$

where $\mathbf{A}^{+}$ is the Moore-Penrose pseudo-inverse which can be written as follows:

$$\mathbf{A}^{+} = (\mathbf{A}^\top \mathbf{A})^{-1}\mathbf{A}^\top . \tag{3.9}$$

Demo

To solve equation (3.9) is equivalent to solve the following optimisation problem:

$$\min_{\mathbf{X}(\lambda)} \quad \|\mathbf{CX} - \mathbf{F}\|_2^2 \tag{3.10}$$

where

$$\mathbf{C} = \begin{bmatrix} - \\ \frac{\mathbf{A}}{\sqrt{\lambda}\mathbb{I}} \\ - \end{bmatrix} \quad \text{and} \quad \mathbf{F} = \begin{bmatrix} - \\ \frac{\mathbf{Y}}{\mathbf{0}} \\ - \end{bmatrix}.$$

Applying equation (3.8) and considering that $\mathbf{C}^\top\mathbf{C} = \mathbf{A}^\top\mathbf{A} + \lambda\mathbb{I}$ and $\mathbf{C}^\top\mathbf{F} = \mathbf{A}^\top\mathbf{Y}$ we have:

$$\begin{aligned} \mathbf{X}(\lambda) &= (\mathbf{C}^\top\mathbf{C})^{-1}\mathbf{C}^\top\mathbf{F} \\ &= (\mathbf{A}^\top\mathbf{A} + \lambda\mathbb{I})^{-1}\mathbf{A}^\top\mathbf{Y} \end{aligned}$$

■

However, when $\mathbf{A}$ is not full rank, i.e $rank(\mathbf{A}) = k < n \leq m$, $\mathbf{A}^\top\mathbf{A}$ is always singular and equation (3.9) cannot be used. More generally, the pseudo-inverse is best computed using the compact singular value decomposition (SVD) of $\mathbf{A}$:

$$\underset{m\times n}{\mathbf{A}} = \underset{m\times k}{\mathbf{U_1}}\ \underset{k\times k}{\mathbf{\Sigma_1}}\ \underset{k\times n}{\mathbf{V_1}^\top} \tag{3.11}$$

as follows

$$\mathbf{A}^+ = \mathbf{V_1}\mathbf{\Sigma_1^{-1}}\mathbf{U_1^\top} \tag{3.12}$$

Demo

In the case matrix $\mathbf{A}$ is non singular, its solution is straight forward:

$$\mathbf{X} = \mathbf{A}^{-1}\mathbf{Y} \tag{3.13}$$

Since the problem shown in equation (3.7) has not solution, the minimum norm given by equation (3.13) is obtained by solving the equivalent problem:

$$\mathbf{A}\hat{\mathbf{X}} = \mathbf{PY}$$

where $\mathbf{P} = \mathbf{U_1}\mathbf{U_1^\top}$ is the projection onto the Col($\mathbf{A}$).

Since $\mathbf{V} = \underset{(n\times k)}{[\ \mathbf{V_1}\ }\ |\ \underset{(n\times k)}{\mathbf{V_2}\ ]}$ and $\mathbf{V_1^\top V_2} = \mathbf{0}$ we can express $\hat{\mathbf{X}} = \mathbf{V_1X_1} + \mathbf{V_2X_2}$ with $\mathbf{X_2} = \mathbf{0}$ because $\hat{\mathbf{X}}$ lives in the Row($\mathbf{A}$) given by $\mathbf{V_1}$, so we have:

$$\begin{aligned}
\mathbf{A}\hat{\mathbf{X}} &= \mathbf{P}\mathbf{Y} \\
\mathbf{U_1}\mathbf{\Sigma_1}\mathbf{V_1}^\top\hat{\mathbf{X}} &= \mathbf{U_1}\mathbf{U_1}^\top\mathbf{Y} \\
\mathbf{V_1}^\top\hat{\mathbf{X}} &= \mathbf{\Sigma_1^{-1}}\mathbf{U_1}^\top\mathbf{Y} \\
\mathbf{V_1}^\top\mathbf{V_1}\mathbf{X_1} &= \mathbf{\Sigma_1^{-1}}\mathbf{U_1}^\top\mathbf{Y} \\
\mathbf{X_1} &= \mathbf{\Sigma_1^{-1}}\mathbf{U_1}^\top\mathbf{Y}
\end{aligned}$$

from this result we can obtain $\hat{\mathbf{X}}$ and therefore the pseudo-inverse expression:

$$\begin{aligned}
\hat{\mathbf{X}} &= \mathbf{V_1}\mathbf{X_1} \\
&= \mathbf{V_1}\mathbf{\Sigma_1^{-1}}\mathbf{U_1}^\top\mathbf{Y} \\
\mathbf{A}^+ &= \mathbf{V_1}\mathbf{\Sigma_1^{-1}}\mathbf{U_1}^\top .
\end{aligned}$$

■

In order to avoid the singularity of the matrix $\mathbf{A}^\top\mathbf{A}$, a regularisation term is introduced:

$$J(\mathbf{X}) = \|\mathbf{A}\mathbf{X} - \mathbf{Y}\|_2^2 + \lambda\|\mathbf{X}\|^2 \tag{3.14}$$

which optimal solution $\mathbf{X}_*$ is well known:

$$\mathbf{X}_* = (\mathbf{A}^\top\mathbf{A} + \lambda\mathbb{I})^{-1}\mathbf{A}^\top y ,$$

Demo

$$\begin{aligned}
J(\mathbf{X}) &= \|\mathbf{A}\mathbf{X} - \mathbf{Y}\|_2^2 + \lambda\|\mathbf{X}\|^2 \\
&= \sum_{t=1}^{N}(\mathbf{X}^\top\mathbf{a}_t - y_t)^2 + \lambda\sum_{i=1}^{p}\mathbf{X}_i^2 \\
&= (\mathbf{X}^\top a_1 - y_1)^2 + \cdots + (\mathbf{X}^\top a_N - y_N)^2 + \lambda(\mathbf{X}_1^2 + \cdots + \mathbf{X}_p^2)
\end{aligned}$$

taking derivatives

$$\begin{aligned}
\frac{\partial J(\mathbf{X})}{\partial \mathbf{X}_1} &= 2(\mathbf{X}^\top\mathbf{a}_1 - y_1)\mathbf{a}_{11} + \cdots + 2(\mathbf{X}^\top\mathbf{a}_N - y_N)\mathbf{a}_{N1} + 2\lambda\mathbf{X}_1 \\
&= 2\mathbf{a}_1^\top(\mathbf{A}\mathbf{X} - \mathbf{Y}) + 2\lambda\mathbf{X}_1 \\
&\vdots \\
\frac{\partial J(\mathbf{X})}{\partial \mathbf{X}_p} &= 2\mathbf{a}_p^\top(\mathbf{A}\mathbf{X} - \mathbf{Y}) + 2\lambda\mathbf{X}_p
\end{aligned}$$

Then we have that:

$$\frac{\partial J(\mathbf{X})}{\partial \mathbf{X}} = 2\mathbf{A}^\top(\mathbf{A}\mathbf{X} - \mathbf{Y}) + 2\lambda\mathbf{X}$$

Since $\frac{\partial J(\mathbf{X})}{\partial \mathbf{X}} = 0$ we have:

$$\begin{aligned} 2\mathbf{A}^\top(\mathbf{AX} - \mathbf{Y}) + 2\lambda\mathbf{X} &= 0 \\ \mathbf{A}^\top\mathbf{AX} - \mathbf{A}^\top\mathbf{Y} + \lambda\mathbf{X} &= 0 \\ (\mathbf{A}^\top\mathbf{A} + \lambda\mathbb{I})\mathbf{X} &= \mathbf{A}^\top\mathbf{Y} \\ \mathbf{X} &= (\mathbf{A}^\top\mathbf{A} + \lambda\mathbb{I})^{-1}\mathbf{A}^\top\mathbf{Y} \end{aligned}$$

■

3.2.5. The Lambda parameter. The additional term $\lambda\|\mathbf{X}\|_2^2$ in the optimisation problem shown in equation (3.14) has two effects on the solution: shrinks the coefficients towards zero and improves the conditioning of the problem.

When $\mathbf{A}$ is orthonormal then $\mathbf{A}^\top\mathbf{A} = \mathbb{I}$ and there is a simple relation between the ridge estimator and the OLS estimator:

$$\begin{aligned} \mathbf{X}_*(\lambda) &= (\mathbf{A}^\top\mathbf{A} + \lambda\mathbb{I})^{-1}\mathbf{A}^\top\mathbf{Y} \\ &= (\mathbb{I} + \lambda\mathbb{I})^{-1}\mathbf{A}^\top\mathbf{Y} \\ &= (1+\lambda)^{-1}\mathbb{I}\mathbf{A}^\top\mathbf{Y} \\ &= (1+\lambda)^{-1}(\mathbf{A}^\top\mathbf{A})^{-1}\mathbf{A}^\top\mathbf{Y} \\ &= (1+\lambda)^{-1}\mathbf{X} \end{aligned}$$

Figure 3.4 shows a visual example of the shrinking of the coefficients:

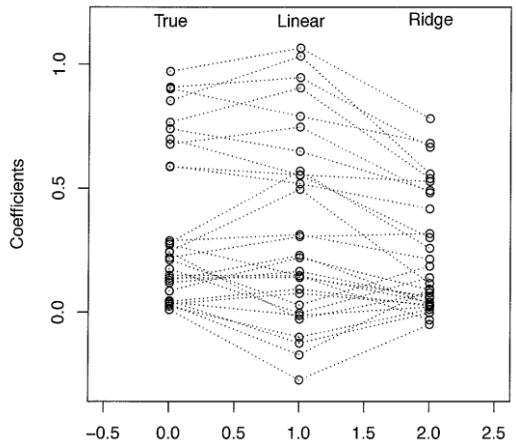


FIGURE 3.4. Shrink of regression coefficients

On the other hand, the effect of adding the term $\lambda\mathbb{I}$ to the matrix $\mathbf{A}^\top\mathbf{A}$ (equation (3.2.4)) improves its condition number since it increases its diagonal values when $\lambda > 0$. The matrix $\mathbf{A}^\top\mathbf{A}$ is symmetrical ($(\mathbf{A}^\top\mathbf{A})^\top = \mathbf{A}^\top\mathbf{A}$) and therefore diagonalizable. If we know the eigenvalue decomposition of $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$, then:

$$\begin{aligned}\mathbf{A}^\top\mathbf{A} + \lambda\mathbb{I} &= \mathbf{V}\boldsymbol{\Sigma}^{\mathbf{2}}\mathbf{V}^\top + \lambda\mathbf{V}\mathbf{V}^\top \\ &= \mathbf{V}(\boldsymbol{\Sigma}^2 + \lambda\mathbb{I})\mathbf{V}^\top,\end{aligned}$$

where

$$\boldsymbol{\Sigma}^2 + \lambda\mathbb{I} = \begin{bmatrix} \sigma_1^2 + \lambda & & & \\ & \sigma_2^2 + \lambda & & \\ & & \ddots & \\ & & & \sigma_n^2 + \lambda\,. \end{bmatrix}$$

where $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n$.

Since the condition number of a matrix $\mathbf{A}$ is defined as:

$$\kappa = \|\mathbf{A}\|\|\mathbf{A}^{-1}\|$$

If matrix $\mathbf{A}$ is non-singular, its condition number can be expressed in terms of its singular values. The effect of adding the regularization term affects the condition number as follows:

$$\begin{aligned}\kappa_{ols} &= \|\mathbf{A}\|\|\mathbf{A}^{-1}\| = \frac{\sigma_1}{\sigma_n} \\ \kappa_{ridge} &= \|\mathbf{A}^\top\mathbf{A} + \lambda\mathbb{I}\|\|(\mathbf{A}^\top\mathbf{A} + \lambda\mathbb{I})^{-1}\| = \frac{\sigma_1 + \lambda}{\sigma_n + \lambda}\end{aligned}$$

It is easy to see that the term λ improves the condition number:

$$\frac{\sigma_1 + \lambda}{\sigma_n + \lambda} < \frac{\sigma_1}{\sigma_n} \qquad \forall \quad \lambda > 0$$

However, λ cannot be too large. Tipically λ is small and its magnitude depends on the matrix $\mathbf{A}$.

For rank deficient matrices we know that $\det(\mathbf{A}\mathbf{A}^\top) = 0$, adding the term $\lambda\mathbb{I}$ we have that $\det(\mathbf{A}\mathbf{A}^\top + \lambda\mathbb{I}) = p(\lambda)$ where $p(\lambda)$ is a polynomial of degree n ($\mathbf{A}$ is $m \times n$). The zeros of $p(\lambda)$ are discretes, so it can be represented as:

$$p(\lambda) = \lambda(\lambda - \lambda_1)^{n_1}(\lambda - \lambda_2)^{n_2}\ldots(\lambda - \lambda_s)^{n_s}$$

where $n_1 + n_2 + \cdots + n_s = n$.

This means that λ must be small in order to ensure that $p(\lambda)$ does not vanish.

3.2.6. Selection of Lambda. One of the ways to determine parameter λ is using the bias-variance tradeoff (see section 3.2.2). This parameter is crucial for ridge regression since it could reduce the expected prediction error by reducing variance, considering a biased estimator. It is known that the prediction error can be express as a decomposition between bias and variance.

The solution of OLS is well known $\hat{\mathbf{X}} = (\mathbf{A}^\top\mathbf{A})^{-1}\mathbf{A}^\top\mathbf{Y}$, and its bias is:

$$\begin{aligned} Bias(\hat{f}(\hat{\mathbf{X}}) &= E[\hat{\mathbf{X}}] - \mathbf{X} \\ &= E[(\mathbf{A}^\top\mathbf{A})^{-1}\mathbf{A}^\top\mathbf{Y}] - \mathbf{X} \\ &= E[(\mathbf{A}^\top\mathbf{A})^{-1}\mathbf{A}^\top(\mathbf{A}\mathbf{X})] - \mathbf{X} \\ &= \mathbf{X} - \mathbf{X} \\ &= 0 \end{aligned}$$

The bias of ridge regression when $\mathbf{A}\mathbf{A}^\top$ is non-singular can be obtained expressing ridge regression solution λ in terms of OLS solution $\hat{\mathbf{X}}$:

$$\begin{aligned} \mathbf{X}(\lambda) &= (\mathbf{A}^\top\mathbf{A} + \lambda\mathbb{I})^{-1}\mathbf{A}^\top\mathbf{Y} \\ &= (\mathbb{I} + \lambda(\mathbf{A}^\top\mathbf{A})^{-1})^{-1}(\mathbf{A}^\top\mathbf{A})^{-1}\mathbf{A}^\top\mathbf{Y} \\ &= (\mathbb{I} + \lambda(\mathbf{A}^\top\mathbf{A})^{-1})^{-1}\hat{\mathbf{X}} \\ &= \mathbf{W}\hat{\mathbf{X}} \end{aligned}$$

where $\mathbf{W} = (\mathbb{I} + \lambda(\mathbf{A}^\top\mathbf{A})^{-1})^{-1}$ it is defined for simplicity. Ridge regression bias is then obtained as:

$$\begin{aligned} Bias(\mathbf{X}(\lambda)) &= E[\mathbf{X}(\lambda)] - \mathbf{X} \\ &= E[\mathbf{W}\hat{\mathbf{X}}] - \mathbf{X} \\ &= \mathbf{W}\mathbf{X} - \mathbf{X} \neq 0 \end{aligned}$$

The variance of OLS is:

$$Var(\hat{\mathbf{X}}) = \sigma^2(\mathbf{A}^\top\mathbf{A})^{-1}$$

and the variance of ridge regression is:

$$\begin{aligned} Var(\mathbf{X}(\lambda)) &= Var(\mathbf{W}\hat{\mathbf{X}}) \\ &= E[(\mathbf{W}\hat{\mathbf{X}} - E[\mathbf{W}\hat{\mathbf{X}}])(\mathbf{W}\hat{\mathbf{X}} - E[\mathbf{W}\hat{\mathbf{X}}])^\top] \\ &= \mathbf{W}E[(\hat{\mathbf{X}} - E[\hat{\mathbf{X}}])(\hat{\mathbf{X}} - E[\hat{\mathbf{X}}])^\top]\mathbf{W}^\top \\ &= \mathbf{W}Var(\hat{\mathbf{X}})\mathbf{W}^\top \\ &= \sigma^2\mathbf{W}(\mathbf{A}^\top\mathbf{A})^{-1}\mathbf{W}^\top \end{aligned}$$

The figure 3.5 shows the bias-variance tradeoff given by equation (3.5).

Despite OLS has zero bias, its variance is greater than ridge for small values of λ. It can be shown that, in terms of prediction error, ridge (black line) is lower than OLS (dotted line) [**HK70**]. Ridge regression shows an increasing squared bias and a decreasing variance.

Demo

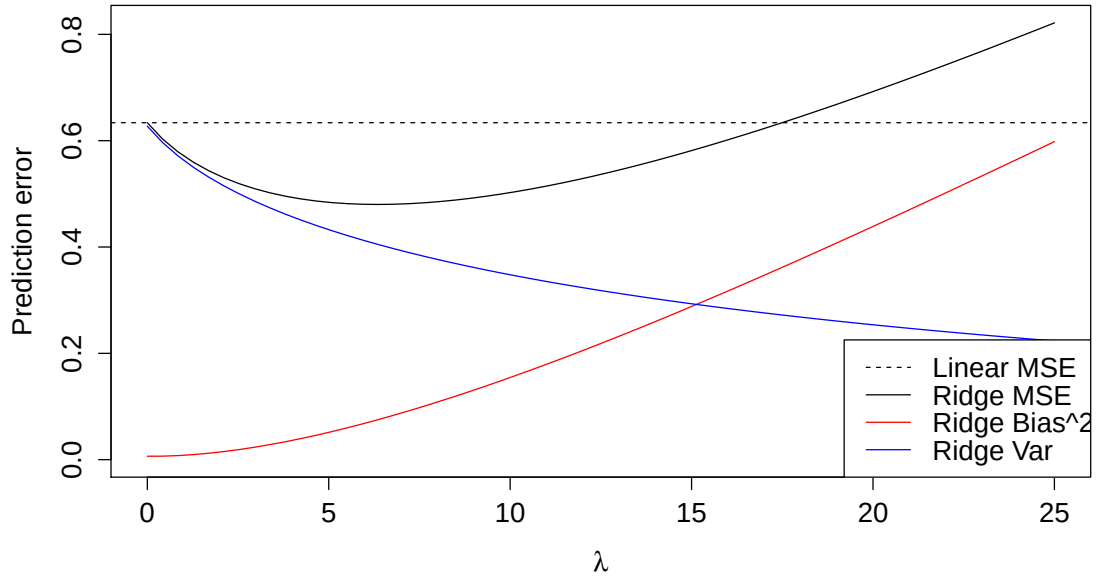


FIGURE 3.5. Bias-variance tradeoff depending on lambda. Dotted line corresponds to OLS prediction error (invariant with lambda). Black line corresponds to RR prediction error which can be decomposed in $Bias^2$ (in red) + Variance (in blue).

Since $Bias(\mathbf{X}(\lambda)) \neq 0$ this imply that

$$Bias(\mathbf{X}(\lambda))^2 > 0$$

we know that $\mathbf{A}^\top\mathbf{A}$ has an eigenvalue decomposition $\mathbf{A}^\top\mathbf{A} = \mathbf{V}\Sigma\mathbf{V}^{-1}$.

$$\begin{aligned}
Var(\mathbf{X}(\lambda)) &= \sigma^2\mathbf{W}(\mathbf{A}^\top\mathbf{A})^{-1}\mathbf{W}^\top \\
Var(\mathbf{X}(\lambda)) &= \sigma^2(\mathbb{I}+\lambda(\mathbf{A}^\top\mathbf{A})^{-1})^{-1}(\mathbf{A}^\top\mathbf{A})^{-1}((\mathbb{I}+\lambda(\mathbf{A}^\top\mathbf{A})^{-1})^{-1})^\top \\
&= \sigma^2(\mathbb{I}+\lambda(\mathbf{V}\Sigma\mathbf{V}^{-1})^{-1}(\mathbf{V}\Sigma\mathbf{V}^{-1})^{-1}((\mathbb{I}+\lambda(\mathbf{V}\Sigma\mathbf{V}^{-1})^{-1})^{-1})^\top \\
&= \sigma^2\mathbf{V}(\mathbb{I}+\lambda\Sigma^{-1})^{-1}\Sigma^{-1}(\mathbb{I}+\lambda\Sigma^{-1})^{-1}\mathbf{V}^{-1} \\
&= \sigma^2\mathbf{V}\mathbf{D}^{-1}\mathbf{V}^{-1}
\end{aligned}$$

where $\mathbf{D}^{-1} = (\mathbb{I}+\lambda\Sigma^{-1})^{-1}\Sigma^{-1}(\mathbb{I}+\lambda\Sigma^{-1})^{-1}$ is a diagonal matrix with coefficients:

$$d_i = \frac{\sigma_i^{-1}}{(1+\lambda\sigma_i^{-1})^2}$$

3.2.7. Machine learning algorithms. In recent years many successful machine learning applications have been developed. Artificial neural networks (ANN) and Support Vector Machines (SVM) have been some of the most popularly used machine learning algorithm [**Hay98**]. Historically, SVMs emerged after ANN.

The main characteristics of neural networks are that they have the ability to learn complex nonlinear input-output relationships, they use sequential training procedures they adapt themselves to the data.

ANN is inspired by biological learning systems organised into layers and have unidirectional connections between the layers, feed-forward networks (FFN) are the most used. FFN consist of a series of layers. The first layer has a connection from the network input. Each subsequent layer has a connection from the previous layer. The final layer produces the network's output. Figure 3.6 shows a FFN with its different layers.

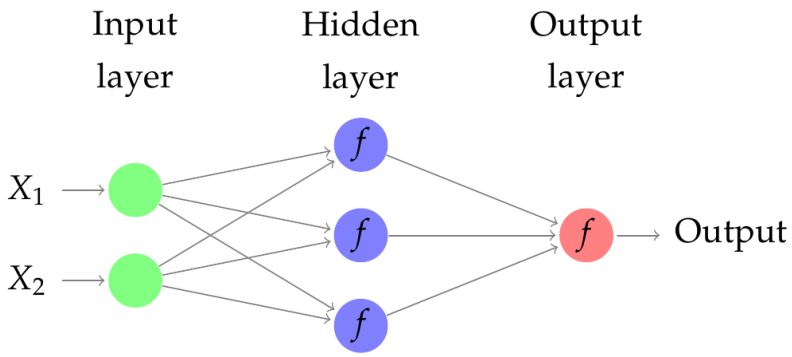


FIGURE 3.6. Feed-forward neural network (FFN). FNN consists of an input layer, an output layer and at least one hidden layer between the input and output layer.

Support Vector Machine (SVM) was introduced in 1992 by Vapnik and his coworkers [**BGV92**]. In its original form, SVM is a training algorithm for linear classification. Only later it was used for regression, principal component analysis, novelty detection and also for non-linear case. However, unlike ANN, it is very well based on theory in statistical learning [**CV95**]. SVM tunes the capacity of maximising the margin between the training patterns and the decision boundary. The solution is expressed as a linear combination of supporting patterns, which are the training patterns close to the decision boundary, called the support vectors.

The major difference between SVM and ANN is in the error optimisation. In ANN, the aim of learning is to obtain a set of weight values which minimise the training error minimum while training adjust the capacity of the machine.

For nonlinear case, SVM mapped the data sets of input space into a higher dimensional feature space, which is linear and the large margin learning algorithm is then applied. However, the mapping can be implicitly done by kernel functions. In the high dimensional feature space, simpler and linear hyper plane classifiers that have maximal margin between the classes can be obtained.

3.3. Online learning

Online learning is a supervised machine learning framework that is useful when we have sequential access to a sample only once. This differs from the classical batch learning, where there is an entire data set available and the learner can build the internal model without any limits in accessing the data. In batch learning, there is time enough to carefully analyse the dataset, build large predictive models and combine them in a sophisticated way. However, when execution time is critical and new data is required to be included in the model, online learning algorithms are a better option.

In batch learning, the training phase is commonly a computationally expensive process. Therefore, when new data arrives it can't be included easily to the model. Moreover, it could happen that we won't have enough time to process the new data before more data arrives. In online learning algorithms there is no training phase, but the model is updated and evaluated at every time step. This model updating is computationally less expensive than a training phase. Online algorithms allow incremental learning by processing one instance at a time. This is done updating the current model instead of building the model from scratch.

Online learning is also useful when some past data may be irrelevant or we want to improve computational efficiency. However, the use of less historical data could affect accuracy.

The goal of online learning is the same as batch learning, predicting targets as accurate as possible. For example, stock market prediction can be seen as online learning. The algorithm makes a prediction of the stock, little time after the real stock price is available, this information can be incorporated to the learner to further improve the prediction accuracy. In general, there is too much data available in an online learning setup, the data set grows continuously. Offline learning has equal or superior accuracy compared to online learning when the same amount of data is used.

Classic statistical theory of sequential prediction enforces strong assumptions on the statistical properties of the input sequence (for example, stationary stochastic process). However, these assumptions can be unknown or change over time. In online learning there is no previous assumption about the data and the sequence is allowed to be deterministic, stochastic or even adaptive.

Moreover, in case we receive data streams, ANN or SVM cannot introduce new information into the model without a re-training process, so we will have to use the same non-updated model until we decide to compute another one if it is possible. Online learning algorithms allow one example at a time to be introduced into an existing model incrementally [**VGS05**]. This is extremely important when the problem has large data streams and real-time forecasting must be done. This is the most common scenario when we want to forecast a wide range of data such as stock prices and volatilities, electricity power, intrusion detection, web-mining, server load, etc. Besides, many problems of

high interest in machine learning can be treated as online ones and they can also use these types of algorithms.

The online learning framework was first introduced in the perceptron algorithm [**Ros58**]. There are several other widely used online methods such as passive-aggressive [**CDK$^+$06**], stochastic gradient descent [**Zha04**], aggregating algorithm [**Vov01**] and the second order perceptron [**CBCG05**]. In [**CBL06**] an in-depth analysis of online learning is provided.

The motivation for online learning is to obtain computational efficiency and tackle the shifting problem, i.e. that the distribution of the data is unknown or changes over time. Online learning algorithms can deal with this problem because they have a tracking ability which is a strategy based on retaining weak dependence on past examples by using two types of models:

a) **memory boundedness:** consists of limiting the number of support vectors in order to improve computational efficiency. One example of this is the budget perceptron [**CKHS03**] which reduces the number of examples used for prediction. Alternatively, in the forgetron algorithm [**DSSS08**] the damage caused by removing old examples is discussed, which can be avoided by removing samples with small influences. Other examples are the sliding window kernel (RLS) [**VVVS06**], which only considers a sliding window of the most recent data, and in [**AS12**] is shown a variant of aggregating algorithm for regression [**Vov01**] considering only a sliding window of the most recent data, optimising also common matrix operations.

b) **weight decay:** one example of this is the shifting perceptron algorithm which implements an exponential decaying scheme for the examples [**CCBG07**]. Performance of an online learning algorithm is measured by the cumulative loss it suffers along its run on a sequence of examples. In order to minimise this loss, the learner may update the hypothesis after each round so as to be more accurate in later rounds.

There is sometimes confusion about online and incremental learning concepts. Incremental learning refers to any online learning process that learns the same model as would be learnt by a batch learning algorithm.

Incremental learning is useful when the input to a learning process is stream data, with the need or desire to be able to use the result of learning at any point in time, based on the input observations received so far.

Incremental learning is very useful when there is no need to record fundamental data and only a summary needs to be retained. Due to this, incremental algorithms are often characterised as memoryless, because no memory of past data is required. The algorithm is online but not incremental if it doesn't produce the same result for all observations that the corresponding batch algorithm would for these same observations.

Algorithm 1 shows the online learning algorithm structure:
where l is some loss function. Performance is later measured after T trials as:

$$L_T = \sum_{t=1}^{T} l_t(\mathbf{y}_t, \hat{\mathbf{y}}_t)$$

Algorithm 1 Structure of a Learning System

1: Receives input $\mathbf{x}_t$
2: Makes prediction $\hat{\mathbf{y}}_t$
3: Receives response $\mathbf{y}_t$
4: Incurs loss $l_t(\mathbf{y}_t, \hat{\mathbf{y}}_t)$

The objective is minimize this loss function for all instances. The quality of online learning algorithms is measured by a quantity known as regret which is the difference between the performance of the online algorithm and its optimal predictor $E^* \in \Theta$ given by:

$$L_T^* = \min_{E \epsilon \Theta} L_T^E \,,$$

where $L_T^E = \sum_{t=1}^{T} l_t(\mathbf{y}_t, \mathbf{y}_t^E)$ and $\mathbf{y}_t^E$ is the expert estimation.

Therefore regret is defined as:

$$R_T = L_T - L_T^*$$

3.3.1. Competitive analysis. Competitive analysis was designed for analysing the performance of an online algorithm compared with its optimal offline algorithm. An optimal offline algorithm can view the sequences of requests in advance. The effectiveness of an online algorithm [**ST85**] may be measured by its competitive ratio which is the worst-case ratio of the online algorithm and the optimal offline algorithm.

3.3.2. Online Ridge Regression. Online RR is the online formulation of the regularised Least Squares method and is based on the following equivalent formulation of the RR optimal solution.

Since

$$\mathbf{A} = \begin{bmatrix} — & \mathbf{a}_1^\top & — \\ — & \mathbf{a}_2^\top & — \\ & \vdots & \\ — & \mathbf{a}_m^\top & — \end{bmatrix} \quad \text{and} \quad \mathbf{B} = \begin{bmatrix} — & \mathbf{b}_1 & — \\ — & \mathbf{b}_2 & — \\ & \vdots & \\ — & \mathbf{b}_m & — \end{bmatrix} \tag{3.15}$$

Equation (3.2.4) can also be written as:

$$\begin{aligned} \mathbf{X}_{\text{ridge}} &= (\mathbf{A}^\top \mathbf{A} + \lambda \mathbb{I})^{-1} \mathbf{A}^\top \mathbf{Y} \\ &= \Big(\sum_{t=1}^{m} \mathbf{a}_t \mathbf{a}_t^\top + \lambda \mathbb{I} \Big)^{-1} \sum_{t=1}^{m} \mathbf{a}_t \mathbf{y}_t \,. \end{aligned}$$

Lets define $\mathbf{S} = \sum_{t=1}^{m} \mathbf{a}_t \mathbf{a}_t^\top + \lambda \mathbb{I}$ and $\mathbf{W} = \sum_{t=1}^{m} \mathbf{a}_t \mathbf{y}_t$, so the algorithm 2 shows the iterative formulation:

Algorithm 2 Online Ridge Regression

Require:
$\{\mathbf{a}_1, \ldots, \mathbf{a}_m\}$: m input vectors
$\{\mathbf{y}_1, \ldots, \mathbf{y}_m\}$: m target vectors
λ: regularization parameter
Ensure:
$\{f(\mathbf{a}_1), \ldots, f(\mathbf{a}_m)\}$: model predictions
Initialize $\mathbf{S} = \lambda\mathbb{I}$ and $\mathbf{W} = 0$
for $t = 1$ to m **do**
  read new $\mathbf{a}_t$
  $\mathbf{X} = \mathbf{S}^{-1}\mathbf{W}$
  output prediction $f(\mathbf{a}_t) = \mathbf{X}^\top \mathbf{a}_t$
  $\mathbf{S} = \mathbf{S} + \mathbf{a}_t\mathbf{a}_t^\top$
  Read new y_t
  $\mathbf{W} = \mathbf{W} + \mathbf{a}_t\mathbf{y}_t$
end for

3.3.3. The Aggregating Algorithm for Regression. The AAR, proposed by [**Vov01**] (also known as Vovk-Azoury-Warmuth predictor [**AW01**]), is an application of the aggregating algorithm to the problem of regression. The idea is introduce the new input vector $\mathbf{x}_{m+1}$ to solve the model parameters:

$$\mathbf{X}_{aar} = \Big(\sum_{t=1}^{m+1} \mathbf{a}_t\mathbf{a}_t^\top + \gamma\mathbb{I} \Big)^{-1} \sum_{t=1}^{m} \mathbf{a}_t\mathbf{y}_t \,. \tag{3.16}$$

If we define $\mathbf{S} = \sum_{t=1}^{m+1} \mathbf{a}_t\mathbf{a}_t^\top + \gamma\mathbb{I}$ and $\mathbf{W} = \sum_{t=1}^{m} \mathbf{a}_t\mathbf{y}_t$, the algorithm 3 is slightly different to the algorithm 2, which updated matrix $\mathbf{S}$ before making the prediction:

3.3.4. Efficient computation. In regression problems we always require getting a matrix inverse which is computationally expensive. However, in stream data problems there is a way to obtain a matrix inverse approximation using the Sherman-Morrison-Woodbury. This formula allows to get the inverse of a matrix $\mathbf{A} + \mathbf{u}\mathbf{v}^\top$ if we previously calculated the inverse of $\mathbf{A}$ as follows:

$$(\mathbf{A} + \mathbf{u}\mathbf{v}^\top)^{-1} = \mathbf{A}^{-1} - \frac{\mathbf{A^{-1}}\mathbf{u}\mathbf{v}^\top\mathbf{A^{-1}}}{\mathbf{1} + \mathbf{v}^\top\mathbf{A^{-1}}\mathbf{u}} \tag{3.17}$$

Alternatively, Coleman and Sun [**CS10**] presented an iterative algorithm which uses $\mathbf{X}(\lambda)$ to approximate $\mathbf{X}(0)$.

Using the compact SVD (shown in equation (3.11)) $\mathbf{X}(\lambda)$ is expressed as follows:

Algorithm 3 *The aggregating algorithm for regression*

Require:

$\{\mathbf{a}_1, \ldots, \mathbf{a}_m\}$: m input vectors

$\{\mathbf{y}_1, \ldots, \mathbf{y}_m\}$: m target vectors

λ: regularisation parameter

Ensure:

$\{f(\mathbf{a}_1), \ldots, f(\mathbf{a}_m)\}$: model predictions

1: Initialize $\mathbf{S} = \lambda\mathbb{I}$ and $\mathbf{W} = 0$
2: **for** $t = 1$ to m **do**
3: read new $\mathbf{a}_t$
4: $\mathbf{S} = \mathbf{S} + \mathbf{a}_t\mathbf{a}_t^\top$
5: $\mathbf{X} = \mathbf{S}^{-1}\mathbf{W}$
6: output prediction $f(\mathbf{a}_t) = \mathbf{X}^\top\mathbf{a}_t$
7: Read new $\mathbf{y}_t$
8: $\mathbf{W} = \mathbf{W} + \mathbf{a}_t\mathbf{y}_t$
9: **end for**

$$\begin{aligned} \mathbf{X}(\lambda) &= (\mathbf{A}^\top\mathbf{A} + \lambda\mathbb{I})^{-1}\mathbf{A}^\top\mathbf{Y} \\ &= \mathbf{V}_1(\Sigma_1^2 + \lambda\mathbb{I})^{-1}\boldsymbol{\Sigma}_\mathbf{1}\mathbf{U}_\mathbf{1}^\top\mathbf{Y} \end{aligned} \tag{3.18}$$

where is easy to see that $\mathbf{X}(\lambda) \to \hat{\mathbf{X}} = \mathbf{V}_1\boldsymbol{\Sigma}_1^{-1}\mathbf{U}_1^\top\mathbf{Y}$ as $\lambda \to 0$.

The method consists in obtaining $\mathbf{X}(\lambda)$ and then refine by adding more terms of its Taylor expansion to approximate $\mathbf{X}(0) = \hat{\mathbf{X}}$. The Taylor expansion about λ_0 is:

$$\mathbf{X}(\lambda) = \mathbf{X}(\lambda_0) + \sum_{k=1}^{\infty} \mathbf{s}_k(\lambda - \lambda_0)^k \tag{3.19}$$

where $\mathbf{s}_k = \frac{1}{k!}\mathbf{X}(\lambda)^{(k)}$ and $\mathbf{X}(\lambda)^{(k)}$ is obtained by taking differences $\frac{\partial}{\partial\lambda}$ of:

$$\begin{aligned} (\mathbf{A}^\top\mathbf{A} + \lambda\mathbb{I})\mathbf{X}(\lambda) &= \mathbf{A}^\top\mathbf{Y} \\ (\mathbf{A}^\top\mathbf{A} + \lambda\mathbb{I})\mathbf{X}(\lambda)^{(1)} + \mathbf{X}(\lambda) &= 0 \\ \mathbf{X}(\lambda)^{(1)} &= -(\mathbf{A}^\top\mathbf{A} + \lambda\mathbb{I})^{-1}\mathbf{X}(\lambda) \\ \mathbf{X}(\lambda)^{(2)} &= -2(\mathbf{A}^\top\mathbf{A} + \lambda\mathbb{I})^{-1}\mathbf{X}(\lambda)^{(1)} \\ &\vdots \\ \mathbf{X}(\lambda)^{(k)} &= -k(\mathbf{A}^\top\mathbf{A} + \lambda\mathbb{I})^{-1}\mathbf{X}(\lambda)^{(k-1)} \end{aligned}$$

for $\lambda = 0$ we have:

$$\mathbf{X}(0) = \mathbf{X}(\lambda_0) + \sum_{k=1}^{\infty} (-1)^k \mathbf{s}_k \lambda_0^k \tag{3.20}$$

therefore in order to ensure convergence, we can see thatλ_0 selection cannot be large.

The algorithm for computing $\mathbf{X}(0)$ is the following:

Algorithm 4 Algorithm for handling rank deficient matrices

Require:

$\mathbf{A}$: design matrix

$\mathbf{Y}$: response matrix

λ: rank deficient parameter

Ensure:

$\mathbf{X}$: parameters

$\mathbf{M} = \mathbf{A}^\top \mathbf{A}$

Initialize $\mathbf{QR} = \mathbf{M}$

$\mathbf{X} = \mathbf{R}^{-1}\mathbf{Q}^\top \mathbf{Y}$

$\mathbf{s} = \mathbf{X}$

for $i = 1, 2, 3, \ldots$ **do**

  $\mathbf{s} = -(\mathbf{M} + \lambda \mathbb{I})^{-1}\mathbf{s}$

  $\mathbf{X} = \mathbf{X} + (-1)^i \mathbf{s}\lambda^i$

end for

The algorithm 4 solves equation (3.20). However, this version is unstable since typically $\|\mathbf{s}\|$ is very large and λ^i is very small (λ is small).

The following algorithm shows a more stable version of algorithm 4.

Algorithm 5 Algorithm for handling rank deficient matrices improved

Require:

$\mathbf{A}$: design matrix

$\mathbf{Y}$: response matrix

λ: rank deficient parameter

Ensure:

$\mathbf{X}$: parameters

$\mathbf{M} = \mathbf{A}^\top \mathbf{A}$

Initialize $\mathbf{QR} = \mathbf{M}$

$\mathbf{X} = \mathbf{R}^{-1}\mathbf{Q}^\top \mathbf{Y}$

$\mathbf{t} = \mathbf{X}$

for $i = 1, 2, 3, \ldots$ **do**

  $\mathbf{t} = \lambda \mathbf{t}$

  $\mathbf{t} = -(\mathbf{M} + \lambda \mathbb{I})^{-1}\mathbf{t}$

  $\mathbf{X} = \mathbf{X} + \mathbf{t}$

end for

Both algorithms are equivalent, but algorithm 5 is more stable and converges in typically less than 10 steps. It is important to notice that the QR factorisation is computed only once and it is computationally less expensive than the SVD.

Despite the bias-variance tradeoff provides a conceptual framework for determining a good model, is not directly useful. Some popular methods for determine model selection are:

- Akaike information criterion (AIC) (Akaike, 1974)
- Bootstrap-based selection (Efron and Tibshirani, 1997)
- Cross-validation (Stone, 1974)

On the other hand, λ can also be used to reduce computational time as it is shown in equation (3.20). Golub ([**Gol65**]) suggested that λ be chosen so that:

$$\frac{\lambda}{\lambda + \delta^2} < 0.1$$

where δ is a lower bound of the smallest non-zero singular value σ_k. One approach is to choose the greatest lower bound for $\delta = \sigma_k$. Hence:

$$\frac{\lambda}{\lambda + \sigma_k^2} < 0.1$$

if we choose $\lambda = \beta\sigma_k^2$ then we get the following relation:

$$\frac{\beta\sigma_k^2}{\beta\sigma_k^2 + \sigma_k^2} = \frac{\beta}{\beta + 1} < 0.1 \Rightarrow \beta < \frac{1}{9}$$

Experiments done by [**CS10**] have shown that $\beta = 0.01$ produces satisfactory results. This means that $\lambda = 0.01\sigma_k^2$.

However, get σ_k imply obtaining first the SVD, which is computational expensive.

Since the algorithm presented by [**CS10**] already compute the QR factorization of the matrix $\mathbf{A}$ they show a procedure for getting a λ approximation using QR.

(1) Compute QR factorization: $\mathbf{A} = \mathbf{Q_1}\mathbf{R_1}$
(2) Let $\mathbf{W}$ denote the set of absolute values of the nonzero diagonal elements of $\mathbf{R_1}$. Let $w_{\min}$ and $w_{\max}$ denote the smallest and largest elements of $\mathbf{W}$ respectively. Both $\lambda_1 = \hat{\beta} w_{\min}^2$ $\lambda_2 = \hat{\beta}\frac{w_{\min}^2}{w_{\max}^2}$ where $\hat{\beta} = 0.00025$ produce satisfactory results.
(3) The λ_{QR} approximation is obtained as: $\lambda_{\text{QR}} = \frac{\lambda_1 + \lambda_2}{2}$

3.4. Evaluation methods

Forecast performance is evaluated using different methods. We have chosen three measures usually used:

MAPE: Mean Average Percent Error which presents forecast errors as a percentage.

$$\text{MAPE} = \frac{1}{N}\sum_{t=1}^{N}\frac{|\mathbf{y}_t - \hat{\mathbf{y}}_t|}{|\mathbf{y}_t|} \times 100 \tag{3.21}$$

MAE: Mean Average Error which measures the distance between forecasts to the true value.

$$\text{MAE} = \frac{1}{N}\sum_{t=1}^{N}|\mathbf{y}_t - \hat{\mathbf{y}}_t| \tag{3.22}$$

RMSE: Root Mean Square Error also measures the distance between forecasts to the true values but, unlike MAE, large deviations from the true value have a large impact on RMSE due to squaring forecast error.

$$\text{RMSE} = \sqrt{\frac{\sum_{t=1}^{N}(\mathbf{y}_t - \hat{\mathbf{y}}_t)^2}{N}} \tag{3.23}$$

3.5. Model selection

Akaike Information Criterion (AIC) is often used in model selection where AIC with smaller values are preferred since they represent a trade-off between bias and variance. AIC is obtained as follows:

$$AIC = \underbrace{-\frac{2l}{N}}_{\text{bias}} + \underbrace{\frac{2k}{N}}_{\text{variance}} \tag{3.24}$$

where

l: is the loglikelihood function
k: number of estimated parameters
N: number of observations

Loglikelihood function is obtained from the Residual Sum of Squares (RSS):

$$l = -\frac{N}{2}\left(1 + ln(2\pi) + ln\left(\frac{RSS}{N}\right)\right) \tag{3.25}$$

Chapter 4

Adaptive parallel algorithm for Vector error correction model

VECM (presented in section 2.6) has been broadly used with low frequency time series. We aimed to adapt VECM to be used in finance with high frequency stream data.

We observed that the number of cointegration relationships varies with the length of historical data used. Moreover, parameters that increased these relationships in time led to better forecasting performance. Our proposal, called an Adaptive VECM (AVECM) is to make a parameters grid search that maximises the number of cointegration relationships in the near past. To ensure the search can be executed fast enough, we used a distributed environment.

The methodology was tested using four 10-second frequency time series of the Foreign Exchange market. We compared our proposal with ARIMA and the naive forecast of the random walk model. Numerical experiments showed that on average AVECM performed better than ARIMA and random walk. Additionally, AVECM significantly improved execution times with respect to its serial version.

4.1. Introduction

In finance, it is common to find variables with long-run equilibrium relationships. This is termed cointegration and reflects the idea that some sets of variables cannot wander too far from each other. Cointegration means that one or more linear combinations of these variables are stationary even though individually they are not. Some

models, such as the Vector Error Correction (VECM), see [**EG87a**], take advantage of this property and describe the joint behaviour of several cointegrated variables. VECM introduces this long-run relationship among a set of cointegrated time series as an error correction term. These time series must be integrated of order 1, denoted I(1), i.e. they become stationary at their first differences. In finance, I(1) time series are very common and to introduce cointegration restrictions in models often improves forecasting, see [**DT98**]. Therefore, VECM has been widely adopted in financial applications, among others: [**MN95**], [**SAZ13**], [**MK00**] and [**ADL01**]. VECM has also been used in pair trading, see [**Her03**], or models with more than two variables, see for example [**MN95**] and [**EP04**]).

VECM parameters are obtained using the ordinary least squares (OLS) method, developed by [**GVL80**]. Since OLS involves many calculations, the parameter estimation is computationally expensive when the number of lagged values and data increases. Moreover, obtaining cointegration vectors is also an expensive routine. This is a main limitation to use VECM with stream data with high frequency. [**?**] addresses the advantage of distributed processing over conventional rolling window processing.

Our aims were to study if a parallel version of VECM can be used with high frequency stream data. Our approach was to determine adaptively the number of observations and lags of VECM which maximise cointegration relations in the past. This search is computationally expensive because the Johansen method, see [**Joh95**], is required to find cointegration vectors, which is of order $O(n^3)$. Therefore, our proposal, called Adaptively Vector Error Correction (AVECM), is to parallelise this search in order to get new parameters before new data arrives. Model effectiveness is focused on out-of-sample forecast rather than in-sample fitting. This criterion allows AVECM prediction capability to be expressed rather than just explaining data history. The forecast capability of our method was measured using MSE and the Theil's U-statistic, see [**?**], widely used in economic forecast. Tests were run using four currency rates: Euro (EUR) to United States Dollar (USD) (EURUSD), British Pound (GBP) to USD (GBPUSD), USD to Swiss Franc (CHF) (USDCHF) and USD to Japanese Yen (JPY) (USDJPY) with a 10-second frequency.

4.2. Methodology

4.2.1. Motivation. Cointegration vectors can be found applying the Johansen method which uses a sample of the last historical data. However, VECM assumes cointegration vectors do not change in time. In fact, [**GNW96**] addresses that the long-run relationships between the time series might change due to several economic factors that can lead to structural breaks in the cointegration relationship. In order to show that the number of cointegration vectors depends on the amount L of historical

data and the number of lags p in the VECM, we used a grid search. We defined a grid of possible values for L and p. L goes throughout $[2, 14]$ hours (1 *hour* = 360 data points) with a step size of 4 hours and p throughout $[1, 5]$ with step size of 1. The first experiment consisted in determining the number of cointegration vectors for all combinations of L and p. We used four forex rates: EURUSD, GBPUSD, USDCHF and USDJPY with 10-second frequency. Data started at 13:00 GMT of the 13th of August 2014, when the New York and London financial markets opened.

Figure 4.1 shows the distribution of the number of cointegration vectors given by the Johansen method for different values of $L = [2, 6, 10, 14]$ hours and $p = 1$. This procedure was carried out by a sliding window of historical data moving 1000 times.

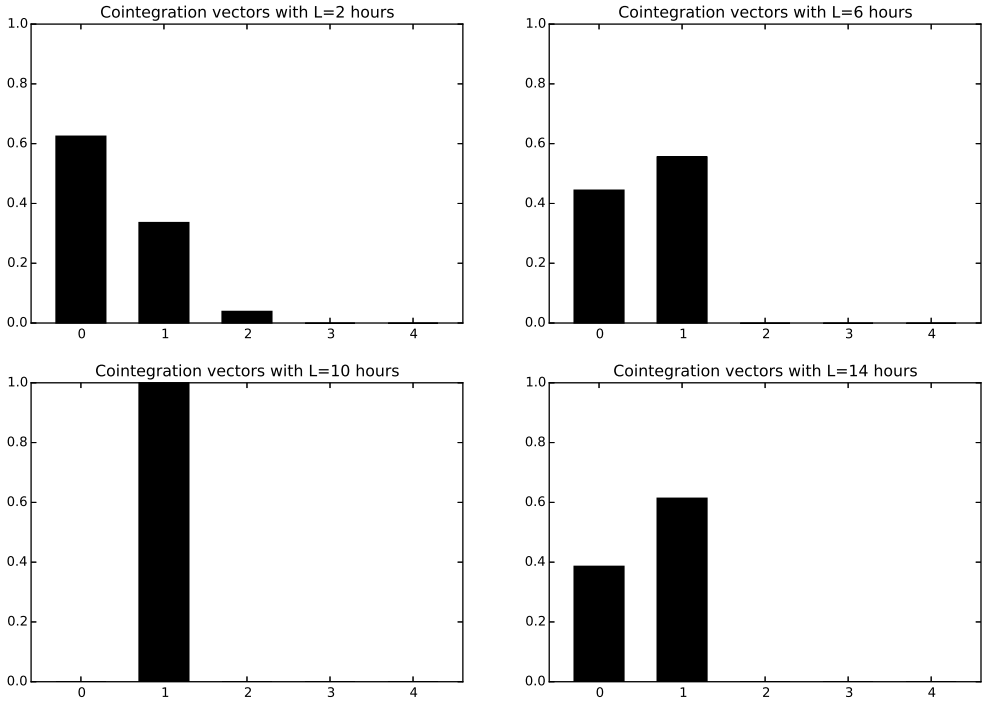


FIGURE 4.1. Distribution of the number of cointegration vectors using $p = 1$ lags. Four possible values for windows size L are shown (2, 6, 10 and 14) hours (1 hour = 360 data points).

From section **??** we know that $r = 0$ means no cointegration and $r = l$ (we are using four rates, so $l = 4$) reveals that no process is I(1) but stationary. The interesting cases of cointegration are those where r lies strictly between 0 and 4, i.e. $0 < r < 4$.

In order to measure the extent of cointegration, we introduce a *percentage of cointegration* as following:

$$PC = \frac{\#\{it \mid it \text{ has } r \text{ c.v. with } 0 < r < l\}}{\#it} \times 100 \tag{4.1}$$

where c.v. stands for cointegration vectors and it is the number of iterations.

The goal of our next experiment was to find a relation between this ratio PC and the performance measure MSE (see equation 4.4). L was defined between $[700, 1500]$ data points with step size 100 and p took values between $[1, 5]$ with step size 1.

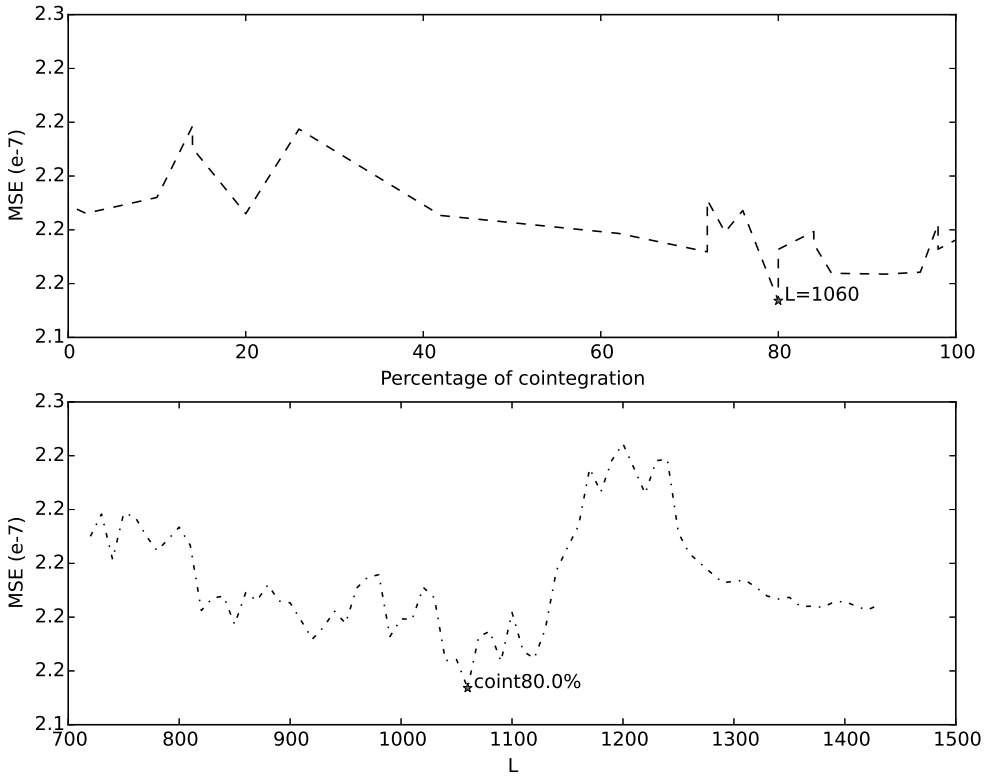


FIGURE 4.2. MSE versus the percentage of cointegration considering 1000 iterations. Best windows size L found was 1060. Below MSE versus L shows a rapidly decreasing behaviour, founding minimum at PC = 80%

Figure 4.2 shows the relation between MSE and PC and L. We found that better cointegration percentage leads to better performance accuracy in terms of reducing MSE.

Therefore we proposed to choose L and p in order to maximise the percentage of cointegration PC in the near past. This process was done every time that new data was processed. However, this search can be slow if we try different values for L and p and therefore we distributed this calculation in order to reduce searching time.

Our proposal is then a modified version of VECM, with parameter p and the amount of historical data used L obtained at every step. Only the search of L and p was done in a distributed environment, since this is the most expensive routine. Our proposal is called Adaptive Vector Error Correction model (AVECM). AVECM is detailed in the algorithm 6 which summarises our proposal.

Algorithm 6 AVECM: Adaptive VECM.

Require:

$\mathbf{y}$: matrix with N input vectors and l time series
j: Starting point of testing
it: Ending point of testing
ps: list of p values
Ls: list of L values ($L < N$)
m: Iterations to determine parameters ($m < N - L$)

Ensure:

$\{\hat{\mathbf{y}}[1], \ldots, \hat{\mathbf{y}}[it]\}$: prediction vectors

1: **for** $i = j$ to it **do**
2: $\mathbf{Y} \leftarrow \mathbf{y}[:, i-1]$
3: $L, p \leftarrow$ `get_best_params`$(Ls, ps, m, \mathbf{Y})$
4: $model = \text{VECM}(\mathbf{Y}, L, p)$
5: $\hat{\mathbf{y}}[i-j] = model.predict()$
6: **end for**

The input of AVECM is time series prices which are cointegrated. The starting point of testing is j and the total number of iterations is it. We need to ensure that j is at least the maximum value of Ls. Ls and ps are the possible values for L and p. The function **get_best_params** makes this grid search on the two vector lists Ls and ps and returns the parameters L and p which maximise the percentage of cointegration PC (see equation 4.1) for a pre-defined number of iterations m. This function is implemented in a distributed environment, thus ensuring a response before new data is available. After L and p parameters are found, VECM is built and used to forecast the next data point.

4.2.2. Model comparison. We compared our proposal, in terms of performance, with the naive forecast of the random walk model and ARIMA. It is still difficult to outperform the random walk model for standard econometric forecasting models despite its simplicity, see [**LM11**]. The random walk model is defined as:

$$\mathbf{y}_t = \mathbf{y}_{t-1} + \epsilon_t \tag{4.2}$$

The naive forecast of the time series difference $\hat{\mathbf{y}}_{t+1}$ for the random walk model is defined as:

$$\hat{\mathbf{y}}_{t+1} = \mathbf{y}_t + \hat{\epsilon}_{t+1} \tag{4.3}$$

where $\hat{\epsilon}_{t+1} = \epsilon_t$.

On the other hand, ARIMA is widely used to forecast returns in finance, see [**Tsa05**]. A process can be modelled as an ARIMA(p, d, q) model if $\mathbf{x}_t = \Delta^d \mathbf{y}_t$, i.e after differencing d times the time series $\mathbf{y}_t$, we get an ARMA(p, q). Since we are modelling returns, we used $d = 1$.

4.2.3. Evaluation methods. Forecast performance was evaluated using two different methods which are frequently used in finance:

: MSE, Mean Square Error measures the distance between forecasts and the true values and large deviations from the true value have a large impact due to squaring forecast error.

$$\text{MSE} = \frac{\sum_{t=1}^{N}(\mathbf{y}_t - \hat{\mathbf{y}}_t)^2}{N} \tag{4.4}$$

: U-statistic, the Theil's U-statistic, presented by [**?**], is a unit free measure obtained as the ratio between the root MSE (RMSE) of a model and the RMSE of the naive random walk model. A U-statistic less than 1 implies the performance is better than the naive model.

4.3. Experimental results

4.3.1. Data. AVECM tests were carried out using four foreign exchange rates all related to USD: EURUSD, GBPUSD, USDCHF and USDJPY. This data was collected from [**Duk14**], a free database which gives access to the Swiss Foreign Exchange marketplace.

The tests were done using 10-second frequency from ask prices from the 11th to the 15th of August 2014. Since one day corresponds to 8640 data points and we used 5 days of data, we have 43,200 data points in total.

4.3.2. Unit root tests. Before running the tests, we firstly checked whether the time series were I(1) using the Augmented Dickey Fuller (ADF) test with lags $p = 1, 2, 3, 4, 5$. [**?**] presented critical values for rejection of hypothesis of a unit root: -2.62 (1%), 1.94 (5%) and 1.62 (10%). Table 1 shows that all currency rates cannot reject the unit root test in their level form considering different lags, but they rejected it with their first differences. This means that all of them are I(1) time series and we are allowed to use VECM and therefore our proposed AVECM.

4.3.3. Performance accuracy. Algorithms AVECM, ARIMA and the naive random walk were tested using four days of data (from the 12th to the 15th of August 2014). For AVECM we considered different number of iterations (parameter m in algorithm 6): 10, 50 and 100. We tried 12, 24 and 47 different pair of combinations for L and p. Possible values for L were in the interval $[2, 14]$ hours and p can have values in between $[1, 5]$. Best AVECM performance was compared against ARIMA and the random walk model. Table 4 shows the out-of-sample performance measures: MSE and U-statistic for AVECM and ARIMA. In terms of both measures we found that AVECM is superior to ARIMA and the naive random walk model. We also included the p-value

Variable	ADF(1)	ADF(2)	ADF(3)	ADF(4)	ADF(5)
EURUSD	-0.052	-0.054	-0.054	-0.054	-0.054
GBPUSD	-0.744	-0.784	-0.805	-0.837	-0.846
USDCHF	-0.476	-0.493	-0.493	-0.495	-0.502
USDJPY	0.357	0.360	0.360	0.367	0.367
Δ EURUSD	-128.4*	-128.4*	-96.85*	-89.12*	-89.12*
Δ GBPUSD	-131.4*	-112.7*	-102.5*	-92.86*	-88.29*
Δ USDCHF	-127.8*	-127.8*	-96.94*	-88.82*	-80.79*
Δ USDJPY	-135.1*	-135.1*	-101.2*	-101.2*	-101.2*

* Indicates significance at 1% level
** Indicates significance at 5% level
*** Indicates significance at 10% level
MacKinnon critical values for rejection of hypothesis of a unit root are: -2.62 (1%), -1.94 (5%) and -1.62 (10%)
ADF(d) Augmented Dickey-Fuller test with lag d

TABLE 1. Unit roots tests for EURUSD, GBPUSD, USDCHF and USDJPY at 10-second frequency.

that proves that the difference in the MSE is significant at the 99% significance level in three of the four currency rates and at 90% in the case of GBPUSD. The U-statistic shows that AVECM and ARIMA are superior to the naive random walk model and that our proposal is also superior to ARIMA.

TABLE 2. AVECM performance

	MSE			U-statistic	
	AVECM	ARIMA	p-value	AVECM	ARIMA
EURUSD	1.0702 e-09	1.1481 e-09	9.2509 e-12	0.6863	0.7108
GBPUSD	1.6630 e-09	1.7408 e-09	6.9519 e-02	0.6866	0.7025
USDCHF	5.8503 e-10	6.3545 e-10	2.8999 e-14	0.6803	0.7091
USDJPY	6.3483 e-06	6.5194 e-06	6.8536 e-05	0.6964	0.7057

4.3.4. Parallel implementation. To determine L and p based on maximising the percentage of cointegration requires use of the Johansen method which is a computationally expensive routine. This procedure is done by the function get_best_params($Ls, ps, m, \mathbf{Y}$) shown in algorithm 6. In order to improve the execution time of this search, our proposal included a parallel search of VECM parameters using high performance computing. The

main objective was to obtain a response before a new data arrived in the following 10 seconds.

The Johansen method is already programmed in the Python Statsmodels library, see [**SP10**], and the parallel implementation was developed using MPI in Python. We chose MPI because it allows large-scale parallel applications with wide portability to be built, being able to run in large clusters or on local computers. We tested our proposal in a cluster with 2 servers Xeon E5-2667 (2.90GHz) of 24 cores each (48 cores in total) and 24GB RAM. In order to compare serial and parallel execution times in AVECM, we set parameter $it = 100$ in algorithm 6.

The L parameter was always chosen between 2 and 14 hours and p always took values between 1 and 5. Parameter $nparams$ represents the number of pairs (L,p) used to maximise the percentage of cointegration.

Execution time depends directly on L, p and $nparams$ used, since they determine the size of matrix $\mathbf{A}$ (see equation **??**) and therefore affect the OLS function execution time. Therefore, if we try more combinations of L and p (increasing $nparams$) the serial algorithm will take longer. For this financial time series we are interested in execution times below 10 seconds (the time series frequency).

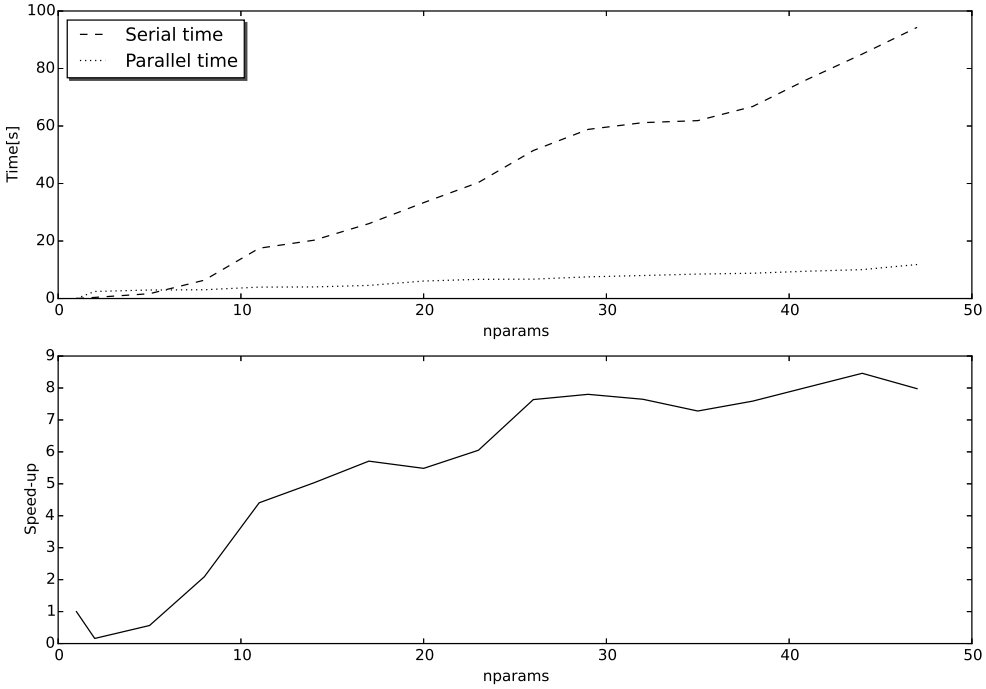


FIGURE 4.3. Computing time of sequential and parallel algorithm is shown in the upper figure. Speed-up is shown below.

The superior figure in 4.3 shows that best performance accuracy measurements are achieved in times near or below 10 seconds in the parallel version. Contrarily, serial

times are higher, above 10 seconds in most cases. Figure 4.3 also shows the speed-up in computing time for AVECM which is near 9X if we use more than 25 parameters ($nparams$).

Execution times do not consider the loading data time, just that of finding best parameters and matrix operations. The time MPI spends transferring data and synchronising processes is about two seconds independently of the number of processes considered. Execution times were measured using the Time python library.

4.4. Conclusions

Cointegration in financial time series has been largely studied and the Johansen method is commonly used to obtain cointegration relationships. In practice, it has been found that cointegration relations change with time. However, model-based cointegration such as VECM assumes that cointegration remains unchanged in time. We empirically showed that the Johansen method is sensitive to the number of lags but also to the amount of data considered.

Moreover, we introduced the notion of *percentage of cointegration* and found that out-of-sample forecast performance MSE is related to the value of this figure in the last samples. We used this information to set the model parameters. Our proposal AVECM consists of an adaptive algorithm to update VECM parameters every time that new data is available. These parameters are found by maximising the percentage of cointegration of the last samples or iterations.

Despite the fact that high frequency Forex data can be spurious, the model performance can be less reliable (and more spurious) relative to the lower frequencies (such as 1 minute or 5 minute intervals) adopted by some other studies. However, the deficiency is offset by gain in accuracy from parallel processing which is capable of searching or examining a much larger state space given the same computational time.

Determining VECM parameters was the most expensive routine and it was run using parallel processes using MPI which allowed a grid search within a range of values for L and p to be made. Tests were done using real currency rates data.

Results showed that our proposed AVECM improves performance measures by finding parameters of L and p maximising the percentage of cointegration.

The parallel implementation allowed the execution times to be reduced more than 9 times and therefore a response time was obtain before 10 seconds. Since we used 10-second frequencys we can say that our proposal is suitable for use in an online context for real applications because response times were less than this frequency.

For future study, it would be interesting to explore the relationship between cointegration and performance in order to propose new criteria for improving VECM parameters. It would also be interesting to include more explaining variables such as bid-ask spread and change in volume.

Chapter 5

Online VECM proposal

Financial time series are known for their non-stationary behaviour. However, sometimes they exhibit some stationary linear combinations. When this happens, it is said that those time series are cointegrated. The Vector Error Correction Model (VECM) is an econometric model which characterises the joint dynamic behaviour of a set of cointegrated variables in terms of forces pulling towards equilibrium. In this study, we propose an Online VEC model (OVECM) which optimises how model parameters are obtained using a sliding window of the most recent data. Our proposal also takes advantage of the long-run relationship between the time series in order to obtain improved execution times. Our proposed method is tested using four foreign exchange rates with a frequency of 1-minute, all related to the USD currency base. OVECM is compared with VECM and ARIMA models in terms of forecasting accuracy and execution times. We show that OVECM outperforms ARIMA forecasting and enables execution time to be reduced considerably while maintaining good accuracy levels compared with VECM.

5.1. The problem

In finance, it is common to find variables with a long-run and/or a short-run equilibrium relationship. This relationship is called cointegration and it reflects the idea of that some set of variables cannot wander too far away from each other.

Cointegration means that one or more combinations of these variables is stationary even though individually they are not.

Some financial models, such as vector error correction model (VECM), take advantage of this property and describe the joint behaviour of several cointegrated financial instruments.

VECM is a special case of the vector autoregressive model (VAR). VAR is a linear model which express future values in terms of its own history and other variables past values. If cointegration exists, VECM allows to introduce an error correction term due to its cointegration estimation error.

VECM as well as VAR model parameters are obtained using ordinary least squares method (OLS). Since OLS involves many calculations, the parameter estimation method is computationally expensive when the number of past values and observations considered increases. Moreover, OLS is an ill-posed problem which admits an infinite number of solutions.

Ridge regression method (RR) tackles this ill-posed problem and is usually formulated instead of OLS. RR includes a regularisation parameter that leads to better generalisation capability. However, RR is still computationally expensive dealing with large datasets. Recently, online learning algorithms have been proposed to solve problems with large data sets because of their simplicity and their ability of updating the model when new data is available.

We propose an online formulation of the VEC model based on the AAR method considering only a sliding window of the most recent data. The algorithm introduces matrix optimisations in order to reduce the number of operations. Our method is later tested with financial data from the foreign exchange market.

5.2. Methodology

Since VECM parameter estimation is computationally expensive, we propose an online version of VECM (OVECM). OVECM considers only a sliding window of the most recent data. Moreover, since cointegration vectors represent long-run relationships which vary little in time, OVECM determines firstly if they require calculation.

OVECM also implements matrix optimisations in order to reduce execution time, such as updating matrices with new data, removing old data and introducing new cointegration vectors.

Algorithm 10 shows our OVECM proposal which considers the following:

- The function `getJohansen` returns cointegration vectors given by the Johansen method considering the trace statistic test at 95% significance level.
- The function `vecMatrix` returns the matrices (2.26) and (2.23) that allows VECM to be solved.
- The function `vecMatrixOnline` returns the matrices (2.26) and (2.23) aggregating new data and removing the old one, avoiding calculation of the matrix $\mathbf{A}$ from scratch.
- Out-of-sample outputs are saved in the variables $\mathbf{Y}_{\text{true}}$ and $\mathbf{Y}_{\text{pred}}$.
- The model is solved using OLS.
- In-sample outputs are saved in the variables $\Delta\mathbf{y}_{\text{true}}$ and $\Delta\mathbf{y}_{\text{pred}}$.

- The function `mape` gets the in-sample MAPE for the l time series.
- Cointegration vectors are obtained at the beginning and when they are required to be updated. This updating is decided based on the in-sample MAPE of the last n inputs. The average of all MAPEs are stored in the variable e. If the average of MAPEs (`mean`(e)) is above a certain error given by the mean_error threshold, cointegration vectors are updated.
- If new cointegration vectors are required, the function `vecMatrixUpdate` only updates the corresponding columns of matrix $\mathbf{A}$ affected by those vectors (see equation 2.26).

Algorithm 7 OVECM: Online VECM

Require:

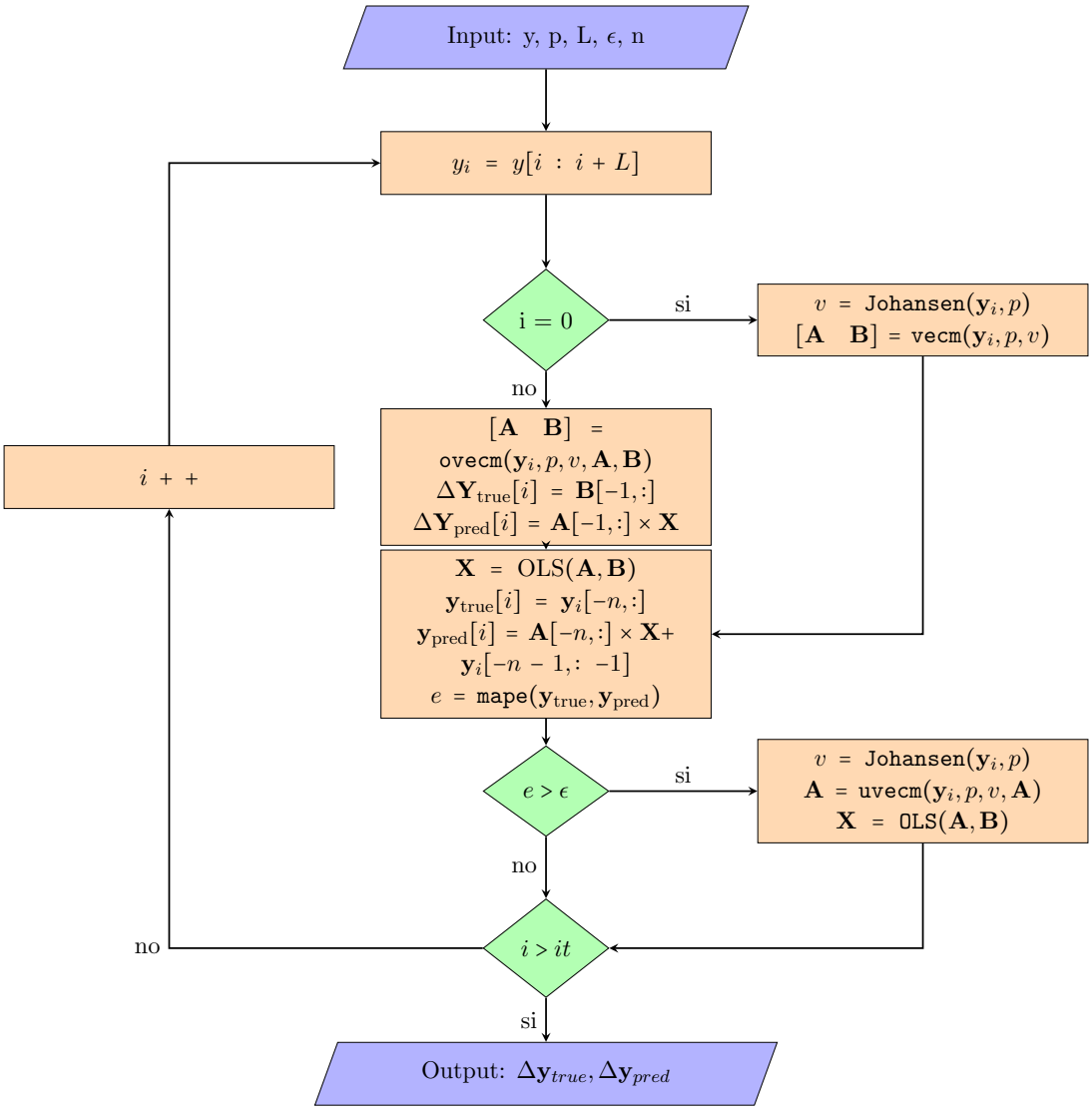
$\mathbf{y}$: matrix with N input vectors and l time series
p: number of past values
L: sliding window size ($L < N$)
mean_error: MAPE threshold
n: interpolation points to obtain MAPE

Ensure:

$\{\mathbf{y}_{\text{pred}}[L+1], \ldots, \mathbf{y}_{\text{pred}}[N]\}$: model predictions

for $i = 0$ to $N - L$ **do**
  $\mathbf{y}_i \leftarrow \mathbf{y}[i : i + L]$
  if i $= 0$ **then**
    $v \leftarrow$ `getJohansen`$(\mathbf{y}_i, p)$
    $[\mathbf{A} \quad \mathbf{Y}] \leftarrow$ `vecMatrix`$(\mathbf{y}_i, p, v)$
  else
    $[\mathbf{A} \quad \mathbf{Y}] \leftarrow$ `vecMatrixOnline`$(\mathbf{y}_i, p, v, \mathbf{A}, \mathbf{Y})$
    $\Delta\mathbf{Y}_{\text{pred}}[i] \leftarrow \mathbf{A}[-1, :] \times \mathbf{X}$
  end if
  $\mathbf{X} \leftarrow$ `OLS`$(\mathbf{A}, \mathbf{Y})$
  $e \leftarrow$ `mape`$(\mathbf{Y}[-n, :], \mathbf{A}[-n, :] \times \mathbf{X})$
  if `mean`$(e) >$ mean_error **then**
    $v \leftarrow$ `getJohansen`$(\mathbf{y}_i, p)$
    $\mathbf{A} \leftarrow$ `vecMatrixUpdate`$(\mathbf{y}_i, p, v, \mathbf{A})$
    $\mathbf{X} \leftarrow$ `OLS`$(\mathbf{A}, \mathbf{Y})$
  end if
end for
$\mathbf{Y}_{\text{true}} \leftarrow \mathbf{Y}[L+1 : N]$
$\mathbf{Y}_{\text{pred}} \leftarrow \mathbf{Y}[L : N-1] + \Delta\mathbf{Y}_{\text{pred}}$

A pseudocode of the algorithm 10 is detailed in the appendix 5.2.



Our proposal was compared against VECM and ARIMA. Both algorithms were adapted to an online context in order to get a reasonable comparison with our proposal (see algorithms 8 and 9). VECM and ARIMA are called with a sliding window of the most recent data, whereby the models are updated at every time step.

Since we know out time series are I(1) SLARIMA is called with $d = 1$. ARIMA is executed for every time series.

Both OVECM and SLVECM time complexity is dominated by Johansen method which is $O(n^3)$. Thus, both algorithms order is $O(Cn^3)$ where C is the number of iterations.

Algorithm 8 SLVECM: Sliding window VECM

Require:

$\mathbf{y}$: matrix with N input vectors and l time series

p: number of past values

L: sliding window size ($L < N$)

Ensure:

$\{\mathbf{y}_{\text{pred}}[L+1], \ldots, \mathbf{y}_{\text{pred}}[N]\}$: model predictions

for $i = 0$ to $N - L$ **do**
  $\mathbf{y}_i \leftarrow \mathbf{y}[i : i + L + 1]$
  $model = VECM(\mathbf{y}_i, p)$
  $\mathbf{Y}_{\text{pred}}[i] = model.predict(\mathbf{y}[i + L])$
end for
$\mathbf{Y}_{\text{true}} = \mathbf{y}[i + L + 1 : N]$

Algorithm 9 SLARIMA: Sliding window ARIMA

Require:

$\mathbf{y}$: matrix with N input vectors and l time series

p: autoregressive order

d: integrated order

q: moving average order

L: sliding window size ($L < N$)

Ensure:

$\{\mathbf{y}_{\text{pred}}[L+1], \ldots, \mathbf{y}_{\text{pred}}[N]\}$: model predictions

for $i = 0$ to $N - L$ **do**
  for $j = 0$ to $l - 1$ **do**
    $\mathbf{y}_i \leftarrow \mathbf{y}[i : i + L + 1, j]$
    $model = ARIMA(\mathbf{y}_i, (p, d, q))$
    $\mathbf{Y}_{\text{pred}}[i, j] = model.predict(\mathbf{y}[i + L, j])$
  end for
end for
$\mathbf{Y}_{\text{true}} = \mathbf{y}[i + L + 1 : N, :]$

5.3. Results

5.3.1. Data. Tests of SLVECM, SLARIMA and our proposal OVECM were carried out using foreign four exchange data rates: EURUSD, GBPUSD, USDCHF and USDJPY. This data was collected from the free database Dukascopy which gives access to the Swiss Foreign Exchange Marketplace [**Duk14**].

The reciprocal of the last two rates (CHFUSD, JPYUSD) were used in order to obtain the same base currency for all rates. The tests were done using 1-minute frequency

from ask prices which corresponded to 1.440 data points per day from the 11th to the 15th of August 2014.

5.3.2. Unit root tests. Before running the tests, we firstly checked if they were I(1) time series using the Augmented Dickey Fuller (ADF) test.

TABLE 1. Unit roots tests

	Statistic	**Critical value**	**Result**
EURUSD	-0.64	-1.94	True
ΔEURUSD	-70.45	-1.94	False
GBPUSD	-0.63	-1.94	True
ΔGBPUSD	-54.53	-1.94	False
CHFUSD	-0.88	-1.94	True
ΔCHFUSD	-98.98	-1.94	False
JPYUSD	-0.65	-1.94	True
ΔJPYUSD	-85.78	-1.94	False

Table 1 shows that all currency rates cannot reject the unit root test but they rejected it with their first differences. This means that all of them are I(1) time series and we are allowed to use VECM and therefore OVECM.

5.3.3. Parameter selection. In order to set OVECM parameters: windows size L and lag order p, we propose to use several window sizes: $L = 100, 400, 700, 1000$. For every window size L we chose lag order with minimum AIC.

ARIMA parameters were also obtained using AIC. Parameters optimisation is presented in table 2:

TABLE 2. Parameters optimisation. VECM order and ARIMA parameters were selected using AIC.

Windows size L L	VECM order (p)	ARIMA order (p, d, q)
100	2	p=2,d=1,q=1
400	5	p=1,d=1,q=1
700	3	p=2,d=1,q=1
1000	3	p=2,d=1,q=1

In OVECM we also define a mean_error variable, which was defined based on the in-sample MAPEs. Figure 5.1 shows how MAPE moves and how mean_error variable help us to decide whether new cointegration vectors are needed.

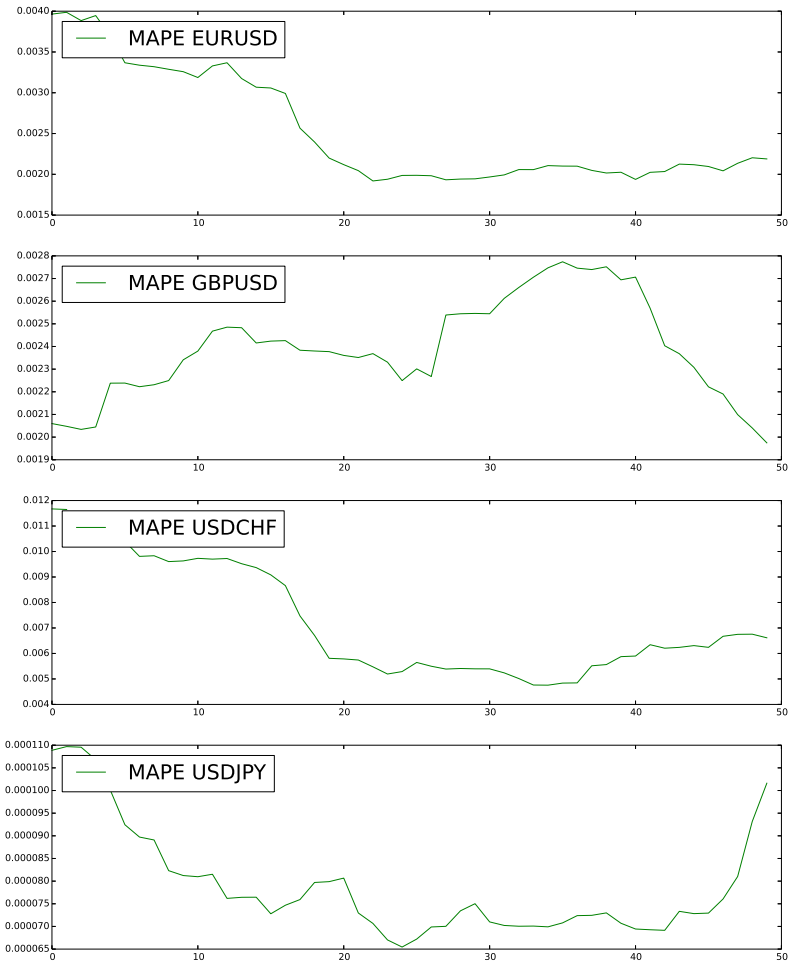


FIGURE 5.1. In-sample MAPEs example for 50 minutes. The average of them is considered to obtain new cointegration vectors.

5.3.4. Execution times. We ran OVECM and SLVECM 400 iterations. SLARIMA execution time is excluded because its is not comparable with OVECM and SLVECM. SLARIMA was created based on statsmodels library routine ARIMA.

The execution times are shown in the table 3.

It is clear that execution time depends directly on L and p since they determine the size of matrix $\mathbf{A}$ and therefore affect the OLS function execution time. It is worthy of note that execution time also depends on mean_error because it determines how many times OVECM will recalculate cointegration vectors which is an expensive routine.

Figure 5.1 shows an example of the in-sample MAPE for 50 iterations. When the average of the in-sample MAPEs is above mean_error new cointegration vectors

TABLE 3. Execution times

	L	order	e	Time[s]
OVECM	100	p=2	0	2.492
OVECM	100	p=2	0.0026	1.606
SLVECM	100	p=2	–	2.100
OVECM	400	p=5	0	3.513
OVECM	400	p=5	0.0041	2.569
SLVECM	400	p=5	–	3.222
OVECM	700	p=3	0	3.296
OVECM	700	p=3	0.0032	2.856
SLVECM	700	p=3	–	3.581
OVECM	1000	p=3	0	4.387
OVECM	1000	p=3	0.0022	2.408
SLVECM	1000	p=3	–	3.609

are obtained. In consequence, OVECM performance increases when mean_error increases. However, this could affect accuracy, but table 4 shows that using an appropriate mean_error doesn't affect accuracy considerable.

5.3.5. Performance accuracy. Table 4 shows in-sample and out-of-sample performance measures: MAPE, MAE and RMSE for OVECM, SLVECM and SLARIMA. Test were done using the parameters defined in table 2. We can see that OVECM has very similar performance than SLVECM and this support the theory that cointegration vectors vary little in time. Moreover, OVECM also out performed SLARIMA based on these three performance measures.

TABLE 4. Model measures

Model				In-sample			Out-of-sample		
Method	L	Parameters	e	MAPE	MAE	RMSE	MAPE	MAE	RMSE
OVECM	100	P=2	0.0026	0.00263	0.00085	0.00114	0.00309	0.00094	0.00131
OVECM	400	P=5	0.0041	0.00378	0.00095	0.00127	0.00419	0.00103	0.00143
OVECM	700	P=3	0.0032	0.00323	0.00099	0.00130	0.00322	0.00097	0.00132
OVECM	1000	P=3	0.0022	**0.00175**	**0.00062**	**0.00087**	**0.00172**	**0.00061**	**0.00090**
SLVECM	100	P=2	-	0.00262	0.00085	0.00113	0.00310	0.00095	0.00132
SLVECM	400	P=5	-	0.00375	0.00095	0.00126	0.00419	0.00103	0.00143
SLVECM	700	P=3	-	0.00324	0.00099	0.00130	0.00322	0.00098	0.00132
SLVECM	1000	P=3	-	**0.00174**	**0.00061**	**0.00087**	**0.00172**	**0.00061**	**0.00090**
SLARIMA	100	p,d,q=2,1,1	-	0.00285	0.00110	0.00308	0.00312	0.00098	0.00144
SLARIMA	400	p,d,q=1,1,1	-	0.00377	0.00101	0.00128	0.00418	0.00106	0.00145
SLARIMA	700	p,d,q=2,1,1	-	0.00329	0.00102	0.00136	0.00324	0.00097	0.00133
SLARIMA	1000	p,d,q=2,1,1	-	**0.00281**	**0.00074**	**0.00105**	**0.00177**	**0.00063**	**0.00092**

We can also notice that in-sample performance in OVECM and SLVECM is related with the out-of-sample performance. This differs with SLARIMA which models with

good in-sample performance are not necessarily good out-of-sample models. Moreover OVECM outperformed SLARIMA using the same window size.

Figure 5.2 shows the out-of-sample forecasts made by our proposal OVECM with the best parameters found based on table 4 which follows the time series very well.

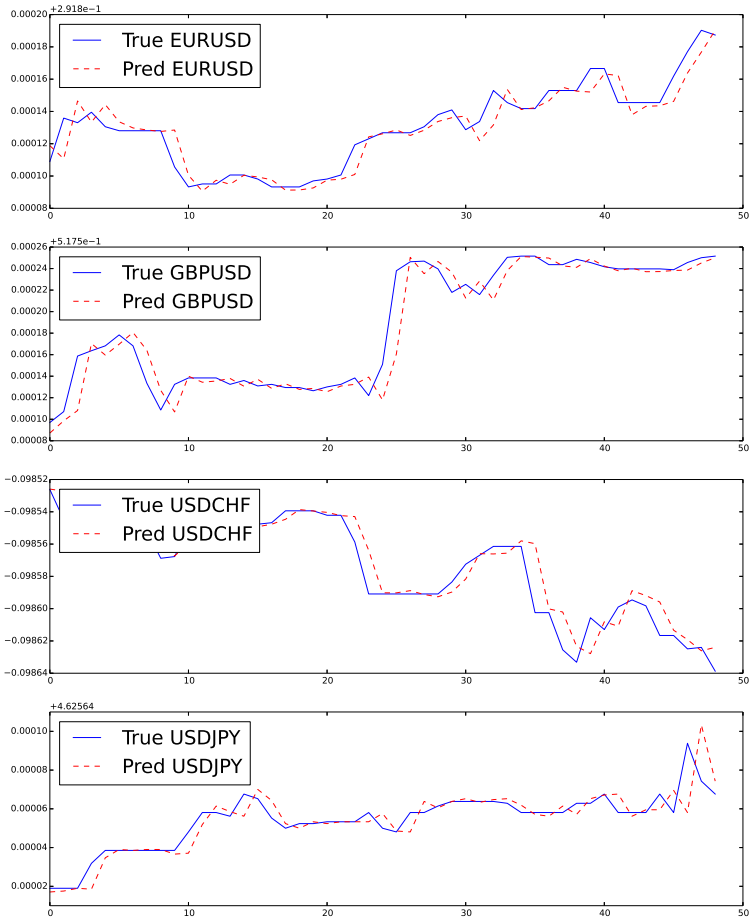


FIGURE 5.2. OVECM forecasting accuracy example for 50 minutes using $L = 1000$ and $p = 3$

5.4. Conclusions

A new online vector error correction method was presented. We have shown that our proposed OVECM considerably reduces execution times without compromising solution accuracy. OVECM was compared with VECM and ARIMA with the same sliding window sizes and OVECM outperformed both in terms of execution time. Traditional VECM slightly outperformed our proposal but the OVECM execution time is lower.

This reduction of execution time is mainly because OVECM avoids the cointegration vector calculation using the Johansen method. The condition for getting new vectors is given by the mean_error variable which controls how many times the Johansen method is called. Additionally, OVECM introduces matrix optimisation in order to get the new model in an iterative way. We could see that our algorithm took much less than a minute at every step. This means that it could also be used with higher frequency data and would still provide responses before new data arrives.

For future study, it would be interesting to improve the out-of-sample forecast by considering more explicative variables, to increase window sizes or trying new conditions to obtain new cointegration vectors.

Since OVECM is an online algorithm which optimises processing time, it could be used by investors as an input for strategy robots. Moreover, some technical analysis methods could be based on its output.

Chapter 6

Intraday Forex rates forecasting using an online cointegration approach

High frequency forex rates Vector error correction model (VECM) parameters are commonly obtained using the ordinary least squares method. Since this parameters could be sensitive to the data, this proposal is to get these parameters using ridge regression which could lead to a better generalization capability. In order to give with a real-time response, an online version of VECM (OVECM) is proposed. OVECM solves VECM considering only a sliding window of historical data and therefore better response times could be achieved.

6.1. The problem

VECM introduces the long-run relationship among a set of cointegrated variables as an error correction term. VECM is a special case of the vector autorregresive model (VAR) model. VAR model expresses future values as a linear combination of variables past values. However, VAR model cannot be used with non-stationary variables, which is a common feature of financial assets. VECM is a linear model but in terms of variable differences. If cointegration exists, variable differences are stationary and they introduce an error correction term which adjusts coefficients to bring the variables back to equilibrium. In finance, many economic time series turn to be stationary when they are differentiated and cointegration restrictions often improves forecasting [**DT98**]. Therefore, VECM has been widely adopted.

Both VECM and VAR model parameters are obtained using ordinary least squares (OLS) method. OLS has two main problems: is sensitive to errors on input data and involves many calculations. The former problem is commonly solved using Ridge Regression (RR) [**HK70**] which introduces a regularization parameter that leads to an unbiased estimation with better generalization capability. The second problem of computational complexity depends on the number of past values and observations considered. Recently, online learning algorithms have been proposed to solve problems with large data sets because of their simplicity and their ability to update the model when new data is available.

Our proposal is an online formulation of the VECM called Online VECM (OVECM) based on consideration of only a sliding window of the historical data. OVECM introduces matrix optimizations in order to reduce the number of operations and also takes into account the fact that cointegration vector space doesn't experience large changes with small changes in the input data. Moreover, OVECM uses RR instead of OLS to obtain VECM parameters. Our method is later tested using four currency rates from the foreign exchange market with a frequency of 10 seconds. Model efectiveness is focused on out-of-sample forecast rather than in-sample fitting. This criteria allows the OVECM prediction capability to be expressed rather than just explaining data history. Our method performance is compared with its optimal offline algorithm.

6.2. Methodology

Since financial problems are stream data problems, it is unfeasible to include all input data in a VECM model. Our proposal consists on an online version of VECM (OVECM) capable of updating the model with new arrival data and give responses in a short period of time. OVECM only considers a time varying window of historical data and the using of RR or its variant AAR (Aggregating Algorithm for Regression) as an alternative of OLS to get model parameters.

On the other hand, getting cointegration vectors using Johansen method is also an expensive procedure. However, since cointegration vectors represent the long-run relationship between the time series, they vary little in time. Our proposal obtains new cointegration vectors only when this long-term relationship changes. This change is detected by tracking the Mean Absolute Percentage Error (MAPE) of the last n in-sample forecasts.

VECM(p) for l cointegrated variables has the following form:

$$\Delta \mathbf{y}_t = \boldsymbol{\alpha}\boldsymbol{\beta}^\top \mathbf{y}_{t-1} + \sum_{i=1}^{p-1} \boldsymbol{\Phi}_i^* \, \Delta \mathbf{y}_{t-i} + \mathbf{c} + \boldsymbol{\epsilon}_t \qquad \forall t = p+1, \ldots, N. \tag{6.1}$$

Transposing each equation of the system (6.1) we can write the VECM(p) model in block-matrix form as:

$$\mathbf{B} = \mathbf{A}\mathbf{X} + \mathbf{E}\,, \tag{6.2}$$

where $\mathbf{B}$ dimension is $((N-p)\times l)$, $\mathbf{A}$ dimension is $((N-p)\times(r+(p-1)l+1))$, $\mathbf{X}$ dimension is $((r+(p-1)l+1)\times l)$, $\mathbf{E}$ dimension is $((N-p)\times l)$ and r is the number of cointegration vectors:

$$\mathbf{B} = \begin{bmatrix} \Delta\mathbf{y}_{p+1}^\top \\ \Delta\mathbf{y}_{p+2}^\top \\ \vdots \\ \Delta\mathbf{y}_{N}^\top \end{bmatrix} \quad \mathbf{X} = \begin{bmatrix} \boldsymbol{\alpha}^\top \\ \boldsymbol{\Phi}_1^{*\top} \\ \boldsymbol{\Phi}_2^{*\top} \\ \vdots \\ \boldsymbol{\Phi}_{p-1}^{*\top} \\ \mathbf{c}^\top \end{bmatrix} \quad \mathbf{E} = \begin{bmatrix} \boldsymbol{\epsilon}_{p+1}^\top \\ \boldsymbol{\epsilon}_{p+2}^\top \\ \vdots \\ \boldsymbol{\epsilon}_{N}^\top \end{bmatrix} \tag{6.3}$$

and

$$\mathbf{A} = \left[\begin{array}{ccccc:c} \mathbf{y}_p^\top\boldsymbol{\beta} & \Delta\mathbf{y}_p^\top & \Delta\mathbf{y}_{p-1}^\top & \cdots & \Delta\mathbf{y}_2^\top & 1 \\ \mathbf{y}_{p+1}^\top\boldsymbol{\beta} & \Delta\mathbf{y}_{p+1}^\top & \Delta\mathbf{y}_p^\top & \cdots & \Delta\mathbf{y}_3^\top & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{y}_{N-1}^\top\boldsymbol{\beta} & \Delta\mathbf{y}_{N-1}^\top & \Delta\mathbf{y}_{N-2}^\top & \cdots & \Delta\mathbf{y}_{N-p-1}^\top & 1 \end{array}\right]. \tag{6.4}$$

However, the number of rows of matrix $\mathbf{A}$ increases with the number of observations $\mathbf{y}_t$. Our proposal considers only the most recent L rows of matrices $\mathbf{A}$ and $\mathbf{B}$ defined as $\mathbf{A}(t)$ and $\mathbf{B}(t)$:

$$\mathbf{A}(t) = \begin{bmatrix} — & \mathbf{a}_{t-L}^\top & — \\ — & \mathbf{a}_{t-L+1}^\top & — \\ & \vdots & \\ — & \mathbf{a}_{t}^\top & — \end{bmatrix} \quad \text{and} \quad \mathbf{B}(t) = \begin{bmatrix} — & \mathbf{b}_{t-L} & — \\ — & \mathbf{b}_{t-L+1} & — \\ & \vdots & \\ — & \mathbf{b}_{t} & — \end{bmatrix}, \tag{6.5}$$

so that VECM parameters $\mathbf{X}(t)$ are found using the following equation:

$$\mathbf{B}(t) = \mathbf{A}(t)\mathbf{X}(t) + \mathbf{E}(t) \tag{6.6}$$

The RR solution $\hat{\mathbf{X}}(t)$ using the sliding window matrices defined above is:

$$\hat{\mathbf{X}}(t) = \mathbf{S}(t)^{-1}\mathbf{W}(t)\,, \tag{6.7}$$

where $\mathbf{S}t$ and $\mathbf{W}t$ are define as:

$$\begin{aligned}\mathbf{S}(t) = \quad \mathbf{A}(t)^\top \mathbf{A}(t) + \lambda \mathbb{I} \quad &= \sum_{i=0}^{L} \mathbf{a}_{t-i} \mathbf{a}_{t-i}^\top + \lambda \mathbb{I} \\ \mathbf{W}(t) = \qquad \mathbf{A}(t)^\top \mathbf{B}(t) \qquad &= \sum_{i=0}^{L} \mathbf{a}_{t-i} \mathbf{b}_{t-i}\end{aligned}$$

It is worth noticing that, at the next time step, the matrices $\mathbf{S}(t+1)$ and $\mathbf{W}(t+1)$ are slightly different to $\mathbf{S}(t)$ and $\mathbf{W}(t)$:

$$\begin{aligned}\mathbf{S}(t+1) &= \mathbf{S}(t) + \mathbf{a}_{t+1}\mathbf{a}_{t+1}^\top - \mathbf{a}_{t-L}\mathbf{a}_{t-L}^\top \\ \mathbf{W}(t+1) &= \mathbf{W}(t) + \mathbf{a}_{t+1}\mathbf{b}_{t+1} - \mathbf{a}_{t-L}\mathbf{b}_{t-L}\,.\end{aligned}$$

The algorithm 10 shows OVECM using three different methods for geting model parameters: OLS, RR, AAR which leads to three different algorithms OVECM-OLS, OVECM-RR, OVECM-AAR:

Our proposal considers the following:

- The function `getJohansen` returns cointegration vectors given by the Johansen method considering the trace statistic test at 95% level of significance. This procedure is called at the first step of the algorithm and every time in-sample MAPE ($\mathbf{e}$) exceeds threshold error defined (mean_error).
- The function `vecMatrix` returns VECM matrices $\mathbf{A}(t), \mathbf{B}(t)$ shown in equation (6.5). This method is only required at the first step.
- The function `vecMatrixOnline` returns new rows $\mathbf{a}_t^\top$ and $\mathbf{b}_t^\top$ given new input data $\mathbf{y}_t$.
- The Solver class (see algorithm 11) obtain VECM parameters using RR, AAR or OLS methods.
- The function `vecMatrixUpdate` updates matrix $\mathbf{A}$ when new cointegration vectors are required (matrix $\mathbf{B}$ is not affected). Model parameters $\mathbf{X}$ is also updated later.

The proposal was compared against an optimal online algorithm which is a time varying VECM (OOVECM). Algorithm 12 shows this modification:

6.3. Experiments

Algorithm 10 OVECM: Online VECM

Require:

$\mathbf{y}$: matrix with N input vectors and l time series
p: number of past values
method: {'OLS','RR','AAR'}
λ: regularization parameter (only requires for RR and AAR methods)
L: window size of data($L < N$)
mean_error: MAPE threshold
n: windows size to obtain in-sample MAPE

Ensure:

$\{\mathbf{y}_{\text{pred}}[L+1], \ldots, \mathbf{y}_{\text{pred}}[N]\}$: model predictions

solver = new `Solver`(method,λ)
for $t = 1$ to $N - L$ **do**
  $\mathbf{Y} \leftarrow \mathbf{y}[t : t + L - 1]$
  $\mathbf{y}_t \leftarrow \mathbf{y}[t + L]$
  if $t = 1$ **then**
    $v \leftarrow \texttt{getJohansen}(\mathbf{Y}, p)$
    $[\mathbf{A} \quad \mathbf{B}] \leftarrow \texttt{vecMatrix}(\mathbf{Y}, p, v)$
    `solver.init_model`$(\mathbf{A}, \mathbf{B})$
  end if
  $[\mathbf{A} \quad \mathbf{B} \quad \mathbf{a}_t \quad \mathbf{a}_L \quad \mathbf{b}_t \quad \mathbf{b}_L] \leftarrow \texttt{vecMatrixOnline}(\mathbf{Y}, \mathbf{y}_t, p, v)$
  $[\mathbf{X} \quad \mathbf{y}_{\text{pred}}[t]] \leftarrow \texttt{solver.regression}(\mathbf{A}, \mathbf{B}, \mathbf{a}_t, \mathbf{b}_t)$
  $\mathbf{Y}_{\text{pred}} = \mathbf{A}\mathbf{X}[-n :] + \mathbf{Y}[-n - 1 : -1]$
  $\mathbf{e} = \texttt{mape}(\mathbf{Y}_{\text{pred}}, \mathbf{Y}[-n :])$
  if $\texttt{mean}(\mathbf{e}) >$ mean_error **then**
    $v \leftarrow \texttt{getJohansen}(\mathbf{Y}, p)$
    $\mathbf{A} \leftarrow \texttt{vecMatrixUpdate}(\mathbf{A}, \mathbf{Y}, p, v)$
    `solver.init_model`$(\mathbf{A}, \mathbf{B})$
    $[\mathbf{a}_t \quad \mathbf{b}_t] \leftarrow \texttt{vecMatrixOnline}(\mathbf{Y}, \mathbf{y}_t, p, v)$
    $[\mathbf{X} \quad \mathbf{y}_{\text{pred}}[t]] \leftarrow \texttt{solver.regression}(\mathbf{A}, \mathbf{B}, \mathbf{a}_t, \mathbf{b}_t)$
  end if
  $[\mathbf{A} \quad \mathbf{B}] \leftarrow \texttt{matrixUpdate}(\mathbf{A}, \mathbf{B}, \mathbf{a}_t, \mathbf{b}t)$
end for

Algorithm 11 Solver Class for Regression Methods

Require:

method: {'OLS','RR','AAR'}

$\mathbf{A}$: VECM design matrix

$\mathbf{B}$: VECM dependant variables

$\mathbf{a}_t$: new row of $\mathbf{A}$

$\mathbf{b}_t$: new row of $\mathbf{B}$

γ: regularization parameter

n: windows size to obtain in-sample MAPE

Ensure:

$\mathbf{X}$: regression solution

$\mathbf{e}$: in-sample MAPE

`init_model`$(\mathbf{A}, \mathbf{B})$

$[m \quad n] = \text{size}(\mathbf{A})$

$\mathbf{S} = \sum_{i=1}^{m} \mathbf{a}_i \mathbf{a}_i^\top + \gamma \mathbb{I}$

$\mathbf{W} = \sum_{i=1}^{m} \mathbf{a}_i \mathbf{b}_i$

`regression`$(\mathbf{A}, \mathbf{B}, \mathbf{a_t}, \mathbf{b_t})$

if method == 'OLS' **then**

  $\mathbf{X} = (\mathbf{A}^\top \mathbf{A})^{-1} \mathbf{A}^\top \mathbf{B}$

else if method == 'RR' **then**

  $\mathbf{X} = \mathbf{S}^{-1} \mathbf{W}$

  $\mathbf{S} = \mathbf{S} + \mathbf{a}_t \mathbf{a}_t^\top - \mathbf{a}_1 \mathbf{a}_1^\top$

  $\mathbf{W} = \mathbf{W} + \mathbf{a}_t \mathbf{b}_t$

else if method == 'ARR' **then**

  $\mathbf{S} = \mathbf{S} + \mathbf{a}_t \mathbf{a}_t^\top - \mathbf{a}_1 \mathbf{a}_1^\top$

  $\mathbf{X} = \mathbf{S}^{-1} \mathbf{W}$

  $\mathbf{W} = \mathbf{W} + \mathbf{a}_t \mathbf{b}_t$

end if

$\mathbf{y}_{\text{pred}} = \mathbf{X}^\top \mathbf{a}_t$

Algorithm 12 OOVECM: Optimal Online Vector Error Correction Model

Require:

$\mathbf{y}$: matrix with N input vectors and l time series

p: number of past values

L: sliding window size ($L < N$)

Ensure:

$\{\Delta\mathbf{y}_{\text{pred}}[L+1], \ldots, \Delta\mathbf{y}_{\text{pred}}[N]\}$: model predictions

1: **for** $i = 0$ to $N - L$ **do**
2: $\quad\mathbf{y}_i \leftarrow \mathbf{y}[i : i + L]$
3: $\quad v \leftarrow \texttt{getJohansen}(\mathbf{y}_i, p)$
4: $\quad[\mathbf{A} \quad \mathbf{B}] \leftarrow \texttt{vecMatrix}(\mathbf{y}_i, p, v)$
5: $\quad\mathbf{X} \leftarrow \text{OLS}(\mathbf{A}, \mathbf{B})$
6: $\quad\Delta\mathbf{Y}_{\text{true}}[i] \leftarrow \mathbf{B}[-1, :]$
7: $\quad\Delta\mathbf{Y}_{\text{pred}}[i] \leftarrow \mathbf{A}[-1, :] \times \mathbf{X}$
8: **end for**
9: MAPE $\leftarrow \texttt{mape}(\Delta\mathbf{Y}_{\text{true}}, \Delta\mathbf{Y}_{\text{pred}})$

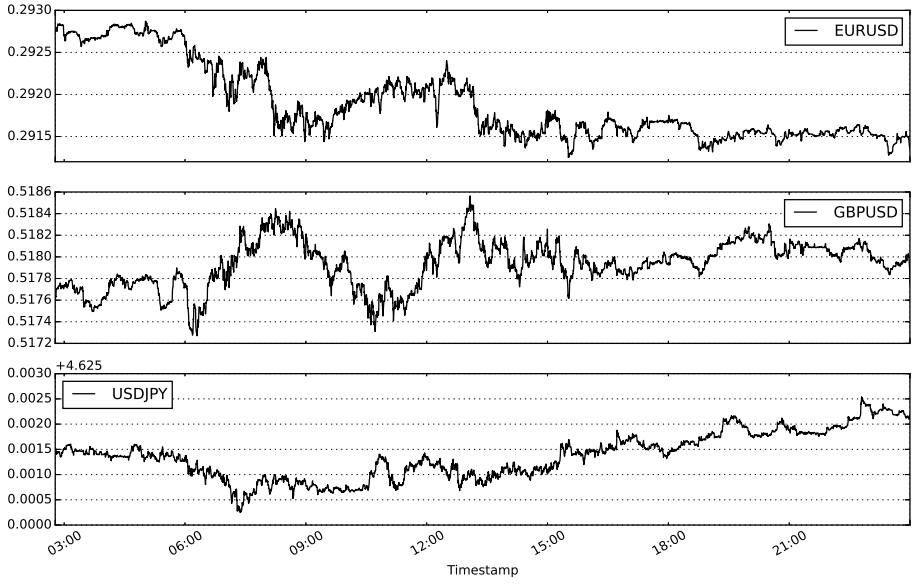


FIGURE 6.1. 10 seconds frequency forex data for EURUSD, GBPUSD and USDJPY

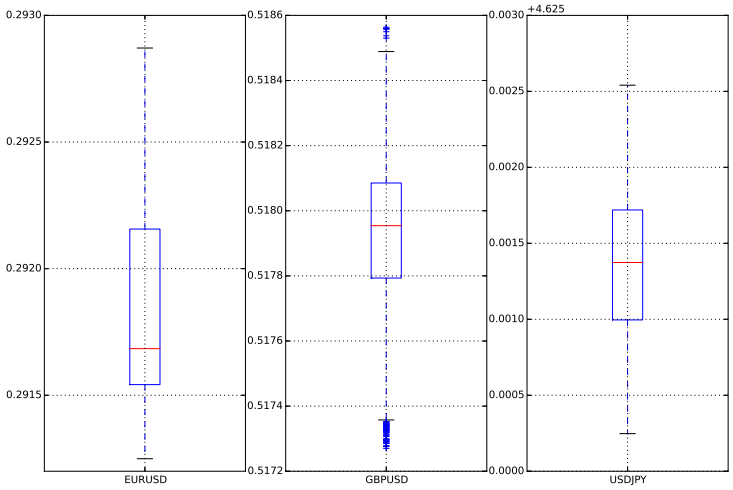


FIGURE 6.2. Distribution of EURUSD, GBPUSD and USDJPY data

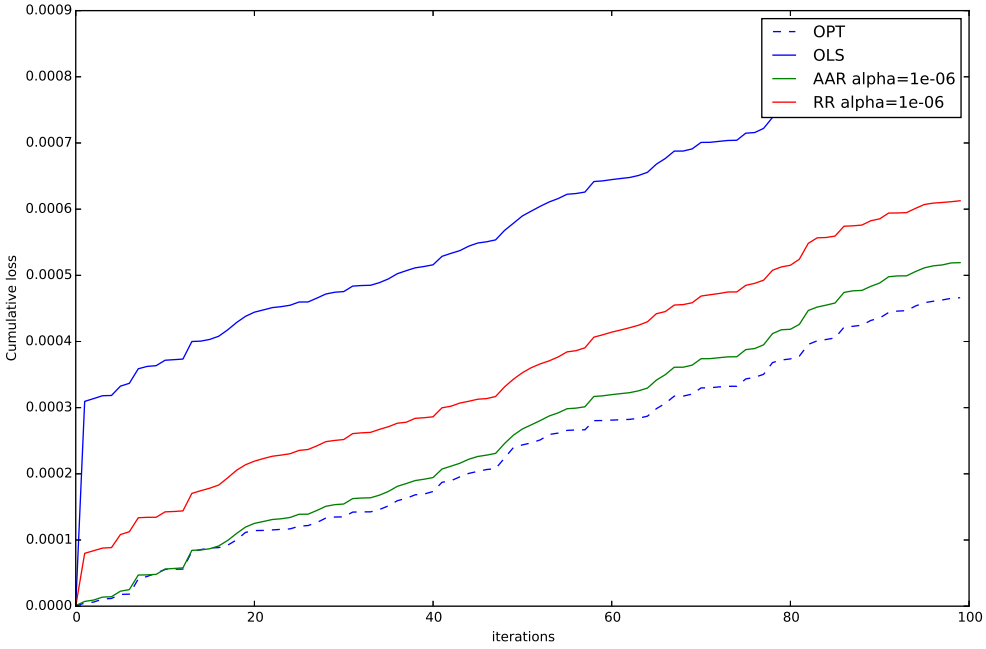


FIGURE 6.3. Cumulative loss of AAR, RR and OLS solutions for VECM against its optimal algorithm

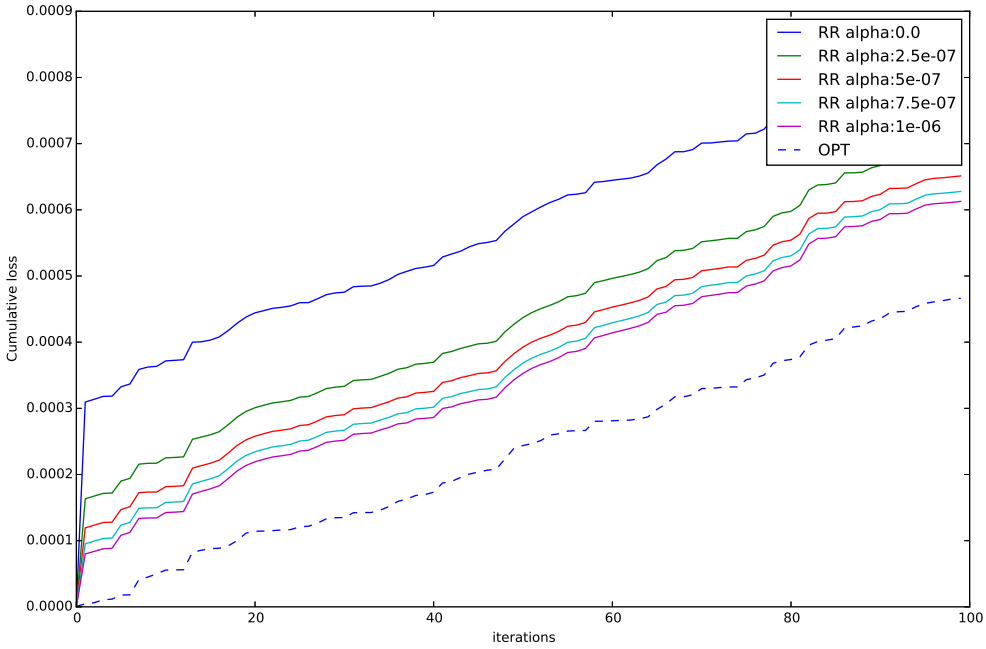


FIGURE 6.4. Cumulative loss RR VECM against its optimal algorithm using different λ values

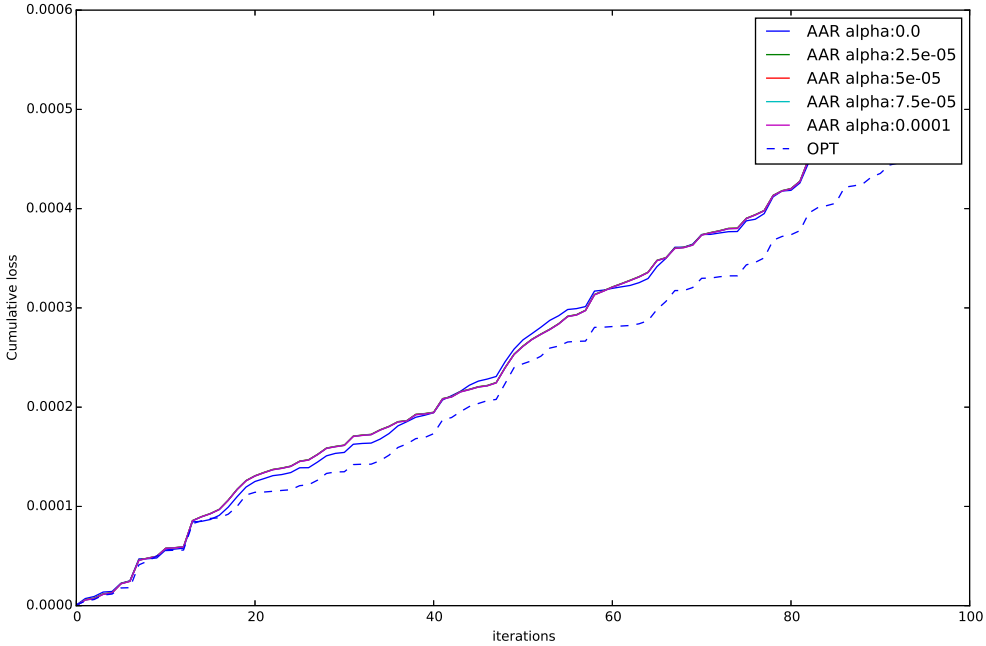


FIGURE 6.5. Cumulative loss AAR VECM against its optimal algorithm using different λ values

Chapter 7

Conclusion

7.1. Conclusions of this thesis

7.2. Contributions of this thesis

7.3. Future Work

Bibliography

[AB98] Torben G Andersen and Tim Bollerslev, *Answering the skeptics: Yes, standard volatility models do provide accurate forecasts*, International Economic Review **39** (1998), no. 4, 885–905.

[ABDL03] Torben G. Andersen, Tim Bollerslev, Francis X. Diebold, and Paul Labys, *Modeling and forecasting realized volatility*, Econometrica **71** (2003), no. 2, 579–625.

[ADL01] Philip Arestis, Panicos O Demetriades, and Kul B Luintel, *Financial development and economic growth: the role of stock markets*, Journal of Money, Credit and Banking (2001), 16–41.

[Ald09] I. Aldridge, *High-frequency trading: A practical guide to algorithmic strategies and trading systems*, Wiley Trading, Wiley, 2009.

[AS12] Paola Arce and Luis Salinas, *Online ridge regression method using sliding windows*, XXXI International Conference of the Chilean Computer Science Society (2012).

[AW01] Katy S Azoury and Manfred K Warmuth, *Relative loss bounds for on-line density estimation with the exponential family of distributions*, Machine Learning **43** (2001), no. 3, 211–246.

[B+98] Leo Breiman et al., *Arcing classifier (with discussion and a rejoinder by the author)*, The annals of statistics **26** (1998), no. 3, 801–849.

[B+06] Christopher M Bishop et al., *Pattern recognition and machine learning*, vol. 4, springer New York, 2006.

[Ban93] A. Banerjee, *Co-integration, error correction, and the econometric analysis of non-stationary data*, Advanced texts in econometrics, Oxford University Press, 1993.

[BAW87] Giovanni Barone-Adesi and Robert E Whaley, *Efficient analytic approximation of american option values*, Journal of Finance **42** (1987), no. 2, 301–20.

[BDvLSTT05] Shai Ben-David, Ulrike von Luxburg, John Shawe-Taylor, and Naftali Tishby, *Theoretical foundations of clustering*, NIPS Workshop, 2005.

[BGV92] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik, *A training algorithm for optimal margin classifiers*, Proceedings of the fifth annual workshop on Computational learning theory, ACM, 1992, pp. 144–152.

[Bol86] Tim Bollerslev, *Generalized autoregressive conditional heteroskedasticity*, Journal of Econometrics **31** (1986), no. 3, 307–327.

[Bro02] C. Brooks, *Introductory econometrics for finance*, Cambridge University Press, 2002.

[BS73] Fischer Black and Myron S Scholes, *The pricing of options and corporate liabilities*, Journal of Political Economy **81** (1973), no. 3, 637–54.

[CBCG05] Nicolò Cesa-Bianchi, Alex Conconi, and Claudio Gentile, *A second-order perceptron algorithm*, SIAM J. Comput. **34** (2005), no. 3, 640–668.

[CBL06] Nicolo Cesa-Bianchi and Gabor Lugosi, *Prediction, learning, and games*, Cambridge University Press, New York, NY, USA, 2006.

[CCBG07] Giovanni Cavallanti, Nicolò Cesa-Bianchi, and Claudio Gentile, *Tracking the best hyperplane with a simple budget perceptron*, Machine Learning **69** (2007), no. 2-3, 143–167.

[CDK+06] K Crammer, O Dekel, J Keshet, S Shalev-Shwartz, and Y Singer, *Online passive-aggressive algorithms*, Journal of Machine Learning Research **7** (2006), 551–585.

[CHJ10] Shiyi Chen, Wolfgang Karl Hardle, and Kiho Jeong, *Forecasting volatility with support vector machine-based garch model*, Journal of Forecasting **29** (2010), no. 4, 406–433.

[CJH08] Shiyi Chen, Kiho Jeong, and Wolfgang Karl Hardle, *Support vector regression based garch model with application to forecasting volatility of financial returns*, SFB 649 Discussion Papers SFB649DP2008-014, Humboldt University, Collaborative Research Center 649, 2008.

[CKHS03] Koby Crammer, Jaz Kandola, Royal Holloway, and Yoram Singer, *Online classification on a budget*, Advances in Neural Information Processing Systems 16, MIT Press, 2003, p. 2003.

[CKP10] Ying Chen, Wolfgang Karl, Hardle, and Uta Pigorsch, *Localized realized volatility modeling*, Journal of the American Statistical Association **105** (2010), no. 492, 1376–1393.

[Col90] Mark Coleman, *Cointegration-based tests of daily foreign exchange market efficiency*, Economics Letters **32** (1990), no. 1, 53–59.

[Con01] Rama Cont, *Empirical properties of asset returns: stylized facts and statistical issues.*

[Cop91] Laurence S Copeland, *Cointegration tests with daily exchange rate data*, Oxford Bulletin of Economics and Statistics **53** (1991), no. 2, 185–198.

[CS10] Thomas F. Coleman and Chunguang Sun, *Solving rank-deficient linear-squares problems.*

[CSKM11] Michael Chlistalla, Bernhard Speyer, Sabine Kaiser, and Thomas Mayer, *High-frequency trading*, Deutsche Bank Research (2011), 1–19.

[CV95] Corinna Cortes and Vladimir Vapnik, *Support-vector networks*, Machine learning **20** (1995), no. 3, 273–297.

[DF79] David A Dickey and Wayne A Fuller, *Distribution of the estimators for autoregressive time series with a unit root*, Journal of the American statistical association **74** (1979), no. 366a, 427–431.

[Die03] TG Dietterich, *Machine learning in nature encyclopedia of cognitive science*, 2003.

[DJW92] Gerald P Dwyer Jr and Myles S Wallace, *Cointegration and market efficiency*, Journal of International Money and Finance **11** (1992), no. 4, 318–327.

[DK97] R. Glen Donaldson and Mark Jack Kamstra, *An artificial neural network-garch model for international stock return volatility*, Journal of Empirical Finance **4** (1997), no. 1, 17–46.

[DSSS08] Ofer Dekel, Shai Shalev-Shwartz, and Yoram Singer, *The forgetron: a kernel-based perceptron on a budget*, 2008.

[DT98] Timothy A Duy and Mark A Thoma, *Modeling and forecasting cointegrated variables: some practical experience*, Journal of Economics and Business **50** (1998), no. 3, 291–307.

[Duk14] Dukascopy, *Dukascopy historical data feed*, 2014.

[DVV07] Robin De Vilder and Marcel P. Visser, *Proxies for daily volatility*, Pse working papers, HAL, 2007.

[EG87a] Robert F. Engle and C. W. J. Granger, *Co-Integration and Error Correction: Representation, Estimation, and Testing*, Econometrica **55** (1987), no. 2, 251–276.

[EG87b] Robert F Engle and Clive WJ Granger, *Co-integration and error correction: representation, estimation, and testing*, Econometrica: journal of the Econometric Society (1987), 251–276.

[Eng82] Robert F. Engle, *Autoregressive conditional heteroscedasticity with estimates of the variance of united kingdom inflation*, Econometrica **50** (1982), no. 4, 987–1007.

[Eng93] ———, *Statistical models for financial volatility*, Financial Analysts Journal **49** (1993), no. 1, pp. 72–78 (English).

[EP04] Robert F Engle and Andrew J Patton, *Impacts of trades in an error-correction model of quote prices*, Journal of Financial Markets **7** (2004), no. 1, 1–25.

[Fam70] Eugene F Fama, *Efficient capital markets: A review of theory and empirical work**, The journal of Finance **25** (1970), no. 2, 383–417.

[FF08] Eugene F. Fama and Kenneth R. French, *Dissecting anomalies*, Journal of Finance **63** (2008), no. 4, 1653–1678.

[GB06] Valeriy Gavrishchaka and Supriya Banerjee, *Support vector machine as an efficient framework for stock market volatility forecasting*, Computational Management Science **3** (2006), no. 2, 147–160.

[Gei75] Seymour Geisser, *The predictive sample reuse method with applications*, Journal of the American Statistical Association **70** (1975), no. 350, 320–328.

[GN74] Clive WJ Granger and Paul Newbold, *Spurious regressions in econometrics*, Journal of econometrics **2** (1974), no. 2, 111–120.

[GNW96] Allan W Gregory, James M Nason, and David G Watt, *Testing for structural breaks in cointegrated relationships*, Journal of Econometrics **71** (1996), no. 1, 321–341.

[Gol65] G. Golub, *Numerical methods for solving linear least squares problems*, Numerische Mathematik **7** (1965), no. 3, 206–216 (English).

[Gra86] Clive W Granger, *Developments in the study of cointegrated economic variables*, Oxford Bulletin of economics and statistics **48** (1986), no. 3, 213–228.

[GVL80] Gene H Golub and Charles F Van Loan, *An analysis of the total least squares problem*, SIAM Journal on Numerical Analysis **17** (1980), no. 6, 883–893.

[Hay98] Simon Haykin, *Neural networks: A comprehensive foundation*, 2nd ed., Prentice Hall PTR, Upper Saddle River, NJ, USA, 1998.

[Her03] Daniel Herlemont, *Pairs trading, convergence trading, cointegration*, YATS Finances & Techonologies **5** (2003).

[Hes93] Steven L. Heston, *A closed-form solution for options with stochastic volatility with applications to bond and currency options*, Review of Financial Studies **6** (1993), 327–343.

[HI04] Shaikh A. Hamid and Zahid Iqbal, *Using neural networks for forecasting volatility of s&p 500 index futures prices*, Journal of Business Research **57** (2004), no. 10, 1116–1125.

[HK70] Arthur E Hoerl and Robert W Kennard, *Ridge regression: Biased estimation for nonorthogonal problems*, Technometrics **12** (1970), no. 1, 55–67.

[HL06] Peter Reinhard Hansen and Asger Lunde, *Consistent ranking of volatility models*, Journal of Econometrics **131** (2006), no. 1-2, 97–121.

[Joh88] Soren Johansen, *Statistical analysis of cointegration vectors*, Journal of Economic Dynamics and Control **12** (1988), no. 2-3, 231–254.

[Joh95] ———, *Likelihood-based inference in cointegrated vector autoregressive models*, Oxford University Press, 1995.

[KJH05] Siem Jan Koopman, Borus Jungbacker, and Eugenie Hol, *Forecasting daily variability of the s&p 100 stock index using historical, realised and implied volatility measurements*, Journal of Empirical Finance **12** (2005), no. 3, 445–475.

[LF05] Sergio Lence and Barry Falk, *Cointegration, market integration, and market efficiency*, Journal of International Money and Finance **24** (2005), no. 6, 873–890.

[LH05a] Asger Lunde and Peter R. Hansen, *A forecast comparison of volatility models: does anything beat a garch(1,1)?*, Journal of Applied Econometrics **20** (2005), no. 7, 873–889.

[LH05b] ———, *A forecast comparison of volatility models: does anything beat a garch(1,1)?*, Journal of Applied Econometrics **20** (2005), no. 7, 873–889.

[LL91] Kon S Lai and Michael Lai, *A cointegration test for market efficiency*, Journal of Futures Markets **11** (1991), no. 5, 567–575.

[LM11] Andrew W Lo and A Craig MacKinlay, *A non-random walk down wall street*, Princeton University Press, 2011.

[MK00] Ramin Cooper Maysami and Tiong Sim Koh, *A vector error correction model of the singapore stock market*, International Review of Economics & Finance **9** (2000), no. 1, 79–96.

[mLC13] Kim man Lui and Terence TL Chong, *Do technical analysts outperform novice traders: Experimental evidence*, Economics Bulletin **33** (2013), no. 4, 3080–3087.

[MN95] Tarun K Mukherjee and Atsuyuki Naka, *Dynamic relations between macroeconomic variables and the japanese stock market: an application of a vector error correction model*, Journal of Financial Research **18** (1995), no. 2, 223–37.

[Nel] Daniel B. Nelson, *Stationarity and persistence in the garch(1,1) model*, Econometric Theory **6**, 318–334.

[NP82] Charles R Nelson and Charles R Plosser, *Trends and random walks in macroeconmic time series: some evidence and implications*, Journal of monetary economics **10** (1982), no. 2, 139–162.

[PG03] Ser-Huang Poon and Clive W. J. Granger, *Forecasting volatility in financial markets: A review*, Journal of Economic Literature **41** (2003), no. 2, 478–539.

[PPG12] Vasilios Plakandaras, Theophilos Papadimitriou, and Periklis Gogas, *Directional forecasting in financial time series using support vector machines: The usd/euro exchange rate*, DUTH Research Papers in Economics 5-2012, Democritus University of Thrace, Department of International Economic Relations and Development, 2012.

[Ric95] Anthony J Richards, *Comovements in national stock market returns: Evidence of predictability, but not cointegration*, Journal of monetary Economics **36** (1995), no. 3, 631–654.

[Ros58] F. Rosenblatt, *The Perceptron: A probabilistic model for information storage and organization in the brain*, Psychological Review **65** (1958), 386–408.

[SAZ13] Byeongchan Seong, Sung K Ahn, and Peter A Zadrozny, *Estimation of vector error correction models with mixed-frequency data*, Journal of Time Series Analysis **34** (2013), no. 2, 194–205.

[SB98] Richard S Sutton and Andrew G Barto, *Introduction to reinforcement learning*, MIT Press, 1998.

[SD84] Said E Said and David A Dickey, *Testing for unit roots in autoregressive-moving average models of unknown order*, Biometrika **71** (1984), no. 3, 599–607.

[Sew11] Martin Sewell, *Characterization of financial time series*, RN **11** (2011), no. 01, 01.

[She95] N. Shephard, *Statistical aspects of arch and stochastic volatility*, Economics discussion paper, Nuffield College, 1995.

[Sim80] Christopher A Sims, *Macroeconomics and reality*, Econometrica: Journal of the Econometric Society (1980), 1–48.

[SP10] Skipper Seabold and Josef Perktold, *Statsmodels: Econometric and statistical modeling with python*, Proceedings of the 9th Python in Science Conference, 2010, pp. 57–61.

[ST85] Daniel D Sleator and Robert E Tarjan, *Amortized efficiency of list update and paging rules*, Communications of the ACM **28** (1985), no. 2, 202–208.

[Sto94] James H Stock, *Unit roots, structural breaks and trends*, Handbook of econometrics **4** (1994), 2739–2841.

[SW88] James H Stock and Mark W Watson, *Testing for common trends*, Journal of the American statistical Association **83** (1988), no. 404, 1097–1107.

[SW11] C. Sammut and G.I. Webb, *Encyclopedia of machine learning*, Springer reference, Springer, 2011.

[Tsa05] R.S. Tsay, *Analysis of financial time series*, Wiley Series in Probability and Statistics, Wiley, 2005.

[Tsy04] Alexey Tsymbal, *The problem of concept drift: definitions and related work*, Computer Science Department, Trinity College Dublin (2004).

[Vap98] Vladimir N. Vapnik, *Statistical learning theory*, Wiley-Interscience, 1998.

[VGS05] Vladimir Vovk, Alex Gammerman, and Glenn Shafer, *Algorithmic learning in a random world*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[Vov01] Volodya Vovk, *Competitive on-line statistics*, International Statistical Review **69** (2001), 2001.

[VVVS06] Steven Van Vaerenbergh, Javier Vía, and Ignacio Santamaría, *A sliding-window kernel RLS algorithm and its application to nonlinear channel identification*, IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2006) (Toulouse, France), May 2006.

[Wei12] Yu Wei, *Forecasting volatility of fuel oil futures in China: GARCH-type, SV or realized volatility models?*, Physica a-statistical mechanics and its applications **391** (2012), no. 22, 5546–5556.

[WK96] Gerhard Widmer and Miroslav Kubat, *Learning in the presence of concept drift and hidden contexts*, Machine learning **23** (1996), no. 1, 69–101.

[Yul26] G Udny Yule, *Why do we sometimes get nonsense-correlations between time-series?–a study in sampling and the nature of time-series*, Journal of the royal statistical society (1926), 1–63.

[Zha04] Tong Zhang, *Solving large scale linear prediction problems using stochastic gradient descent algorithms*, ICML 2004: Proceedings of the twenty-first international conference on Machine Learning. OMNIPRESS, 2004, pp. 919–926.