Risk Management using Model Predictive Control

Farzad Noorian

A thesis submitted in fulfilment of the requirements
for the degree of Doctor of Philosophy

Faculty of Engineering & Information Technologies

University of Sydney

2016

Abstract

Forward planning and risk management are crucial for the success of any system or business dealing with the uncertainties of the real world. Previous approaches have largely assumed that the future will be similar to the past, or used simple forecasting techniques based on ad-hoc models. Improving solutions requires better projection of future events, and necessitates robust forward planning techniques that consider forecasting inaccuracies.

This work advocates risk management through optimal control theory, and proposes several techniques to combine it with time-series forecasting. Focusing on applications in foreign exchange (FX) and battery energy storage systems (BESS), the contributions of this thesis are three-fold. First, a short-term risk management system for FX dealers is described, in which the optimal risk-cost profiles are obtained through dynamic control of the dealers' positions on the spot market. This approach is formulated as a stochastic model predictive control (SMPC) problem, incorporating elements which model client flow, transaction cost, market impact, exchange rate volatility, and fluctuations caused by macroeconomic announcements. Second, an evolutionary approach to non-linear time-series forecasting is proposed. Using context-free grammars (CFG), a template for combining different steps of feature extraction, model selection, and hyper-parameter optimisation into a single forecasting program is described. Subsequently, using grammatical evolution (GE), an evolutionary search is undertaken in the grammar space to find the optimal forecaster for a specific time-series. Third, a novel measure for evaluating forecasting models, as a part of the predictive model in finite horizon optimal control applications, is proposed. Specifically, for the case of linear-quadratic time-invariant systems, a closed-form equation for the increase in cost of a discrete-time finite horizon controller (FHC) due to prediction error is derived, and it is demonstrated that this measure is a better choice for model validation compared to standard error measures.

Using both synthetic and historical data, the proposed techniques were validated and benchmarked. Real-world data from Westpac Institutional Bank retail FX trades were

used to test the proposed FX risk management and forecasting systems. It is shown that while the best model for describing FX trades is a Gaussian distribution, the stochastic risk management system exhibits better risk management on a risk-cost Pareto frontier compared to rule-based hedging strategies, with up to 44.7% lower cost for the same level of risk.

Similarly, for a real-world BESS application, data from TransGrid iDemand management system was used. The results indicated that the GE optimised forecasting models outperformed other prediction models by at least 9%, improving the overall peak shaving capacity of the system to 57.6%.

Statement of Originality

I certify that the intellectual content of this thesis is the product of my own work and that all the assistance received in preparing this thesis and sources have been acknowledged, specifically:

- The idea of FX risk management solely by controlling positions, and a cloud-based system for automatic forecasting, were originally proposed by Dr. Barry Flower (Westpac Institutional Bank) and Professor Philip H.W. Leong.
- The research direction has been provided by Professor Leong.
- Formalising FX risk management by using an MPC framework was suggested by Dr. Syed Ahmed Pasha.
- Ideas of using context-free grammars for time-series prediction was originally conceived in a discussion with Professor Leong, Dr. Richard I.A. Davis, and Mr. Anthony Mihirana de Silva.
- Parts of appendices are copied with permission from a paper co-authored with Mr. de Silva.

Acknowledgements

First and foremost, I have to express my sincere thanks to Professor Philip Leong for his continuous support during this project. His unique ideas, encouragement, and idealism were the main drivers behind this thesis, and I am deeply indebted to his inspirational enthusiasm, patience, and insightful advice.

I am extremely grateful to Dr. Barry Flower, Head of FX & Commodities Technology at Westpac Institutional Bank. His resourcefulness and management skills, as well as his insight and expertise have greatly assisted this thesis. Without his support, this research would have never started.

Many people helped me along the way and I want to thank them for all their help, support, interest, and valuable hints. Especially, I am obliged to thank the Computer Engineering Lab's financial engineering researchers, Dr. Syed Ahmed Pasha, Dr. Mahsa Baktashmotlagh, and Dr. Jia-Chen Hua, as well as Dr. Richard Davis, Mr. Anthony Mihirana de Silva, Mr. Lei Li, and Mr. Duncan Moss, with whom I had the opportunity to co-author papers. I have received a similarly warm reception from our collaborators at Westpac, especially from Dr. Giulio Katis, Dr. Mark Lauer, Dr. Darren Vorster, Mr. Jeremy Roberts, and Mr. Steve Findlay.

I would also like to thank my co-supervisor, Prof. Alistair McEwan, and CARLab researchers, Prof. Craig Jin and Dr. Nicolas Epain, for their advice. I am thankful to other CEL lab researchers, Dr. JunKyu Lee, Dr. Michael Frechtling, Mr. Nicholas Fraser, Mr. Zhe Zhang, and Mr. Stephen Tridgell for their feedback on my work and their help in proofreading my papers and thesis.

I am grateful to Prof. Anthony Vassallo and his research team for introducing me to the TransGrid iDemand research project, and for their insight on future electricity markets and smart grids. I also thank Prof. Eddie Anderson, The University of Sydney Business School, and Prof. Erik Schlogl, University of Technology, Sydney (UTS), for their time and suggestions on FX research.

I am deeply indebted to my parents, as without their support and encouragement I would have never accomplished anything. Thanks for always providing me with an opportunity to succeed in life, not only educationally, but emotionally, mentally, and spiritually. Finally, I would also like to thank my sister Parisa, not only for her patience and support, but for helping me in this thesis with her expertise and wisdom.

Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

ANN	**A**rtificial **N**eural **N**etwork
AR	**A**uto **R**egressive
ARIMA	**A**uto **R**egressive **I**ntegrated **M**oving **A**verage
ETS	**E**xponen**T**ial **S**moothing
FHC	**F**inite **H**orizon **C**ontrol
FX	**F**oreign e**X**change
GA	**G**enetic **A**lgorithms
GE	**G**rammatical **E**volution
GP	**G**enetic **P**rogramming
IIR	**I**nfinite **I**mpulse **R**esponse
LQ	**L**inear **Q**uadratic
LPV	**L**inear **P**arameter-**V**arying
LTI	**L**inear **T**ime-**I**nvariant
MAE	**M**ean **A**bsolute **E**rror
ML	**M**achine **L**earning
MPC	**M**odel **P**redictive **C**ontrol
MSE	**M**ean **S**quare **E**rror
NARX	**N**on-linear **A**uto **R**egressive with e**X**ogenous input
P&L	**P**rofit and **L**oss
QP	**Q**uadratic and **P**rogramming
RHC	**R**eceding **H**orizon **C**ontrol
RMSE	**R**oot **M**ean **S**quare **E**rror
SHC	**S**hrinking **H**orizon **C**ontrol
SMPC	**S**tochastic **M**odel **P**redictive **C**ontrol
SVM	**S**upport **V**ector **M**achine
SVR	**S**upport **V**ector **R**egression

Symbols

$t \in [0, 1, \cdots, N]$	Discrete time

Foreign exchange hedging:

x	Dealer's position
f	Client flow
h	Hedging action
s	Bid-ask spread
δ	Market impact coefficient
p	Logarithmic market price relative to home currency
r	Logarithmic returns
v	Volatility of returns
C	Cumulative transaction cost
L	Profit and loss
ρ	Risk of loss

Battery energy storage systems:

b	Battery charge state
u	Battery charge/discharge rate power
l	Load power
s	Solar generation power
d	Net power demand
g	Power demand from the grid
p	Price of the grid electricity per power unit
C	Total cost of the grid electricity

Chapter 1

Introduction

1.1 Aims and Motivation

Risk has been defined as "the effect of uncertainty on objectives" [1]. Any real-world system is influenced by several internal and external factors, of which only a few can be modelled accurately. The resulting uncertainty in any model causes the final outcome to deviate from the desired objective. This deviation is often undesirable, and manifests itself as monetary loss, material loss, or even loss of life.

Generally, risk management is implemented through one of the following techniques [2]:

- Risk avoidance: The easiest way to mitigate risk is to avoid it entirely. Unfortunately, this is not always possible.
- Risk transfer: Risk can be transferred to others who are willing to accept it for a premium. This is commonly done in the form of *insurance*.
- Risk reduction: By pre-planning and dynamically dealing with uncertainties, the undesirable effects of risk can be reduced.

This thesis aims to develop a system to improve risk management of a foreign exchange (FX) dealer. The FX market is a continuous, global and decentralised market where different international currencies are traded. This market enables international firms to easily convert currencies, which is required for trade and investment. Banks are an important part of this market, brokering for small institutes and big corporations with minimal costs, and thus facilitate international trade through their specialised services.

Since the early 1970s, the exchange rate between different major currencies has been continuously *floating* [3]. Rate fluctuations, measured in percentage in points (pips) and occurring in milliseconds, can cause profit and loss, and hence financial risk. While speculative traders are attracted to this risk, corporations try to apply *hedging* strategies

to manage it. An FX dealer is especially vulnerable, since he or she is not only affected by the FX rate, but also has to consider the effect of cash flow from his or her clients.

Various financial derivatives and *options* have been created to reduce risk exposure of financial bodies by transferring it to another party, at the cost of paying a *premium*. To remain competitive in the already *efficient* foreign exchange market, banks need to not only use these tools, but also employ other techniques.

A hedging strategy to manage an FX dealer's risk requires three major components:

1. A model to forecast future FX market rates.
2. A model to forecast future client transactions.
3. A hedging algorithm to decide the optimal action based on the current state and the forecasts, to minimise loss risk and transaction costs with regards to uncertainty of the forecasts.

Crafting each of these components has some unique challenges:

1. The FX market is very *efficient*, nearly *perfect*, and unpredictable[1] [3, 4]. Any prediction more accurate than a random walk can be turned into a trading algorithm to make profit [5], regardless of any trading with clients.
2. Many clients actively try to hide their trade intentions, in order to avoid arbitrage or frontrunning [6]. This is typically performed by either breaking their trades into many smaller trades, or trading with multiple dealers [7].
3. Risks can be reduced using risk management techniques, at the expense of its premium cost. Alternatively, costs can be reduced by accepting a level of risk.

The FX market shares similarities with several other inventory/storage systems with dynamic supply and demand. For example, the degree of uncertainty in a grid connected battery energy storage system (BESS) with renewable energy generation has the following similarities:

1. Renewable energy sources are *intermittent*, with generation affected by predictable cycles (e.g., no photovoltaic generation at nights) as well as uncertain factors (e.g., weather conditions).
2. Energy consumption commonly does not follow a fixed schedule and is subject to weather patterns, resident habits, or external events.
3. There are several conflicting objectives, including cost minimisation, peak shaving, and improving the life of the energy storage system, that have to be considered by the energy management system.

[1] Definitions of efficiency and perfectness for a market, and their impact on price predictability, are explored in Section 2.2.1.

Additionally, the requirements to operate in the FX market, which can be malicious and ill-conditioned, are more stringent than many other domains. Hence, many of the techniques developed or used in FX hedging can be re-purposed for other applications.

1.2 Approach and Contributions

In this thesis, model predictive control (MPC) [8] is used to deal with uncertainty. MPC is a modern control theory technique, and employs a *model* to *predict* the future states of a system over a finite time horizon. Based on this model, the controllable inputs of the system are determined such that an objective cost function is minimised, and thus the system is controlled towards the desirable outcome. Operating in discrete time, MPC constantly updates the state of the system and re-optimises the controllable inputs based on new observations. The combination of dynamic model updates with a future looking, multi-period optimisation, enables MPC to pre-plan for the best outcome while adapting to uncertainties arising from model misspecifications, prediction errors, or observation noise. As a result, this methodology creates a powerful control algorithm for handling risk.

This thesis divides the model predictive risk management into two steps: first, using stochastic model predictive control (SMPC) methodology, a stochastic shrinking horizon formulation of the underlying risky process is devised. Then, any non-trivial part of the predictive model, specially the external inputs, are treated as time-series and forecast using machine learning (ML) techniques. To control the system, an optimiser is applied to the combined predictive model to find the optimal control actions. The structure of this system is presented in Figure 1.1.

The main contributions of this work are as follows:

- A risk management framework for an FX dealer using SMPC [9, 10].
  An FX dealer's monetary losses are divided into two categories: transaction costs, which are the costs incurred in trading; and currency risk, which is the profit or loss due to fluctuations of the market rate. A stochastic model for the dealer is proposed, incorporating the client flow, FX rate volatility, and market impact. The model is carefully constructed to obtain a quadratic objective function, which is efficiently solved using quadratic programming (QP). Minimising the objective function results in an improved risk-cost Pareto frontier for the FX dealer compared to rule-based hedging strategies.
  Furthermore, using a scenario generation oracle and stochastic models, sensitivity of the SMPC framework to modelling inaccuracies and forecast errors of the client flow, FX volatility, and market impact coefficient is studied.

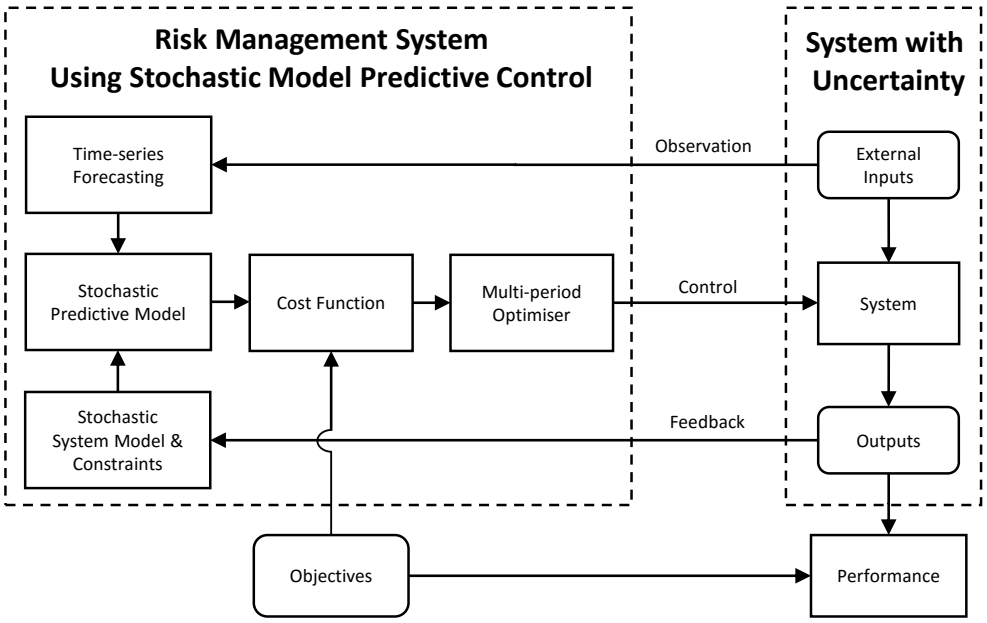


FIGURE 1.1: Architecture of the proposed risk management system using stochastic model predictive control and time-series forecasting.

We are not aware of any published previous work related to FX hedging using this methodology.

- An automation framework for non-linear time-series forecasting [11, 12].
  Although much research has been done on time-series forecasting, building a real-world forecasting model often requires an expert who should go through the time-consuming effort of trial-and-error to find the best combination of data pre-processing, feature extraction, feature selection or dimensionality reduction, model selection, model training, hyper-parameter tuning, and cross-validation. In this work, grammatical evolution (GE) is used to automate this process. First, an extensible *grammar* is defined that allows valid combinations of the aforementioned steps to be sequenced as a single forecasting program. In the second step, an evolutionary optimisation technique is used to search through the grammar space to find the optimal forecasting program.
  This framework is a generalisation of our research on expert assisted time-series forecasting, previously applied to electrical grids [13], stock market [14], and closed loop heating systems [15].
- An error measure for time-series prediction in finite horizon control applications [16].
  It was observed that tuning forecasting models using an ordinary mean square error (MSE) did not yield the optimal results with the MPC controller, and the best results were obtained through selecting the model with *backtesting*, i.e., a full simulation of the system interacting with the forecasting model and the controller.

This was formally investigated by deriving a closed-form solution for the effects of forecasting errors on the cost function of a receding horizon controller, for any linear-quadratic (LQ) system. This deviation of cost from optimal, dubbed ΔJ, was analysed on benchmark problems and was shown to be a better measure for tuning time-series prediction models.

The problem of the FX dealer, which is an LQ system, was studied with this new measure, and ΔJ was used as the GE cost function for evolutionary forecaster optimisations.

- The proposed FX hedging model and the associated forecasting techniques were applied and tested on historic FX trade data provided by Westpac Institutional Bank (Figure 1.2a).
- A similar management system, combining MPC and forecasting, was devised to control a grid connected battery energy storage system (BESS) with local renewable energy generation (Figure 1.2b). The techniques were tested on historic electricity consumption and renewable energy generation data from the TransGrid iDemand installation, and the results were compared with the iDemand's current demand management methodology.

1.3 Thesis Structure

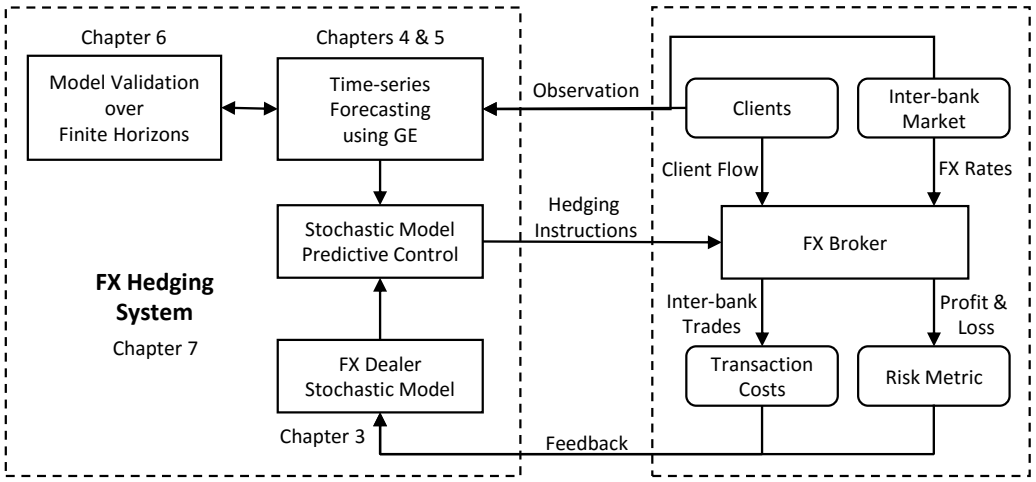
The structure of this thesis is as follows.

Chapter 2 covers the fundamentals of FX market, BESS, MPC, and GE. Several notions and techniques, including FX market hierarchy, details of an MPC controller and its applications, context-free grammars, and genetic algorithms (GA) are introduced in this chapter.

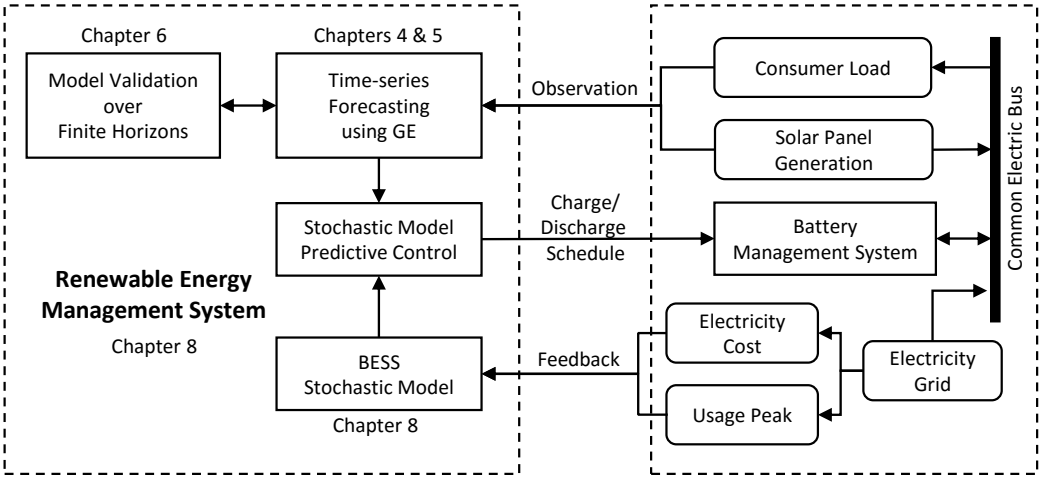
In Chapter 3, a stochastic formulation modelling an FX dealer is proposed. The formulation is then employed within an SMPC framework to offer a full risk management strategy.

Chapter 4 studies a ML-based approach to time-series prediction. Prediction is formulated as a regression task, and different pre-processing, feature selection, and dimensionality reduction techniques that can be used in this process are reviewed.

Chapter 5 describes a GE approach to time-series forecasting. A customisable template structure, in form of a context-free grammar, is devised based on the forecasting models explored in Chapter 4, and the optimal model for a data is obtained using evolutionary search within this grammar.



(A) The proposed FX hedging system.



(B) The proposed renewable energy management system.

FIGURE 1.2: Applications of the proposed risk management architecture.

Chapter 6 studies the effects of forecasting inaccuracies on finite horizon optimal control of LQ systems. A closed-form solution measuring this effect is extracted, and it is shown that this error does not necessarily correlate with common time-series error measures, such as mean square error (MSE).

Chapter 7 presents and discusses the results of the proposed forecasting and risk management techniques on synthetic and historic data in the FX market.

Chapter 8 applies the aforementioned methodologies to peak shaving and cost saving in a grid-connected battery energy storage system, in combination with local renewable energy generation. The proposed techniques are then tested on historic generation and demand data.

Finally, in Chapter 9 the work is summarised, and future research directions are suggested.

Several appendices are also included. In Appendix A, details of the R package *gramEvol* are explained. This package was created as a part of implementing Chapter 5. In Appendix B, the error measure derived in Chapter 6 is verified for the hedging formulation through analytical sensitivity testing. Appendix C expands the theory of energy management system proposed in Chapter 8 to any battery storage system. In the end, Appendix D proposes a cloud-based architecture for distributed implementation of the energy management system.

Chapter 2

Background

2.1 Introduction

This chapter establishes the theoretical foundation on which the proposed risk management methodology is based, including portfolio optimisation, model predictive control, and grammatical evolution. In addition, the foreign exchange market and its properties, as well as battery energy storage systems are discussed, and previous interdisciplinary approaches towards applying control techniques to financial and energy applications are briefly studied.

2.2 Foreign Exchange Market

The foreign exchange market, also known as the FX market and the Forex market, is the largest financial market worldwide [17], with an average daily turnover of 5.3 trillion US dollars worldwide [18], and 150 billion dollars in Australia[1] [20]. Being a global and decentralised market allows the participants to trade at working hours of different dealers, and thus keep the market open from Monday 8:00 a.m. in Sydney, Australia, to Friday 4:00 p.m. in New York, United States. This is effectively equal to five 24-hour trading days from Sunday 10 p.m. to Friday 10 p.m. GMT.

In the FX market, prices are quoted for *currency pairs*, which are named using ISO 4217 three character naming system [3]. For example, the price of Australian dollar (AUD) against United States dollar is denoted with AUDUSD, the price of Euro (EUR) against New Zealand dollar (NZD) with EURNZD, and Pound Sterling (GBP) for Japanese Yen

[1] In comparison, the worldwide gross domestic product (GDP) and Australian GDP in 2014 were 77.8 trillion dollars and 1.45 trillion dollars respectively [19].

(JPY) with JPYGBP. These six major currencies (i.e., USD, EUR, JPY, GBP, AUD, and NZD), constitute 82.9% of the market share [18]. Although commonly the *mid* market price is quoted, every pair is associated with two prices: the *bid* (or the buy) price, and the *ask* (or *offer*, or sell) price. The difference between the bid and ask price is referred to as the *bid-ask spread*, and is the main source of trading costs.

Additionally, the prices are a function of the contract maturity dates. For example, a *forward* transaction, which is to be settled on a future date, usually considers the effects of the interest rate on the contract value compared to a *spot* contract, where the delivery date is two business days from the trade date and is effectively "on the spot".

Unlike other financial markets, the FX market is *over-the-counter* (OTC) and not *exchange* based, and the trading happens directly between the participating financial institutions. Major FX players include central banks and treasuries, international banks, and major securities dealers [21]. The aforementioned entities often broker for smaller participants, including:

- corporations, that wish to access the FX market to facilitate international trade,
- retail FX dealers, which trade foreign currencies with individual clients, and
- hedge fund institution, who engage in *speculation* to profit from price movements.

The data about these trades is often confidential, and the only publicly accessible data regarding FX markets are the prices quoted by dealers. Even considering this information, the literature is divided and inconclusive whether the private information about dealers and their clients can be used to model FX rates [22–24].

2.2.1 FX Market Properties and Predictability

Due to the high number of participants and the large volume of trades, the FX market is considered both near *perfect* and very *efficient* [21].

A market is called perfect when there is *perfect competition* among its participants [25]. Characteristics of a perfect market include:

- No barriers to entry or exit: Entry to the market is available through FX dealers worldwide.
- Liquid market: There are enough participants to allow large trades.
- Market liquidity: Large trades do not cause drastic changes in prices.
- Perfect information: All participants have equal access to the information about the assets being traded.
- No single participant has the market power to set prices.
- Governments or central banks do not commonly intervene in the market.

A market is considered efficient when market prices reflect all the available information about the assets, and quickly adjust to the release of new information [26, 27].

The foremost consequence of these assumptions is a strong form of the *efficient market hypothesis* (EMH) [28], that due to equal distribution of information, there is no way to earn excess profit from the market. Closely related is the *random walk hypothesis* [29], which states that as there is no extra information available about the direction of market movements, the price evolutions follow a random walk process, and hence are unpredictable.

Lack of a better model for forecasting, compared to a random walk, has been the subject of various studies [30–32]. In a few cases, it has been shown that while some forecasting techniques could have successfully predicted the market in a certain period, their predictive properties are lost with time [33, 34]. This phenomenon, dubbed the *adaptive market hypothesis* [35], is a direct result of the EMH: with the availability of new information, including the forecasting technique methodology, the prices will adapt to include their own forecasts, and thus become unpredictable [36].

Although there are empirical observations that the EMH does not always hold for the FX market [37, 38], trading with forecasts based on these inefficiencies is not profitable [39, 40]. Furthermore, some market participants engage in criminal activities to obtain advantageous information over others, such as insider trading and using yet to be released information, participants conspiring together to change prices to their own profit, and *frontrunning*, as in the cases of 2010 Australian FX insider trading [41] and the 2013 Forex Scandal [42]. These are in addition to the common practice of exploiting *arbitrage* opportunities in the FX market [43].

Hence, not only have financial institutions become sensitive about publicising their trading strategies, they actively try to hide their intents by splitting their trade volume over time [6, 44].

2.2.2 FX Risk Management and Hedging

Exchange rates market movements can cause monetary gains and loses to FX market participants. Due to their unpredictable nature, these fluctuations pose a financial risk, better known as the *FX risk*, *currency risk*, or *FX exposure* [45]. The FX risk exposure can have enormous impact not only on the FX dealers, but also companies who trade internationally and thus the global economy. Studies have shown that even simple FX risk management protocols, such as limiting the foreign currencies held, can improve

profitability [46]. As a result, managing the FX risk, also known as *FX hedging*, is strongly encouraged by governmental bodies [47, 48].

As noted in the introduction chapter, risk management practices in any field can be divided into three groups. In the context of FX exposure risk, these practices include [45, 49]:

- Risk avoidance: FX risk can be avoided by only accepting home currency payments or working exclusively with foreign currency in international transactions. This practice may not be possible in all contexts.
- Risk reduction: An international company can either
  - add currency surcharges for foreign trades to offset price changes,
  - limit the amount of the foreign currency held, or
  - exchange foreign currency for home currency on the same day to avoid long term exposure to price volatility.

  These techniques impose additional costs, reducing international competitiveness and profitability.
- Risk transfer: Transferring currency risk is possible through FX *derivatives* [3, 50], such as:
  - Forward and futures contracts: Forward and futures contracts allow delivery of foreign currency, at a determined price, at a future date.
  - Foreign currency swap: Swap contracts allow two parties (such as multinational companies) to *swap* their cash flow, loans, or credits, in different currencies. Technically, this is equal to a spot transaction (to exchange the required *foreign* cash) and a forward transaction (to offset the spot contract exchange).
  - Currency options: An *option* allows its buyer to have the option of buying (or selling) a currency sometime in future at a set price, from the *writer* (i.e., the option seller). Using this method, the maximum loss is limited to the premium paid for the option. Different types of options are implemented, including *call* options, which allow their buyer to choose to buy a currency at a *strike* price (the pre-set designated price), and *put* options, which allow their buyer to choose to sell a currency at the strike price, or not.

To use derivatives for hedging, an appropriate derivative has to be selected such that changes in its value would offset the changes in the original inventory or investment [51]. For example, a put option's price rises when the price of its underlying asset falls. This type of hedging is widely practised in the FX market [50], and had a 337 billion dollars daily turnover in 2013 [18].

Selecting the optimal *hedging ratio*, i.e., the size ratio of a hedging contract to the original inventory, requires comparing the derivative's premium against an estimate of

future price volatility [52]. Considering the uncertainty of the market prices, as well as the dealer's FX inventory, the hedging ratio should be adjusted dynamically by buying new derivatives or liquidating the current contracts, based on price updates and the inventory sizes change [53]. An improved approach is to consider future market changes over a longer horizon. This can be accomplished by using stochastic models such as the Garman–Kohlhagen model for pricing FX options [54], as well as information about the expected future cash flow [55, 56].

An alternative to using derivatives for hedging is to directly build one using a *hedging portfolio*. One can select and hold a collection of assets, such that their market value would follow the changes in the target asset. This technique is mainly used to hedge when derivatives are not available, or the derivative itself is being hedged by the option writer. Using portfolios for hedging the risk of exchange traded assets has been thoroughly explored in the literature [57–59]. However, to the best of our knowledge, FX risk hedging using portfolios has been first formally introduced in a previous work by this author [9].

2.3 Portfolio Optimisation and Foreign Exchange

A portfolio is a collection of assets, including stocks, bonds, and cash, held by an investor. The concept of holding a portfolio instead of investing one's wealth on a single asset is historically well-known, and started as a type of risk diversification; although, until the second half of the 20^{th} century, choosing a portfolio was based on the investors' experience [60]. In 1952, Harry Markowitz published his seminal work on "Portfolio Selection". Markowitz proposed a quantitative way to model the return and risk of a portfolio, and formulated a mean-variance optimisation to select the optimal portfolio for a return-risk trade-off [61]. This technique, later known as the *modern portfolio theory*, revolutionised the quantitative finance [62].

2.3.1 Modern Portfolio Theory

Modern (or Markowitz) portfolio theory (MPT) deals with quantitative evaluation of return and risk. The fundamental concept of MPT is that the risk of a portfolio of assets is less than the risk of individual assets [61].

In MPT, selecting a portfolio is based on assigning a weight, w_i, to each asset i in the market, to determine the share of asset i in the portfolio. It is commonly assumed that

the sum of weights is normalised to one, or

$$\sum w_i = 1.$$

MPT assumes the price of an asset can be modelled by a random walk process; therefore, the *return* on the asset, i.e., profit or loss due to the change of its price, can be modelled using

$$r \sim \mathcal{N}(\mu, \sigma^2)$$

where r is the return on the asset, and $\mathcal{N}(\mu, \sigma)$ is a Gaussian distribution with mean μ and standard deviation σ.

Consequently, the expected return of the portfolio can be obtained using

$$\mathbb{E}[\sum w_i r_i] = \sum w_i \mathbb{E}[r_i] = \sum w_i \mu_i.$$

The risk of the portfolio is defined by its *volatility*, as its variance from the expected return:

$$\mathbb{V}\mathrm{ar}[\sum w_i r_i] = \sum_i w_i^2 \sigma_i^2 + \sum_i \sum_j w_i w_j \sigma_i \sigma_j \rho_{ij}$$

Here, ρ_{ij} is the correlation between r_i and r_j.

These equations can be more concisely expressed in vector form. Assuming

$$\boldsymbol{w} = [w_1 \; w_2 \; \cdots \; w_N]^{\mathrm{T}}, \text{ and}$$
$$\boldsymbol{r} = [r_1 \; r_2 \; \cdots \; r_N]^{\mathrm{T}},$$

the expected return is

$$\mathbb{E}[\boldsymbol{w}^{\mathrm{T}} \boldsymbol{r}] = \boldsymbol{w}^{\mathrm{T}} \mathbb{E}[\boldsymbol{r}] = \boldsymbol{w}^{\mathrm{T}} \boldsymbol{\mu},$$

and the volatility simplifies to

$$\mathbb{V}\mathrm{ar}[\boldsymbol{w}^{\mathrm{T}} \boldsymbol{r}] = \boldsymbol{w}^{\mathrm{T}} \mathbb{V}\mathrm{ar}[\boldsymbol{r}] \boldsymbol{w} = \boldsymbol{w}^{\mathrm{T}} \boldsymbol{\Sigma} \boldsymbol{w}$$

where $\boldsymbol{\mu}$ is the vector of the expected returns, and $\boldsymbol{\Sigma}$ is the *covariance matrix* of returns.

MPT suggests maximising the expected returns while minimising the volatility. These two objectives are combined and parameterised by $\lambda \geq 0$, the risk preference factor, into

mean-variance optimisation formalised by

$$\underset{\boldsymbol{w}}{\text{argmin}} \quad \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\Sigma}\boldsymbol{w} - \lambda \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\mu}.$$

The solutions of this optimisation form a *Pareto frontier*, also known as the *efficient frontier*. The investor can select between the minimum risk combination of assets using $\lambda = 0$, or maximise the expected returns by $\lambda \to \infty$.

2.3.2 Extensions to Modern Portfolio Theory

Since the introduction of MPT formulation, several extensions have been proposed to improve its practical results:

- Additional constraints: MPT formulation allows short selling, i.e., selling an asset without having it, using $w_i < 0$. In some markets, short selling is not allowed. This restriction is implemented by adding a constraint on weights. Similar constraints can be implemented, limiting the weight of certain assets in a portfolio [62].
- Additional terms and costs: In the real-world, investors have to pay a transaction fee to their brokers for every trade, or may be required to pay taxes on their profits [63]. MPT formulation can be improved by adding a *transaction cost* term, which specially becomes important when the assets are not liquid. This term is also useful for improving portfolio *rebalancing* (i.e., selling previous assets to buy new ones) [64].
- Alternative measures of risk: MPT assumes that the returns follow a Gaussian distribution. In practice, the distribution of returns for many stocks are fat-tailed and skewed. Variance, which is a *symmetric* measure, penalises the excess returns and losses alike, and is also less sensitive to the extreme fat-tail of non-Gaussian distributions which could incur very high losses [65].

  As a result, measures can be used to be sensitive to the distribution tails, such as
  - *value at risk* (VaR), defined by

    $$\mathrm{VaR}_\alpha(L) = \inf\{l \in \mathbb{R} : P(L > l) \leq 1 - \alpha\},$$

    which measures the α-percentile of loss distribution, i.e., the smallest threshold l, such that the probability of losses L being equal to or exceeding this threshold, is equal to or greater than α [66],
  - the *expected shortfall* (ES), also known as *conditional value at risk* (CVaR), which measures the expected losses strictly exceeding VaR [66], and

  - lower partial moments, expressed by

$$\mathrm{LPM}_n = \max\left(0, \boldsymbol{w}^{\mathrm{T}}\left(\boldsymbol{r} - \boldsymbol{\tau}\right)\right)^n,$$

    which can be used to penalise the returns below threshold τ, parameterised with order n [67].

- Alternative objectives: If the final goal of the portfolio is to track a certain derivative, for example an option, the objective of the optimisation can be changed to *absolute deviation of wealth*, i.e.,

$$\underset{\boldsymbol{w}}{\operatorname{argmin}} |\boldsymbol{w}^{\mathrm{T}}\boldsymbol{p} - t|,$$

  or *shortfall of wealth* from the desired value, formalised by

$$\underset{\boldsymbol{w}}{\operatorname{argmin}} \max(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{p} - t, 0)$$

  where $\boldsymbol{p}$ is the vector of asset prices, and t is the tracking target [58].

2.3.3 Multi-period Portfolio Optimisation

MPT formulation is a single-period optimisation, and only considers the current state of the market in its formulation; however, in reality, portfolio managers need to dynamically and actively manage their portfolio.

Historically, techniques such as *amortisation* (i.e., breaking a single large transaction to multiple smaller ones over time) was used to extend the results of the single-stage optimisation to multiple periods [68]. Nevertheless, different studies have proven that using a native multi-period optimisation is more advantageous and offers superior results over amortised single-stage optimisations in asset allocation and portfolio management problems [69, 70].

Different methodologies have been proposed to optimise multi-period portfolios [71]. Optimal asset selection using computational intelligence techniques, such as evolutionary algorithms [72, 73] and particle swarm optimisation (PSO) [74, 75], is an example of such methodologies. These algorithms can quickly approximate solutions to complex stochastic programming problems with long horizons, while handling non-convex formulations caused by inclusion of additional constraints and cost terms [76].

It has been shown, however, that using modern control theory techniques, these multi-period portfolio optimisations can be reduced to convex form and solved exactly and efficiently [77]. This class of techniques will be discussed in Section 2.5.2.

2.3.4 FX Inventory as a Portfolio

An FX dealer keeps an inventory of different foreign currencies to trade with different *counterparties*, i.e., both the participants of the inter-bank market, and his or her corporate, retail, or institutional clients.

From a dealer's point of view, this inventory is a portfolio of assets. Each FX trade is equal to buying a certain currency and selling another. For example, a dealer selling AUDUSD to a client is actually taking a *short position* (i.e., selling) on AUD and *long position* (i.e., buying) on USD. The dealer has to eventually *close* all of his or her *positions* by converting all foreign currency to his or her home currency, i.e., settle by buying back short positions from the market, and selling long positions to the market.

Due to lack of transparency, studies on the FX dealers' portfolio management practices are more scarce compared to market makers in other markets [78]. One exception is the work by Lyons [79], who analysed an FX dealer with daily volume well over 1 billion dollars over a week. It was noted that:

- Half-life of the dealer's positions is only 10 minutes on average, and is twice as long in a high volatility market compared to low volatility conditions.
- Open positions rarely rose above 40 million dollars, well below the typical 100–150 million dollars intraday position limits imposed on senior dealers at major banks.
- Before the end of every day, the dealer closed all of his positions. Lyons suggested two rationales:
  1. Carrying an open position requires monitoring it through the evening.
  2. A dealer's comparative advantage in speculation is when he is seated at his desk observing order flows and quotes.

Lyons argued that these signs show aggressive inventory management and active speculation, considering the dealer's $100,000 profit per day, compared to the average $10,000 for a similar equity dealer. In any case, no insights on the inventory management techniques were given.

Practices of FX inventory management of the Bank of Canada were analysed in [53]. The Bank of Canada "selectively" hedged its spot FX risk in derivatives market using FX forwards by varying its *hedge ratio*, i.e., the ratio of exposure to forwards risk versus spots risk, in order to take advantage of private information while adjusting to the market volatility and premium involved in hedging. Appreciation of FX risk premia in the context of hedging, based on the variation in market prices, is separately confirmed by Tien [80].

2.3.5 FX Price and Volatility Modelling

Currency prices can be modelled by studying the main driving force of the FX market. As in any other market, the supply and demand of foreign currency is the main driver of the prices, which itself is driven by international trade [81]. Several factors are used as proxies to determine a country's demand for imports and supply of exports, including mainly *macroeconomic* factors, such as GDP, trade deficit, and industrial production statistics [82]. Any announcements releasing information about these factors can cause jumps in the FX prices [83, 84], as the market players try to adapt their prices to this new information [85].

In addition to these announcement based movements, due to the high number of influencing factors and market players as described in the previous sections, ordinary price fluctuations follow a geometric Brownian motion mode [86]. Consequently, a *jump-diffusion* model [84] is commonly used to model FX rate:

$$\mathrm{d}p(t) = \mu(t)\mathrm{d}t + \sigma(t)\mathrm{d}W(t) + k(t)\mathrm{d}q(t)$$

Here, $p(t)$ is the logarithm of FX rate, and t denotes time. The components of this differential equation are:

- Drift, which models the risk free interest rate using the drift coefficient μ.
- Diffusion, modelling the ordinary prices movements using the *Wiener* process (i.e., the standard Brownian motion) $W(t)$, and the *diffusion coefficient* (i.e., the volatility) σ.
- Jumps, which account for reactions to news, including anticipated *macroeconomic news* announcements, or unanticipated releases of information. k measures the jumps' intensity and $q(t)$ is a counting process.

Empirical studies have been undertaken to quantify FX rate volatility, transaction costs and the effect of news announcements in FX market. McGroartya et al. [87] confirmed the results of many previous studies regarding the existence of intra-day patterns in the FX spot market. While the FX market is a true global market, liquidity and dealer participation depends on the geographical distribution of the currency being traded. As a result, volatility and bid-ask spreads exhibit M-shaped and U-shaped daily patterns respectively, as the markets go through the daily cycle of open, trade, and close. Evans [84] showed that jumps accompanying the anticipated events are prevalent and significant. Furthermore, approximately one third of jumps are caused by US macroeconomic news announcements, and the size of jumps is correlated with the informational surprise of the announcement. Scholtus et al. [88] found that the bid-ask spread widens and

volatility increases in the period around the announcements, which also coincides with a decrease in market depth.

2.3.6 FX Cost Model and Market Impact

A widely accepted method for measuring the transaction cost in the FX market is the bid-ask spread [89], as the *cost* of a transaction only depends on the *friction* between buying a currency from the market and selling it back. The transaction cost for an order size x can be modelled through the cost function

$$f_{cost}(x) = s_{\frac{1}{2}}|x|, \tag{2.2}$$

where $s_{\frac{1}{2}}$ is the *half spread* (i.e., bid-ask spread divided by two).

Several bid-ask spread models are proposed for exchange-based equity markets, mainly by major financial institutions (e.g., Northfield model [90], Bloomberg model, JP Morgan model, and Deutsche Bank models [91]). The main concept behind these models is measuring the *market impact* caused by an order.

In any market, the number of participants currently willing to buy or sell an asset at the current bid and ask price is limited. If a large order is executed in this market, it effectively absorbs all market liquidity at that price. The rest of participants are only willing to trade at a higher price, leading to a price rise (or in case of selling, price fall), and consequently the gap between the bid and ask price widens. This effect is often temporary, as new participants may enter to profit from the recently risen (or fallen) price, and therefore rebalancing the spread. In some cases, the price movements might be permanent. This effect of the order size on market price and spread, is known as the *market impact* [92]. To avoid this impact, and therefore reduce trading costs, traders commonly *amortise* their volume, i.e., gradually buy or sell over time [91, 92].

The equity spread models capture the market impact as a function of the order size, current and historical prices, volatility, previous bid-ask spreads, and expected daily and period trade volume [90, 91]. From the aforementioned parameters, the daily or period trade volume is usually not publicly accessible in the FX market. Additionally, considering that the FX market is OTC and is not anonymous, the quoted prices in the foreign exchange market also depend on the expectations of the provider from its counterparty: dealers are known to "skew" spreads favourable to their current positions and vary prices based on their trade history with each counterparty [93]. Other evidences also show that the relation between the volatility and the spreads are due to the number of trades [84, 88], and both follow a certain daily cycle [87].

2.4 Battery Energy Storage System

A battery energy storage system (BESS) is a device which stores electrical energy in one or more electrochemical cells [94]. Batteries have several advantages over other energy storage systems, such as pumped hydroelectric or compressed air energy storage systems: batteries respond rapidly to load changes, and have high energy efficiency and low standby losses. The main disadvantages of batteries are high manufacturing costs and shorter life spans [95].

Historically, using battery storage with the electrical grid was debated as its only practical case was to improve the power quality (i.e., voltage depressions and power interruptions) [96]. With wide scale adoption of renewable energy systems (RES), including wind and solar power, the landscape of energy generation and consumption has changed, resulting in new interests towards BESS [97]: wind and solar RES are intermittent energy sources, with generation affected by predictable cycles (e.g., no solar generation at nights) as well as uncertain environmental factors (e.g., weather conditions). Using BESS allows the generated energy to be stored and used on-demand [98].

It has been shown that while RES are good at reducing dependence on the electrical grid, *going off-grid* (i.e., completely disconnecting from the grid) is not an economical choice, and the best strategy is staying connected while minimising the demand from the grid [99].

Different aspects of using BESS, in combination with RES, *smart grids*, and even as off-grid, have been widely studied. The approaches commonly use stochastic information management schemes to model the uncertainty in generation, consumption, and electricity pricing [100]. Examples include determining the optimal battery size with regards to the uncertainty of PV solar generation and consumer power requirements, against its economic benefits [101–103].

2.4.1 Batteries in Grid Connected Systems

In *grid connected* systems, BESS can be used to reduce grid congestion during peak hours [104]. Grid demand, i.e., consumption minus renewable generation, is not evenly distributed during the day. Consumption peaks in the evening, which coincides with the nightly decline in photovoltaic (PV) solar generation. Considering that RES have a stochastic nature and are not controllable, congestion management has been implemented through two different approaches:

- Load management, where the consumption is shifted to off-peak hours by scheduling the load [105, 106].

- BESS Scheduling: A BESS can be charged during off-peak hours, filling the demand *valleys*, and then discharged to *shave* the *peaks*, resulting in a flat demand profile [107].

The first approach is intrusive to the consumers' habits, and requires significant investment for automating load management through *advanced metering infrastructure* (AMI) [108] and *smart appliances* [109]. The second approach only requires connecting the battery to the grid through an inverter [110], and the rest is handled by scheduling algorithm software.

If the demand profile is known in advance, the optimal peak shaving schedule can be derived analytically [111]. As the demand is a stochastic process, dynamic scheduling, i.e., constantly re-optimising the energy dispatch schedule based on recent observations and model updates, is a more realistic approach for real-world implementations [112]. Different optimisation schemes have been proposed, including dynamic programming [113], mixed integer linear programming [114], chance constrained programming [115], or even fuzzy approaches [116, 117]. It has been shown that utilising forecasts, even using naïve techniques, can significantly improve peak shaving [118, 119].

2.5 Model Predictive Control

Model predictive control (MPC), also known as finite horizon control (FHC), is a process control technique that uses multi-period optimisation over a finite horizon [8].

The idea of MPC [120] (Figure 2.1) can be summarised as follows:

- A *model* is available to *predict* the future behaviour of a system towards its inputs, over a *finite time horizon*. The predictive model is typically represented using a linear state-space model.
- By minimising a *cost function*, which incorporates the predictive model, the optimal input sequence to the system over the finite horizon can be found. Commonly, the cost function is of quadratic form.
- The system is controlled in discrete time using a digital computer; hence, after applying the first set of inputs, the model can be updated based on new observations to include unaccounted changes, due to external disturbances, observation noise, and/or predictive model inaccuracies.
- At the next time step, optimisation is repeated for the new horizon using the updated model.
- The sequence of
    1. apply inputs for current time-step

2. update the model
3. optimise cost function

is repeated until the process ends.

The model update in MPC acts as a feedback pathway to the *controller*, and thus this technique exhibits a robustness typical of closed-loop control systems [8]. The downside of using MPC is the computational power required for re-optimising the cost function at each time-step.

Selecting the horizon at each new time-step takes two main forms:

- Receding horizon: The horizon is shifted one time-step forward, and the same horizon length is kept. This sliding technique is known as *receding* horizon control (RHC) (Figure 2.2a).
- Shrinking horizon: If the optimisation objective has a strict deadline, the new horizon begins one time-step forward while the same termination time is held. This causes the length of the horizon to shrink as time advances (Figure 2.2b). Consequently, this technique is referred to as *shrinking* horizon control (SHC) [121].

MPC methodology is well studied [8, 122, 123] and widely adopted in different engineering applications and industries [124]. Additionally, since the underlying idea of using a predictive model for determining the optimal future actions is inherent to many real-world problems, a similar methodology is studied under different names for inventory management [125] and dynamic scheduling [126] problems.

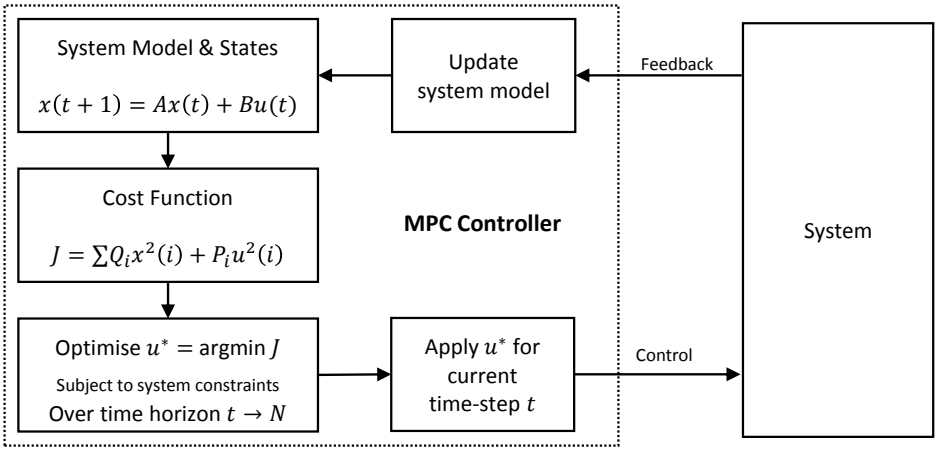
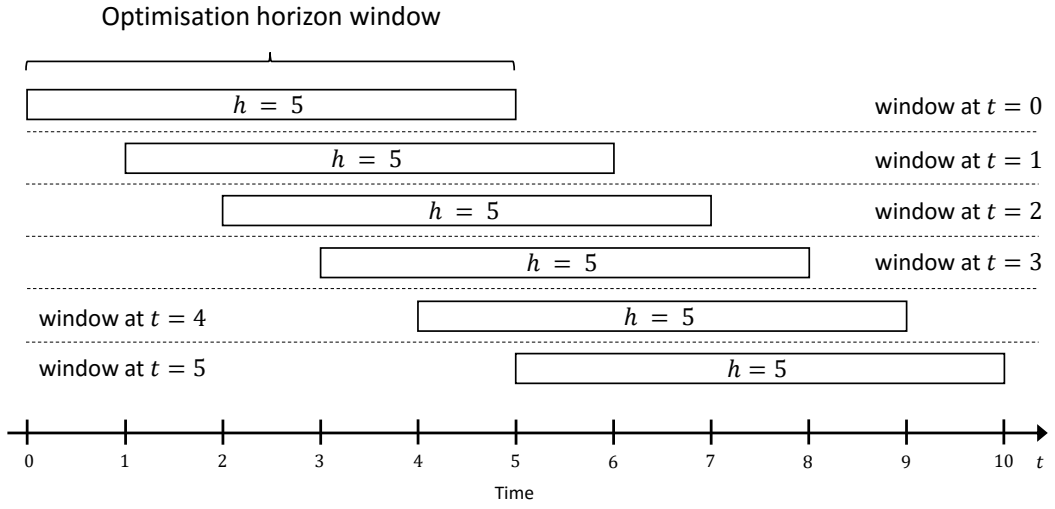
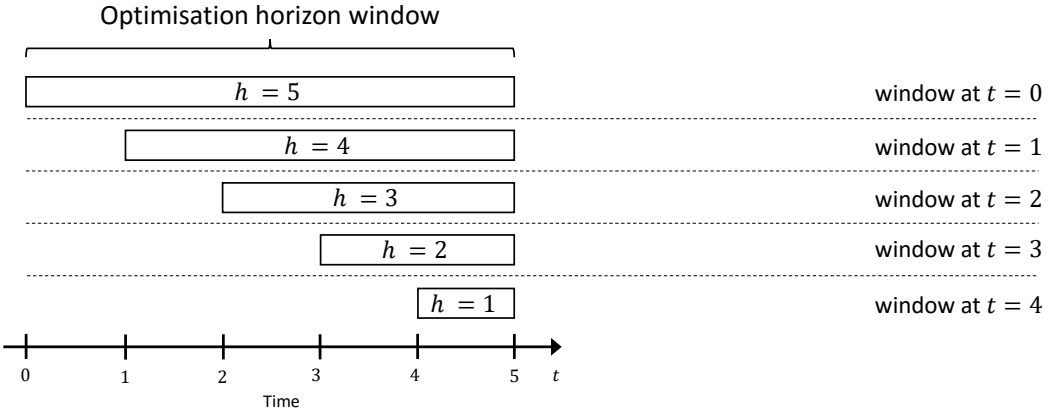


FIGURE 2.1: Architecture of an MPC controller. The system has a linear state-space representation, with states x and inputs u. The cost function is quadratic and is parameterised by P_t and Q_t, where t denotes time. The optimiser finds the optimal control actions u^* over a finite horizon t to N, but only $u^*(t)$ for the current time-step t is applied, and the rest are discarded. At the next time-step, the system model is updated based on new observations, and the optimisation is repeated for the new time horizon.



(A) Receding horizons.



(B) Shrinking horizons.

FIGURE 2.2: Receding versus shrinking horizon control. In receding horizon control, the optimisation horizons are shifted forward at each time-step t. In shrinking horizon control, the start of the horizon moves forwards, but its termination time does not change, and as a result the horizon shrinks.

2.5.1 Stochastic MPC

An important extension to MPC is stochastic model predictive control (SMPC). Normally, MPC problems are studied for linear time-invariant (LTI) systems; extending this to uncertain systems, SMPC introduces random variables into the optimisation, and is more versatile in cases where a deterministic model of the underlying system is not available. For example, controlling a linear parameter varying (LPV) system, where the parameters of the linear system have a stochastic nature, can be posed as a SMPC problem [127].

Ordinarily, such a problem requires solving a multi-stage stochastic dynamic programming, and it quickly becomes infeasible to solve as the horizon length grows. Recent literature has proposed *sub-optimal* techniques for solving this problem: by discretising

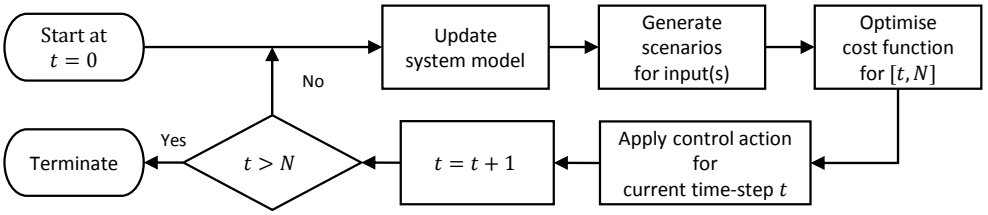


FIGURE 2.3: Flowchart of stochastic shrinking horizon MPC, operational during time-steps $t \in [0, 1, \cdots, N]$. Compared to an ordinary MPC, stochastic MPC includes an additional scenario generation step.

the stochastic variable evolution to a finite number of *scenarios*, the problem is approximated by a random convex program [128]. For linear-quadratic systems, this technique which is alternatively referred to as *random* model predictive control (RMPC) [129] or *scenario-based* model predictive control (SCMPC) [130], translates to a quadratic programming (QP) formulation, and is solved efficiently using numeric solvers. SCMPC has been shown to be stable [131], with analytic bounds on the number of scenarios required to obtain any level of robustness [132, 133]. Moreover, the computational complexity of this method scales quadratically with the length of control horizon and does not depend on the uncertainty/disturbance dimensions [134]. Flowchart of this algorithm is shown in Figure 2.3.

Scenarios allow a versatile range of models to be adopted in simulations, compared to the limited nature of analytical formulations. Hence, scenario generation techniques have been widely studied [135], especially for use in Monte Carlo techniques [136], and are applied to different aspects of financial risk management [137, 138].

2.5.2 MPC Applications in Finance

There has been a growing interest in using stochastic MPC in many financial applications, as this technique is a natural solution for multi-period stochastic systems with uncertain models.

Portfolio optimisation and asset allocation is one such application. A receding horizon approach for portfolio selection, under the assumption of zero transaction cost, has been formulated by Herzog et al. [139]. Topaloglou et al. [140] included currency forwards in an international stocks portfolio to combine hedging and portfolio allocation using dynamic stochastic programming. Primbs [141] considered both problems of wealth maximisation and index tracking for a portfolio with different constraints using SRHC. Calafiore [77] used a RHC strategy for portfolio allocation. By employing affine policies he managed to convert the stochastic programming problem to a convex quadratic programming

problem. In further publications, this method was enhanced with transaction costs [142] and asymmetric measures of risk [67].

Options hedging is another area where RHC approaches have been extensively used. Gondzio et al. [143] extended the simple delta-vega hedging approach for hedging contingent claims using stochastic optimisation. Hedging a basket of European call options using receding horizon techniques has been studied both with a mean-variance model approach [57] and a scenario based approach [144]. In their other work, Meindl and Primbs [145] incorporated a utility function as a part of their optimisation function for option hedging. Recent works by Bemporad et al. [58, 146, 147] have shown that hedging options using stochastic model predictive control can perform extremely well, with performance approaching that of prescient hedging models for European style options.

Other applications of dynamic stochastic programming to FX cash flow risk management include the work by Volosov et al. [55], where spot and forward rates random behaviour were predicted by a vector error correction model and used in a two-stage stochastic programming model. This technique, however, assumed the foreign cash flow to be deterministic.

2.5.3 MPC for Optimal BESS Scheduling

As noted before, dynamic scheduling is another term for model predictive control (MPC) methodology. Hence, MPC and SMPC have been mentioned explicitly and utilised extensively in the BESS optimal scheduling problem, as well as many other energy system applications. Approaches mainly differ in assumptions, system configuration, and objectives. Variations include different stochastic models for wind only, solar only, or hybrid combinations of renewable energies [148], utilising solar radiation forecasts versus naïve PV models [149], modelling consumption behaviour through Markov chains [150], considering lossy batteries [151], including thermal energy storage (TES) in addition to batteries [152], formulating the problem through scenario-based stochastic programming SMPC [153], or utilising random stochastic MPC (RMPC) implementations [154].

2.6 Grammatical Evolution

Grammatical evolution (GE) [155] is an evolutionary optimisation technique for generating complete programs, optimised towards performing a certain task. GE is formed by combining the ideas from context-free grammars (CFG) [156] and genetic algorithms (GA) [157].

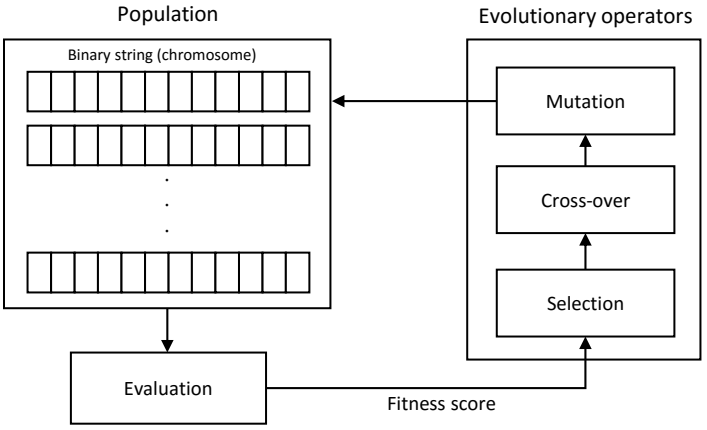


FIGURE 2.4: The evolutionary optimisation process and associated operators.

GE is an alternative to genetic programming (GP) [158] for generating programs via evolution. While GP directly operates on the actual program's tree structure, GE applies evolutionary operators on binary strings which are subsequently converted to the final program. GP normally requires a custom search strategy to generate correct programs, whereas GE can utilise an unconstrained evolutionary search, relying on the grammar to generate correct programs.

Customisability of the grammar in GE allows quick and easy integration of domain-specific knowledge into an optimisation problem. GE has been successfully applied to many research areas in science and engineering, including computational finance and smart grid forecasting. A survey by McKay et al. [159] discusses the range of GE research and applications.

In this section, the components of GE, i.e., CFG and GA, are studied. Consequently, the architecture of a GE optimiser is presented and discussed.

2.6.1 Genetic Algorithm

Genetic algorithm (GA) [157] is an optimisation algorithm which operates on a *population* of *chromosome*s, performing evolutionary operations including selection, crossover, and mutation as illustrated in Figure 2.4. Inspired by biological evolution, GA has been successfully used in applications with complex fitness landscapes and multiple local optimas [160].

In canonical GA, a chromosome is represented by a binary string. Normally, modern GA implementations do not directly operate on binary values; instead, bits are grouped into n-bit values creating a *codon*, each of which is used as a parameter in the optimisation objective. If the problem is made of multiple building blocks, codons related to

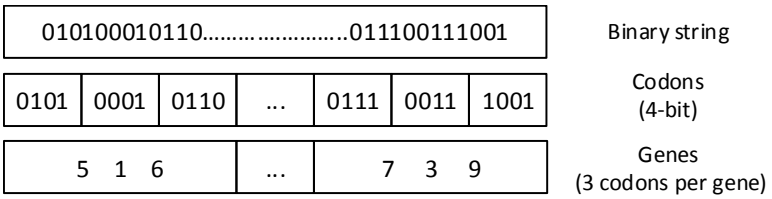


FIGURE 2.5: Chromosome representations in GA. The underlying data is stored as a binary string, while it is logically divided to codons and genes.

each block are grouped together as a *gene* (Figure 2.5). This arrangement is a logical presentation of data and does not affect the low level representation of the chromosome.

The initial population is created from randomly generated chromosomes, each representing a solution to the formalised problem. The chromosomes are then evaluated based on a given *cost function*, $\phi(\cdot)$. The objective is to minimise the cost function, or maximise the *fitness* of the chromosome. The better scoring chromosomes are deemed to be more desirable, and hence their data is retained. Conversely, the low scoring chromosomes are discarded from the population and replaced with new chromosomes to form a new *generation*. *Elitism* favours the highest ranking chromosomes and directly forwards them to the new generation's population. Others are created by a recombination of *selected* chromosomes.

The *selection* operator is applied to select chromosomes with a likelihood proportional to their fitness score. Different selection schemes exist, including roulette wheel selection and tournament selection. In roulette wheel selection, the probability of selecting the ith chromosome, denoted with b_i, follows a Bernoulli distribution by $p = \phi(b_i) / \sum_{j=0}^{n} \phi(b_j)$.

The *crossover* operator is applied on two randomly selected chromosomes. In canonical GA, a single-point crossover is used, where a position in the binary string is chosen at random and the opposing string sections of the two parents are exchanged, creating two new offsprings.

The *mutation* operator randomly flips single bits on a specific chromosome with a predefined mutation probability. Mutation is necessary to maintain genetic diversity from one generation of a population to the next.

The evolutionary process is repeated until a given termination criterion is satisfied. This criterion may include reaching a pre-determined number of generations, finding a chromosome with fitness better than a desirable minimum, or a lack of improvement in the population fitness despite evolution.

Since its introduction in 1975 [161], other techniques and evolutionary algorithms have been proposed to extend the canonical GA. For example, to facilitate complex data

representation, GA is often implemented with integer or floating point codons, and evolutionary operators are applied directly to the codons instead of the underlying bit string. This method also takes advantage of the architecture of modern processors to speed-up computation. For a review of other GA techniques, readers are referred to a survey by Srinivas and Patnaik [162].

2.6.2 Context-free Grammars

Grammar is a set of rules that describes the syntax of sentences and expressions in a language. While grammar was originally invented for studying natural languages, they are extensively used in computer science for describing programming languages.

A context-free grammar (CFG) is a mechanism to generate patterns and strings using hierarchically organised production rules [163]. A CFG is described by the tuple $(\mathcal{T}, \mathcal{N}, \mathcal{R}, \mathcal{S})$, where $\mathcal{T}$ is a set of terminal symbols, $\mathcal{N}$ is a set of non-terminal symbols with $\mathcal{N} \cap \mathcal{T} = \varnothing$, and $\mathcal{S} \in \mathcal{N}$ is the *start* symbol. A non-terminal symbol is one that can be replaced by other non-terminal and/or terminal symbols, while terminal symbols are literals. $\mathcal{N}$ and $\mathcal{T}$ form the lexical elements used in $\mathcal{R}$, the production rules of a CFG. $\mathcal{R}$ is defined as a set of relations (also referred to as production rules) in the form of $x \to \alpha$ with $x \in \mathcal{N}$, $\alpha \in (\mathcal{N} \cup \mathcal{T})^*$, where $*$ is the Kleene star. If the grammar rules are defined as $\mathcal{R} = \{x \to xa, x \to ax\}$, a is a terminal symbol since no rule exists to change it.

CFGs are commonly described using Backus–Naur form (BNF) [156]. To differentiate between terminal and non-terminal symbols in the BNF, the non-terminal symbols are enclosed within angle brackets (i.e., '<' and '>'). Furthermore, in each production rule, possible replacement sequences are separated by a vertical bar (i.e., '|').

An example of grammar in BNF notation is given in Table 2.1. In this grammar, the start symbol ($\mathcal{S}$) is <*expr*>. Each of the non-terminal symbols defined in $\mathcal{N}$, <*expr*>, <*op*>, <*coef*> and <*var*>, can be replaced by an appropriate terminal as specified in $\mathcal{R}$. For example, <*expr*> can either expand to (<*expr*>)<*op*>(<*expr*>) or <*coef*>×<*var*>, and <*op*> can be replaced by one of the +, -, ×, or ÷ operators.

2.6.3 Genotype to Phenotype Mapping using Grammar Rules

To combine grammars with GA optimisation, a way is required to convert the numeric chromosome data, through the grammar, to a program. Notice that by a *program*, we refer to any sequence of instructions that perform a specific task. This ranges from a single

expression (e.g., `sin(x)`), to several statements with function declarations, assignments, and control flow.

In evolutionary biology, chromosome data is referred to as the *genotype*, while an organism's observable characteristics are called the *phenotype*. Biological organisms use complicated methods to *map* their genotype to phenotype. Advanced evolutionary algorithms, such as GE, use a similar notion.

In GE, genotype to phenotype mapping is performed according to the production rules of a CFG selected using the chromosome's codon values. The usual mapping function used is the `mod` rule defined as

$$\text{(codon integer value)} \ \texttt{mod} \ \text{(number of rules for the current non-terminal)},$$

where `mod` is the modulus operator. Mapping begins from the start symbol $\mathcal{S}$, and continues by replacing each non-terminal element (defined in set $\mathcal{N}$), according to the production rule $\mathcal{R}$ chosen by the mapping function. At each step, the resulting expression can contain terminal (defined in set $\mathcal{T}$) or non-terminal elements. The mapping continues until all non-terminal elements are replaced with terminals.

If the chromosome is too short, it may run out of codons with non-terminal elements still remaining. A common approach is to *wrap* the chromosome and continue the mapping process by reusing the codons from the beginning. However, in cyclic grammars, infinite recursion may occur. This is addressed by introducing a limit on the number of allowed chromosome wrappings and assigning a poor fitness score to the chromosome if the limit is reached.

In Table 2.2, using this technique, an example of program generation is shown. For this purpose, the grammar of Table 2.1 is used. Consider the chromosome with a 16-bit

TABLE 2.1: An example grammar in BNF notation. The three first lines define the non-terminal ($\mathcal{N}$), terminal ($\mathcal{T}$), and start ($\mathcal{S}$) symbol sets respectively. The rest of the lines define the production rules ($\mathcal{R}$).

$\mathcal{N} = \{expr, op, coef, var\}$
$\mathcal{T} = \{\div, \times, \texttt{+}, \texttt{-}, v_1, v_2, c_1, c_2, \texttt{(}, \texttt{)}\}$
$\mathcal{S} = <expr>$

$\mathcal{R}$ = Production rules:

$\langle expr \rangle$	::= $(\langle expr \rangle)\langle op \rangle(\langle expr \rangle) \mid \langle coef \rangle \times \langle var \rangle$	(1.a) (1.b)
$\langle op \rangle$	::= $\texttt{+} \mid \texttt{-} \mid \times \mid \div$	(2.a), (2.b), (2.c), (2.d)
$\langle coef \rangle$	::= $c_1 \mid c_2$	(3.a), (3.b)
$\langle var \rangle$	::= $v_1 \mid v_2$	(4.a), (4.b)

TABLE 2.2: Production of an expression using the grammar of Table 2.1. The process starts from the start symbol $\mathcal{S}$, and continues by replacing the first symbol present in $\mathcal{N}$ with another. This later symbol is selected from the production rules $\mathcal{R}$ according to the value of the current codon. In 8 steps, all of non-terminal symbols are replaced and the string $\{2, 1, 0, 0, 3, 3, 3, 1\}$ is mapped to $(c_1 \times v_1) \div (c_2 \times v_2)$.

Step	Codon	`mod` operator	Rule	Current element state
0			$\mathcal{S}$	`<expr>`
1	2	2 `mod` 2	0 (1.a)	(`<expr>`)`<op>`(`<expr>`)
2	1	1 `mod` 2	1 (1.b)	(`<coef>`×`<var>`)`<op>`(`<expr>`)
3	0	0 `mod` 2	0 (3.a)	(c_1 ×`<var>`)`<op>`(`<expr>`)
4	0	0 `mod` 2	0 (4.a)	(c_1 × v_1)`<op>`(`<expr>`)
5	3	3 `mod` 4	3 (2.d)	(c_1 × v_1) ÷ (`<expr>`)
6	3	3 `mod` 2	1 (1.b)	(c_1 × v_1) ÷ (`<coef>`×`<var>`)
7	3	3 `mod` 2	1 (3.a)	(c_1 × v_1) ÷ (c_2 ×`<var>`)
8	1	1 `mod` 2	1 (4.a)	(c_1 × v_1) ÷ (c_2 × v_2)

genotype, $\{2, 1, 0, 0, 3, 3, 3, 1\}$, where the integer numbers represent 2-bit codon values. There are two production rules to choose from for the start symbol $\mathcal{S} = <expr>$: (1.a) and (1.b). The `mod` operation on the current codon becomes 2 `mod` 2 = 0, hence rule (1.a) is chosen, and the expression transforms to (`<expr>`)`<op>`(`<expr>`). The successive application of rules continues for eight steps, when all non-terminal elements are exhausted. The resulting phenotype, $(c_1 \times v_1) \div (c_2 \times v_2)$, can be later evaluated in different contexts as a numerical value.

2.6.4 Evolving a Grammar

As explained before, the objective of GE is to automatically generate a program that minimises a cost function, by combining CFGs and GA:

1. A grammar is defined to describe the syntax of the programs.
2. A cost function is defined to assess the quality (i.e., the cost or fitness) of a program.
3. GA is used to search within the space of all programs definable by the grammar, translated from chromosome data through genotype to phenotype mapping, in order to find the program with the lowest cost.

The flow of information between different components of GE is presented in Figure 2.6.

One must note that the overall cost function, which includes the genotype to phenotype mapping as well as the user defined cost function, i.e., mapping a binary string to a program and subsequently to a numeric score, is often non-smooth and non-convex, precluding gradient-based optimisation algorithms and favouring evolutionary optimisation techniques.

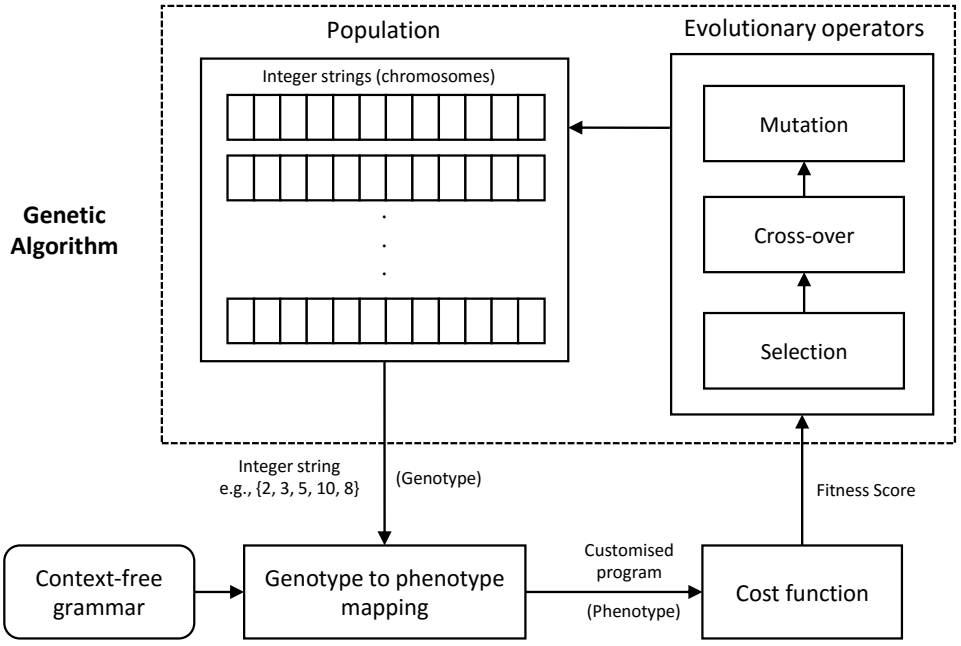


FIGURE 2.6: Architecture of a grammatical evolution optimiser.

2.7 Summary

This chapter reviewed the theoretical background and collected the prior art related to the objectives of this thesis:

- Foreign exchange (FX) market, the largest financial market worldwide, is global and decentralised, and as a result exhibits different properties from exchange-based markets. Many concepts, including the FX market hierarchy, previous research on exchange rate predictability, market impact, transaction costs, volatility, and currency risk were introduced, and common financial techniques including portfolio optimisation and hedging (i.e., risk management) using financial options were explained.
- Battery energy storage systems (BESS) are important components of modern smart grids. The main application of batteries is to increase reliability of intermittent energy sources, including solar and wind power, and to shave peak demand by storing electricity at off-peak periods and dispatching energy during high demand.
- Model predictive control (MPC) is a modern process control technique. The central idea of MPC is using a model to predict the future outcome of a system. By minimising a cost function, which incorporates this predictive model, the optimal input sequence to the process over a finite horizon can be found. This optimisation, performed in discrete time, is repeated at each time-step by sliding the horizon towards the future, or shrinking it. Hence, MPC is also known as finite horizon control, receding horizon control, and shrinking horizon control.

- Grammatical evolution (GE) is an evolutionary optimisation technique for creating programs optimised for a user defined task. It combines context-free grammars (CFG) as a template for generating programs, and genetic algorithm (GA) for finding the optimal program in the grammar space.

The following chapters will utilise these techniques to model, predict, and manage risk.

Chapter 3

Stochastic Model Predictive Control for Short-Term Risk Management in Foreign Exchange

3.1 Introduction

Foreign exchange (FX) dealers facilitate international trade by connecting corporate, retail, and institutional clients, with the inter-bank FX market. Unexpected or even anticipated fluctuation in FX rates, combined with an accumulation of large positions absorbed from their clients' trading flow, can create a significant risk for these dealers. In this chapter, a methodology is formalised to manage this risk over short time periods in the FX spot market.

3.2 Assumptions

We define the specific problem of an FX dealer, who accepts flow from corporate, retail, and institutional clients. The dealer wishes to *hedge* the risk of his positions only by opening or closing new positions in the spot market. It is assumed:

- The dealer does not perform speculation and is only interested in hedging.
- The dealer makes decision in discrete time-steps.
- There is a limit on how much the dealer can trade at each time-step.
- There is a limit on how much open position the dealer is allowed or willing to keep at each time-step.

- The dealer must close all positions at the end of a trading session (e.g., daily or weekly). This is common practice to avoid carrying an open position's risk during non-business hours [79].
- Market impact affects the inter-bank market's bid-ask spread [164].
- Client flow is always initiated by the clients, and the dealer is unable to refuse them, assuming credit status and other conditions (e.g., anti-money laundering laws) are satisfied.
- The dealer works with local clients (either corporate, retail, or institutional), under their own geographic time zone. These types of clients usually buy more foreign currency than sell [165].
- FX rate jumps resulting from macroeconomic news announcements are a large contributor to the dealer's risk [84]. The timing of these events is known, but the direction and magnitude of the jumps are not.

The dealer's profit and loss (P&L) arises from three major sources:

1. Transaction costs received from clients.
2. Transaction costs paid to inter-bank market counterparties for hedging trades.
3. Market volatility.

Here, we assume that the profit from transaction costs received from clients is not influenced by hedging. As a result, they are not considered and only volatility and transaction costs paid to inter-bank market counterparties are included in the optimisation formulation.

3.3 Problem Formulation

3.3.1 Dealer Dynamics

We formulate the problem in discrete time notation, with the dealer trading at $t \in [0, 1, ..., N]$. The trading session ends at $t = N + 1$. At any time t, the dealer holds a position of $x_k(t) \in \mathbb{R}$ for currency k. The dealer's initial position is denoted by $x_k(0)$. Positions vary in time as a result of the accumulated client flow $f_k(t) \in \mathbb{R}$ and the dealer's hedging actions $h_k(t) \in \mathbb{R}$ (Figure 3.1):

$$x_k(t+1) = x_k(t) + h_k(t) + f_k(t) \tag{3.1}$$

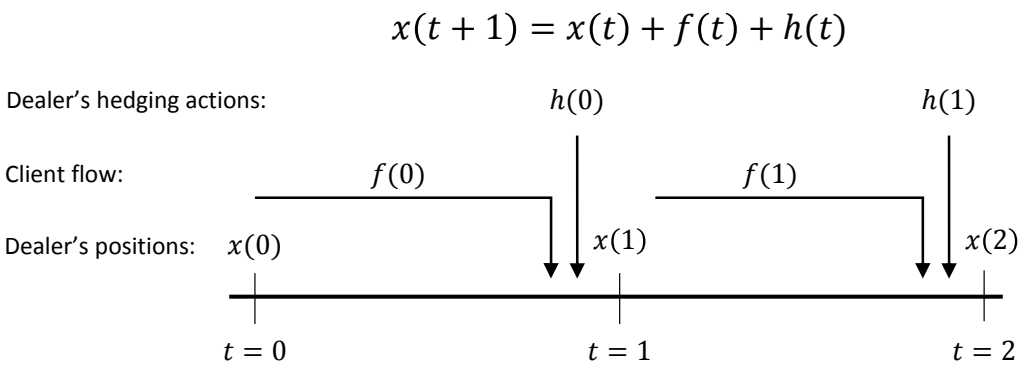


FIGURE 3.1: Hedging state-space dynamics. The dealer's position at time $t+1$, $x(t+1)$, is a result of his position at time t being updated with the accumulated client flow in this period $f(t)$ and hedged with $h(t)$.

The dealer must close all positions at the end of a trading session (e.g., daily or weekly), and therefore $x_k(N+1) = 0, \forall k$. As a result, the last hedging action has to be

$$h_k(N) = -(x_k(N) + f_k(N)). \tag{3.2}$$

At time t, the cumulative transaction costs paid by the dealer for hedging is

$$C_k(t) = \sum_{i=0}^{t} f_{cost}(h_k(i), i, k), \tag{3.3}$$

where $f_{cost}(h, t, k) = s_k(h, t)|h|$ is a time dependent transaction cost function based on the bid-ask spread $s_k(h, t)$ quoted for trading h at time t, and $|h|$ is the size (i.e., absolute value) of h. The spread is known to be highly volatile, and is affected by market impact (i.e., increases with the size of h), market conditions (e.g., the spread decreases in liquid markets), and the history of trades between counterparties. The spreads are modelled as having a linear relationship with the size of trade [164],

$$s_k(h, t) = \delta_k(t)|h| \tag{3.4}$$

where $\delta_k(t)$ is the time-varying market impact coefficient. This results in a quadratic cost function:

$$f_{cost}(h, t, k) = \delta_k(t)h^2 \tag{3.5}$$

Additionally, we limit the size of hedging actions to $h_{k,max}$:

$$|h_k(t)| \leq h_{k,max} \tag{3.6}$$

This is realistic as the dealers will have trading limits with their counterparties in the inter-bank market and usually wish to avoid impacting the market's liquidity beyond a certain degree.

FX rate volatility changes the value of positions, causing profit and loss. We denote the logarithmic market price of currency k with $p_k(t)$ and its returns $r_k(t)$ with

$$r_k(t) = p_k(t) - p_k(t-1).$$

The profit and loss (P&L), $L_k(t)$, is therefore given by

$$L_k(t) = \sum_{i=1}^{t} x_k(i) r_k(i). \tag{3.7}$$

The dealer usually wishes to avoid the risk of excessive exposure, and thus the maximum position of each currency is restricted to

$$|x_k(t)| \leq x_{k,max}. \tag{3.8}$$

3.3.2 Definition of Risk

We define FX risk as the unpredictable loss of the dealer's portfolio value due to FX rate changes.

Based on the efficient market hypothesis, we assume that predicting the direction of FX price changes, i.e., the sign of $r_k(t)$, is not feasible: formally speaking, $r_k(t)$ has a symmetrical probability distribution, and is zero-mean (i.e., $\mathbb{E}[r_k(t)] \approx 0$). As a result, the only available information regarding future price changes is their variance,

$$v_k(t) = \mathbb{V}\mathrm{ar}[r_k(t)],$$

which is also known as the *volatility*.

This assumption considers any profitable FX rate change to be undesirable, but as stated in Section 3.2, the dealer's objective is only hedging, and speculative trades can be performed irrelevant to the incoming client flow.

Consequently, we characterise the dealer's risk exposure by the variance of P&L, and define it using

$$\rho_k(t) = \mathbb{V}\mathrm{ar}[L_k(t)]. \tag{3.9}$$

3.3.3 Hedging Objectives

Dealers wish to reduce their transaction costs and risk exposure, subject to the discrete space representation and constraints defined in this section. Considering the probabilistic nature of client flow and FX rate, we formulate this as a mean-variance optimisation,

$$\begin{aligned} \underset{h_k(t);\ \forall t,k}{\text{argmin}} \quad & \mathbb{E}\left[\sum_{k=1}^{m} C_k(N)\right] + \lambda \mathbb{V}\text{ar}\left[\sum_{k=1}^{m} L_k(N)\right] \\ \text{subject to} \quad & x_k(N+1) = 0 \\ & |h_k(t)| \leq h_{k,max} \\ & |x_k(t)| \leq x_{k,max} \end{aligned} \tag{3.10}$$

where m is the number of currencies in the dealer's portfolio, and $0 \leq \lambda < \infty$ is the risk aversion factor.

This minimisation requires values of $r_k(t)$, $f_k(t)$, and $\delta_k(t)$. From these, only variables at $t = 0$ are observed, and the rest have to be modelled by stochastic variables. In this thesis, we use *prediction* (and equally *forecasting*) to refer to statistical or machine learning techniques that determine a *future* stochastic variable's probability distribution or statistical properties (e.g., mean and variance).

Optimising (3.10) results in a risk-cost Pareto frontier parameterised by λ. Using this information, the dealers choose the best hedging strategy considering contextual factors, such as their utility of risk.

3.3.4 Matrix Notation

To simplify notation, we introduce variable $y_k(t)$, as the *unhedged position* of the dealer:

$$\begin{cases} y_k(0) = x_k(0) \\ y_k(t) = \sum_{i=0}^{t-1} f_k(i) + x_k(0) \end{cases} \tag{3.11}$$

We use the following matrix notation to simplify representation of the dealer dynamics. Vectors $\boldsymbol{x}_k, \boldsymbol{f}_k, \boldsymbol{h}_k, \boldsymbol{\delta}_k, \boldsymbol{r}_k,\ \boldsymbol{y}_k \in \mathbb{R}^N$ are the collection of variables $x_k(t),\ f_k(t),\ h_k(t),$

$\delta_k(t)$, $r_k(t)$, and $y_k(t)$ in time:

$$\begin{aligned}
\boldsymbol{x}_k &= [x_k(0)\ x_k(1)\ \cdots\ x_k(N-1)]^{\mathrm{T}} \\
\boldsymbol{f}_k &= [f_k(0)\ f_k(1)\ \cdots\ f_k(N-1)]^{\mathrm{T}} \\
\boldsymbol{h}_k &= [h_k(0)\ h_k(1)\ \cdots\ h_k(N-1)]^{\mathrm{T}} \\
\boldsymbol{\delta}_k &= [\delta_k(0)\ \delta_k(1)\ \cdots\ \delta_k(N-1)]^{\mathrm{T}} \\
\boldsymbol{r}_k &= [r_k(1)\ r_k(2)\ \cdots\ r_k(N)]^{\mathrm{T}} \\
\boldsymbol{y}_k &= [y_k(1)\ y_k(2)\ \cdots\ y_k(N)]^{\mathrm{T}}
\end{aligned}$$

Notice that $\boldsymbol{r}_k$ and $\boldsymbol{y}_k$ are indexed differently from the others.

Vectors $\boldsymbol{X}, \boldsymbol{H}, \boldsymbol{F}, \boldsymbol{\delta}, \boldsymbol{R}, \boldsymbol{Y} \in \mathbb{R}^{mN}$ are defined as concatenations of $\boldsymbol{x}_k$, $\boldsymbol{f}_k$, $\boldsymbol{h}_k$, $\boldsymbol{\delta}_k$, $\boldsymbol{r}_k$, and $\boldsymbol{y}_k$ over all currencies. For example, for three currencies USD, EUR, and GBP,

$$\begin{aligned}
\boldsymbol{X} &= [\boldsymbol{x}_{USD}^{\mathrm{T}}\ \boldsymbol{x}_{EUR}^{\mathrm{T}}\ \boldsymbol{x}_{GBP}^{\mathrm{T}}]^{\mathrm{T}}, \\
\boldsymbol{H} &= [\boldsymbol{h}_{USD}^{\mathrm{T}}\ \boldsymbol{h}_{EUR}^{\mathrm{T}}\ \boldsymbol{h}_{GBP}^{\mathrm{T}}]^{\mathrm{T}}, \\
\boldsymbol{F} &= [\boldsymbol{f}_{USD}^{\mathrm{T}}\ \boldsymbol{f}_{EUR}^{\mathrm{T}}\ \boldsymbol{f}_{GBP}^{\mathrm{T}}]^{\mathrm{T}}, \\
\boldsymbol{\delta} &= [\boldsymbol{\delta}_{USD}^{\mathrm{T}}\ \boldsymbol{\delta}_{EUR}^{\mathrm{T}}\ \boldsymbol{\delta}_{GBP}^{\mathrm{T}}]^{\mathrm{T}}, \\
\boldsymbol{R} &= [\boldsymbol{r}_{USD}^{\mathrm{T}}\ \boldsymbol{r}_{EUR}^{\mathrm{T}}\ \boldsymbol{r}_{GBP}^{\mathrm{T}}]^{\mathrm{T}}, \text{ and} \\
\boldsymbol{Y} &= [\boldsymbol{y}_{USD}^{\mathrm{T}}\ \boldsymbol{y}_{EUR}^{\mathrm{T}}\ \boldsymbol{y}_{GBP}^{\mathrm{T}}]^{\mathrm{T}}.
\end{aligned}$$

Similarly, vectors $\boldsymbol{x}(t)$, $\boldsymbol{f}(t)$, $\boldsymbol{h}(t)$, $\boldsymbol{\delta}(t)$, $\boldsymbol{r}(t)$, $\boldsymbol{y}(k) \in \mathbb{R}^m$ are the concatenation of their respective variables at time t for all k, e.g.,

$$\begin{aligned}
\boldsymbol{x}(t) &= [x_{USD}(t)\ x_{EUR}(t)\ x_{GBP}(t)]^{\mathrm{T}}, \\
\boldsymbol{f}(t) &= [f_{USD}(t)\ f_{EUR}(t)\ f_{GBP}(t)]^{\mathrm{T}}, \\
\boldsymbol{h}(t) &= [h_{USD}(t)\ h_{EUR}(t)\ h_{GBP}(t)]^{\mathrm{T}}, \\
\boldsymbol{\delta}(t) &= [\delta_{USD}(t)\ \delta_{EUR}(t)\ \delta_{GBP}(t)]^{\mathrm{T}}, \\
\boldsymbol{r}(t) &= [r_{USD}(t)\ r_{EUR}(t)\ r_{GBP}(t)]^{\mathrm{T}}, \text{ and} \\
\boldsymbol{y}(t) &= [y_{USD}(t)\ y_{EUR}(t)\ y_{GBP}(t)]^{\mathrm{T}}.
\end{aligned}$$

Notice that all vectors except the optimisation objectives $\boldsymbol{H}$, $\boldsymbol{h}(t)$, and $\boldsymbol{h}_k$, are stochastic processes.

3.4 Rule-based Hedging Strategies

Forecasting or even modelling $f_k(t)$, $r_k(t)$, and $\delta_k(t)$ can be a challenging task. In this section, the transaction cost and the risk (i.e., (3.3) and (3.9)) are analysed to offer insight regarding hedging in absence of any predictive models. These analyses are used to create rule-based hedging strategies, which will serve as benchmarks.

3.4.1 Minimum Risk Strategy

Obtaining the minimum risk is possible by minimising (3.9),

$$\min \mathbb{V}\text{ar}[\sum_{i=1}^{N} x_k(i) r_k(i)],$$

which is satisfied by $x_k(t) = 0$, or

$$h_k(t) = -f_k(t).$$

In this method, every position opened through the trade with a client is closed immediately, and consequently no position is exposed to risk. This strategy, however, entails the highest cost as a transaction is performed for every trade with the clients. Furthermore, if $|f_k| > h_{k,max}$, the constraint on hedging size (3.6) is not met.

3.4.2 Minimum Cost Strategy

Minimising transaction cost is possible by minimising (3.3),

$$\min \sum_{i=1}^{N} \delta_k(i) h_k(i)^2,$$

which is satisfied by

$$h_k(t) = 0.$$

In this method, no hedging is performed, and therefore no transaction cost is incurred. However, this is impractical as this assumes $N \to \infty$ to avoid the end-of-session position closing costs of (3.2), and (3.8) is not satisfied if $|\sum f_k(i)| > x_{k,max}$.

3.4.3 Limited Position Strategy

This strategy sets a hard limit x_{max} for the dealer's positions:

$$h_k(t) = \begin{cases} x_{max} - (x_k(t) + f_k(t)) & x_k(t) + f_k(t) > x_{max} \\ -x_{max} - (x_k(t) + f_k(t)) & x_k(t) + f_k(t) < -x_{max} \\ 0 & \text{otherwise} \end{cases} \tag{3.15}$$

As this limit changes, the strategy sweeps the risk-cost profiles from minimum risk ($x_{max} = 0$, i.e. holding no risk, as in Section 3.4.1) to minimum cost ($x_{max} \to \infty$, i.e., letting the incoming client buy and sell orders neutralise each other as in Section 3.4.2).

3.4.4 Gradual Closing Strategy

Gradual closing strategy distributes the hedging actions required to close the current positions over the remaining trading time-steps. This is further parameterised by $0 \leq \lambda \leq 1$ as a risk aversion parameter, resulting in actions determined by

$$h_k(t) = -(x_k(t) + f_k(t)) \frac{(N-t)\lambda + 1}{N - t + 1}$$

where $\lambda = 1$ minimises risk by instantly closing all positions, and $\lambda = 0$ results in the positions being closed gradually with equally sized hedging actions, and therefore minimising cost by avoiding market impact.

3.4.5 Advantages and Disadvantages

Despite their ease of implementation, there are major disadvantages to the rule-based non-predictive strategies:

- No information regarding client flow, volatility, or market impact is used.
- Constraints on h and x ((3.6) and (3.8)) are not upheld.

As a result, not only the hedging results are far from optimal, but also the dealer could be left with a large open position at the end of trading session, imposing high risk or additional transaction costs.

3.5 Single-stage Hedging Strategy

A more accurate way of obtaining a hedging action is to perform a single-stage optimisation. In this strategy, (3.10) is solved with $N = 1$, which results in a single hedging action. This action is then amortised over the available steps. At the next time-step, the model parameters are updated using the observed market information and the optimisation is repeated. This strategy is effectively a simplified form of shrinking horizon control (SHC), as explained in Section 2.5 and presented in Figure 2.2b.

The optimisation objective of this strategy can be expressed formally as

$$\begin{aligned} \underset{h_k(0);\ \forall k}{\text{argmin}} \quad & \mathbb{E}\left[\sum_{k=1}^{m}\left(f_{cost}^{(N)}(h_k(0)) + f_{cost}(h_k(1))\right)\right] + \lambda \mathbb{V}\text{ar}\left[\sum_{k=1}^{m} x_k(1) r_k(1)\right] \\ \text{subject to} \quad & x_k(1) = x_k(0) + h_k(0) \\ & h_k(1) = -x_k(1) - f_k(1) \\ & |h_k(t)| \leq h_{k,max}, \quad t \in [0,1] \\ & |x_k(1)| \leq x_{k,max} \end{aligned} \tag{3.16}$$

where $f_{cost}^{(N)}$ is the cost function amortised over N payments:

$$f_{cost}^{(N)}(h) = N f_{cost}(\frac{1}{N}h) = \frac{\delta h^2}{N} = \frac{1}{N} f_{cost}(h)$$

Eq. (3.16) assumes:

- The dealer starts with position $x(0)$.
- There is no client flow at $t = 0$.
- The dealer hedges at time $t = 0$ to prepare for future flow.
- The market price changes at $t = 1$, creating a risk for positions held at that time.
- The only client flow occurs $t = 1$.
- The dealer closes all positions before $t = 2$ using $h_k(1)$.
- Market impact coefficient is constant, i.e., $\delta_k(1) = \delta_k(0) = \delta_k$.

This sequence is illustrated in Figure 3.2.

Minimising (3.16) requires $f_k(1)$, $r_k(1)$, and δ_k and yields $h_k(0), \forall k$. Using the matrix notation introduced in Section 3.3.4, we denote these values for the whole currency portfolio as vectors $\boldsymbol{f}$, $\boldsymbol{r}$, $\boldsymbol{\delta}$, and $\boldsymbol{h}$ respectively.

$\boldsymbol{h}(1)$ can be computed recursively as

$$\boldsymbol{h}(1) = -\boldsymbol{x}(0) - \boldsymbol{h} - \boldsymbol{f}.$$

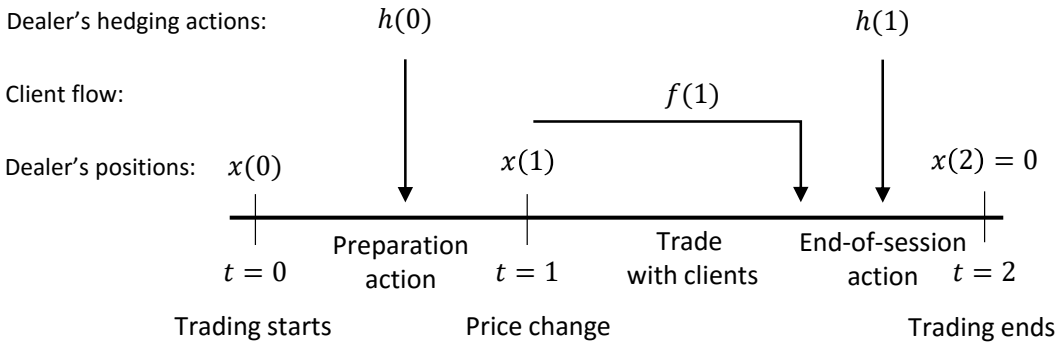


FIGURE 3.2: Single-stage hedging state-space dynamics. The dealer prepares for incoming client flow by hedging action $h(0)$, absorbs the incoming client flow $f(1)$, and closes all position using $h(1)$. Only one change of price is observed, at $t = 1$, which causes risk to the dealer's position..

Consequently, the cost function of (3.16) is reorganised as

$$\begin{aligned} J =& \mathbb{E}\left[\frac{1}{N}\boldsymbol{h}^{\mathrm{T}}\boldsymbol{\Delta}\boldsymbol{h} + (-\boldsymbol{h} - \boldsymbol{f} - \boldsymbol{x}(0))^{\mathrm{T}}\boldsymbol{\Delta}(-\boldsymbol{h} - \boldsymbol{f} - \boldsymbol{x}(0))\right] \\ &+ \lambda\mathbb{V}\mathrm{ar}\left[(\boldsymbol{h} + \boldsymbol{x}(0))^{\mathrm{T}}\boldsymbol{r}\right], \end{aligned}$$

where $\boldsymbol{\Delta}_{m\times m} = \mathrm{diag}(\boldsymbol{\delta})$ is the diagonal market impact matrix, and $\mathrm{diag}(\boldsymbol{x})$ creates a diagonal matrix from vector $\boldsymbol{x}$.

Expanding this cost function, replacing $\boldsymbol{f} + \boldsymbol{x}(0)$ with $\boldsymbol{y}$ as defined in (3.11), and sorting by the order of $\boldsymbol{h}$, yields

$$\begin{aligned} J =& \mathbb{E}\left[\frac{N+1}{N}\boldsymbol{h}^{\mathrm{T}}\boldsymbol{\Delta}\boldsymbol{h} + 2\boldsymbol{h}^{\mathrm{T}}\boldsymbol{\Delta}\boldsymbol{y} + \boldsymbol{y}^{\mathrm{T}}\boldsymbol{\Delta}\boldsymbol{y}\right] + \\ & \lambda(\boldsymbol{h} + \boldsymbol{x}(0))^{\mathrm{T}}\mathbb{V}\mathrm{ar}\left[\boldsymbol{r}\right](\boldsymbol{h} + \boldsymbol{x}(0)) \\ =& \frac{N+1}{N}\boldsymbol{h}^{\mathrm{T}}E\left[\boldsymbol{\Delta}\right]\boldsymbol{h} + 2\boldsymbol{h}^{\mathrm{T}}\mathbb{E}\left[\boldsymbol{\Delta}\boldsymbol{y}\right] + \mathbb{E}\left[\boldsymbol{y}^{\mathrm{T}}\boldsymbol{\Delta}\boldsymbol{y}\right] + \\ & \lambda\left(\boldsymbol{h}^{\mathrm{T}}\mathbb{V}\mathrm{ar}[\boldsymbol{r}]\boldsymbol{h} + 2\boldsymbol{h}^{\mathrm{T}}\mathbb{V}\mathrm{ar}[\boldsymbol{r}]\boldsymbol{x}(0) + \boldsymbol{x}(0)^{\mathrm{T}}\mathbb{V}\mathrm{ar}[\boldsymbol{r}]\boldsymbol{x}(0)\right) \\ =& \boldsymbol{h}^{\mathrm{T}}\left(\frac{N+1}{N}E\left[\boldsymbol{\Delta}\right] + \lambda\mathbb{V}\mathrm{ar}[\boldsymbol{r}]\right)\boldsymbol{h} + \\ & 2\boldsymbol{h}^{\mathrm{T}}\left(\left(\mathbb{E}\left[\boldsymbol{\Delta}\right]\mathbb{E}\left[\boldsymbol{y}\right] + \mathbb{C}\mathrm{ov}\left[\boldsymbol{\Delta}, \boldsymbol{y}\right]\right) + \lambda\mathbb{V}\mathrm{ar}[\boldsymbol{r}]\boldsymbol{x}(0)\right) + \\ & \mathbb{E}\left[\boldsymbol{y}^{\mathrm{T}}\boldsymbol{\Delta}\boldsymbol{y}\right] + \boldsymbol{x}(0)^{\mathrm{T}}\mathbb{V}\mathrm{ar}[\boldsymbol{r}]\boldsymbol{x}(0). \end{aligned}$$

Let $\bar{y} = \mathbb{E}\left[\boldsymbol{y}\right]$ be the expected client flow, $\bar{\boldsymbol{\Delta}} = \mathbb{E}\left[\boldsymbol{\Delta}\right]$ be the expected market impact matrix, and $\boldsymbol{\Sigma} = \mathbb{V}\mathrm{ar}[\boldsymbol{r}]$ be the market rate volatility covariance matrix. The terms that do not contain $\boldsymbol{h}$ can be removed, as they do not influence the optimisation. This

simplifies J to

$$\begin{aligned} J =& \boldsymbol{h}^{\mathrm{T}} \left(\frac{N+1}{N} \bar{\boldsymbol{\Delta}} + \lambda \boldsymbol{\Sigma} \right) \boldsymbol{h} + \\ & 2\boldsymbol{h}^{\mathrm{T}} \left(\bar{\boldsymbol{\Delta}} \bar{\boldsymbol{y}} + \mathbb{C}\mathrm{ov}\left[\boldsymbol{\Delta}, \boldsymbol{y} \right] + \lambda \boldsymbol{\Sigma} \boldsymbol{x}(0) \right). \end{aligned}$$

If the client flow, and consequently $\boldsymbol{y}$, are not correlated with the market (e.g., $\boldsymbol{\Delta}$ and $\boldsymbol{r}$), this cost function can be simplified further:

$$\begin{aligned} J =& \boldsymbol{h}^{\mathrm{T}} \left(\frac{N+1}{N} \bar{\boldsymbol{\Delta}} + \lambda \boldsymbol{\Sigma} \right) \boldsymbol{h} + \\ & 2\boldsymbol{h}^{\mathrm{T}} \left(\bar{\boldsymbol{\Delta}} \bar{\boldsymbol{y}} + \lambda \boldsymbol{\Sigma} \boldsymbol{x}(0) \right) \end{aligned}$$

The cost function is quadratic. Without constraints, it can be solved analytically with

$$\boldsymbol{h} = - \left(\frac{N+1}{N} \bar{\boldsymbol{\Delta}} + \lambda \boldsymbol{\Sigma} \right)^{\dagger} \left(\bar{\boldsymbol{\Delta}} \bar{\boldsymbol{y}} + \lambda \boldsymbol{\Sigma} \boldsymbol{x}(0) \right),$$

where $\dagger$ is the Moore–Penrose pseudoinverse operator. If constraints are present, quadratic programming (QP) techniques can be used to efficiently minimise this cost function.

Assuming no constraints, the results can be interpreted as follows:

- For $\lambda = 0$ (i.e., maximum risk preference), the minimisation results in $\boldsymbol{h} = -\frac{N}{N+1} \bar{\boldsymbol{y}}$, which is then amortised to $\frac{-1}{N+1} \bar{\boldsymbol{y}}$ per time-step. This can be interpreted as dividing the expected client flow equally into N hedging trades and one closing trade, which results in the minimum transaction costs.
- For $\lambda \to \infty$ (i.e., maximum risk aversion), the minimisation results in $\boldsymbol{h} = -\boldsymbol{x}(0)$. Here, the optimiser tries to close the current open positions before the change of market rate at $t = 1$ to avoid any risk.
- For $0 < \lambda < \infty$, an intermediate value for $\boldsymbol{h}$ is obtained that balances between the transaction cost (from $\boldsymbol{\Delta}$), and the risk (from $\boldsymbol{\Sigma}$).

For $\lambda \to \infty$, the hedging action is amortised to $\boldsymbol{h} = \frac{-\boldsymbol{x}(0)}{N}$, which does not yield the desired minimum cost. To neutralise the amortisation, and yield a result similar to other strategies in obtaining the *global minimum risk*, we replace the original $\boldsymbol{x}(0)$ with $N\boldsymbol{x}(0)$, which results in the following cost function:

$$J = \boldsymbol{h}^{\mathrm{T}} \left(\frac{N+1}{N} \bar{\boldsymbol{\Delta}} + \lambda \boldsymbol{\Sigma} \right) \boldsymbol{h} + 2\boldsymbol{h}^{\mathrm{T}} \left(\bar{\boldsymbol{\Delta}} \bar{\boldsymbol{y}} + \lambda N \boldsymbol{\Sigma} \boldsymbol{x}(0) \right)$$

3.5.1 Advantages and Disadvantages

This strategy has the advantage of requiring only one forecast for each variable:

- $\bar{\boldsymbol{\Delta}}$: the average market impact coefficients.
- $\bar{\boldsymbol{y}}$: the expected total client flow. As the initial position $\boldsymbol{x}(0)$ is already observed, using (3.11), this equals to forecasting $\boldsymbol{f}$.
- $\boldsymbol{\Sigma}$: the volatility covariance matrix.

However, the obtained results are not optimal. The cost function distributes the hedging actions evenly, and any flow asymmetries during the day and market events are disregarded. For example, when $\lambda \to \infty$, only the initial positions $\boldsymbol{x}(0)$ are hedged, and the risk introduced by client trades is neglected.

The same problem affects the constraint on position limits (3.8), as only the last positions are considered in the QP optimisation, and intermediate positions resulting from the amortisation process are not considered.

3.6 Dynamic Risk Management using SMPC

Stochastic model predictive control (SMPC) is a powerful process control methodology, which is applicable to stochastic processes where a state-space model is defined to predict the system's behaviour. The dealer model (3.1) is effectively a state-space representation, with the uncontrollable inputs modelled using stochastic variables. Consequently, this problem can be efficiently solved using stochastic control theory.

3.6.1 Optimisation Model

An optimal solution to the hedging problem is given by solving (3.10) for every available hedging time-step:

$$\begin{aligned} \underset{h_k(t);\ \forall t,k}{\text{argmin}} \quad & \mathbb{E}\left[\sum_{k=1}^{m}\sum_{t=0}^{N} f_{cost}(h_k(t))\right] + \lambda \mathbb{V}\text{ar}\left[\sum_{k=1}^{m}\sum_{t=1}^{N} x_k(t) r_k(t)\right] \\ \text{subject to} \quad & x_k(t+1) = x_k(t) + h_k(t) + f_k(t) \\ & x_k(N+1) = 0 \\ & |h_k(t)| \leq h_{k,max} \\ & |x_k(t)| \leq x_{k,max} \end{aligned} \tag{3.17}$$

Replacing the transaction cost function in (3.17) with (3.5), applying the end of trade closing constraint (3.2), and expanding $x_k(t)$ recursively using (3.1) results in the following cost function:

$$\begin{aligned} J =& \mathbb{E}\left[\sum_{k=1}^{m}\sum_{t=0}^{N-1}\delta_k(t)h_k(t)^2 + \sum_{k=1}^{m}\delta_k(N)\left(\sum_{i=0}^{N-1}h_k(i) + \sum_{i=0}^{N}f_k(i) + x_k(0)\right)^2\right] + \\ & \lambda\mathbb{V}\mathrm{ar}\left[\sum_{k=1}^{m}\sum_{t=1}^{N}\left(\sum_{i=0}^{t-1}h_k(i) + \sum_{i=0}^{t-1}f_k(i) + x_k(0)\right)r_k(t)\right] \end{aligned}$$

Substituting $y_k(t)$ from (3.11) simplifies the above to

$$\begin{aligned} J =& \mathbb{E}\left[\sum_{k=1}^{m}\sum_{t=0}^{N-1}\delta_k(t)h_k(t)^2 + \sum_{k=1}^{m}\delta_k(N)\left(\sum_{i=0}^{N-1}h_k(i) + y_k(N+1)\right)^2\right] + \\ & \lambda\mathbb{V}\mathrm{ar}\left[\sum_{k=1}^{m}\sum_{t=1}^{N}\left(\sum_{i=0}^{t-1}h_k(i) + y_k(t)\right)r_k(t)\right]. \end{aligned} \tag{3.18}$$

Using the notation of Section 3.3.4, (3.18) can be written in matrix form as

$$\begin{aligned} J =& \mathbb{E}\Big[\boldsymbol{H}^{\mathrm{T}}\boldsymbol{\Delta}\boldsymbol{H} + (\boldsymbol{\Upsilon}^{\mathrm{T}}\boldsymbol{H} + \boldsymbol{y}(N+1))^{\mathrm{T}}\mathbf{D}(\boldsymbol{\Upsilon}^{\mathrm{T}}\boldsymbol{H} + \boldsymbol{y}(N+1))\Big] + \\ & \lambda\mathbb{V}\mathrm{ar}\left[(\boldsymbol{\Psi}\boldsymbol{H} + \boldsymbol{Y})^{\mathrm{T}}\boldsymbol{R}\right], \end{aligned} \tag{3.19}$$

where

$$\boldsymbol{\Psi}_{mN\times mN} = \mathbf{I}_{m\times m}\otimes\mathbf{S}_{N\times N} = \begin{bmatrix} \mathbf{S} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{S} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{S} \end{bmatrix},$$

$$\begin{aligned} \boldsymbol{\Upsilon}_{mN\times m} &= \mathbf{I}_{m\times m}\otimes\vec{\mathbf{1}}_{N\times 1}, \\ \mathbf{D}_{m\times m} &= \mathrm{diag}(\boldsymbol{\delta}(N)), \\ \boldsymbol{\Delta}_{mN\times mN} &= \mathrm{diag}(\boldsymbol{\delta}), \end{aligned}$$

$\mathbf{S}$ is a lower triangular matrix of 1's, $\vec{\mathbf{1}}_{N\times 1}$ is $[1\ 1\ \ldots\ 1]_{N\times 1}$, $\mathbf{I}$ is the identity matrix, $\mathrm{diag}(\boldsymbol{x})$ creates a diagonal matrix from vector $\boldsymbol{x}$, and $\otimes$ is the Kronecker product.

Additionally, matrix form of (3.11) is

$$\boldsymbol{Y} = \boldsymbol{\Psi}\boldsymbol{F} + \boldsymbol{\Upsilon}\boldsymbol{x}(0).$$

Stochastic operators in (3.19) can be expanded,

$$\begin{aligned} J =& \boldsymbol{H}^{\mathrm{T}}\mathbb{E}[\boldsymbol{\Delta}]\boldsymbol{H} + \boldsymbol{H}^{\mathrm{T}}\boldsymbol{\Upsilon}\mathbb{E}[\mathbf{D}]\boldsymbol{\Upsilon}^{\mathrm{T}}\boldsymbol{H} + \\ & 2\boldsymbol{H}^{\mathrm{T}}\boldsymbol{\Upsilon}\mathbb{E}[\mathbf{D}\boldsymbol{y}(N+1)] + \mathbb{E}[\boldsymbol{y}(N+1)^{\mathrm{T}}\mathbf{D}\boldsymbol{y}(N+1)] + \\ & \lambda\left(\mathbb{V}\mathrm{ar}[\boldsymbol{Y}^{\mathrm{T}}\boldsymbol{R}] + \boldsymbol{H}^{\mathrm{T}}\boldsymbol{\Psi}^{\mathrm{T}}\mathbb{V}\mathrm{ar}[\boldsymbol{R}]\boldsymbol{\Psi}\boldsymbol{H} + 2\boldsymbol{H}^{\mathrm{T}}\boldsymbol{\Psi}^{\mathrm{T}}\mathbb{C}\mathrm{ov}[\boldsymbol{R}, \boldsymbol{Y}^{\mathrm{T}}\boldsymbol{R}]\right), \end{aligned}$$

and reordered by $\boldsymbol{H}$, resulting in a quadratic cost function:

$$\begin{aligned} J =& \boldsymbol{H}^{\mathrm{T}}\left(\mathbb{E}[\boldsymbol{\Delta}] + \boldsymbol{\Upsilon}\mathbb{E}[\mathbf{D}]\boldsymbol{\Upsilon}^{\mathrm{T}} + \lambda\boldsymbol{\Psi}^{\mathrm{T}}\mathbb{V}\mathrm{ar}[\boldsymbol{R}]\boldsymbol{\Psi}\right)\boldsymbol{H} + \\ & 2\boldsymbol{H}^{\mathrm{T}}\left(\boldsymbol{\Upsilon}\mathbb{E}[\mathbf{D}\boldsymbol{y}(N+1)] + \lambda\boldsymbol{\Psi}^{\mathrm{T}}\mathbb{C}\mathrm{ov}[\boldsymbol{R}, \boldsymbol{Y}^{\mathrm{T}}\boldsymbol{R}]\right) + \\ & \mathbb{E}[\boldsymbol{y}(N+1)^{\mathrm{T}}\mathbf{D}\boldsymbol{y}(N+1)] + \lambda\mathbb{V}\mathrm{ar}[\boldsymbol{Y}^{\mathrm{T}}\boldsymbol{R}] \end{aligned} \tag{3.20}$$

3.6.2 Hedging Solution and Constraints

Unconstrained MPC problems with quadratic cost and linear state-space are referred to as linear-quadratic (LQ) control problems and their solution can be found using the matrix algebraic Riccati equation [8]. With deterministic constraints, (3.20) can be solved efficiently using numeric quadratic programming techniques.

The problem formulation in (3.17) includes four constraints. The first two are equality constraints and were implicitly implemented as a part of the cost function in Section 3.6.1. The third constraint, $|h_k(i)| \leq h_{k,max}$, is deterministic and easily described by

$$\begin{cases} -\boldsymbol{H} \leq \boldsymbol{H}_{max} \\ \boldsymbol{H} \leq \boldsymbol{H}_{max} \end{cases}.$$

The fourth constraint requires special attention, as $x_k(i), i \in [1, \cdots, N]$ are stochastic variables, and therefore are not deterministically constrainable before being observed. However, in a real-world multi-stage optimisation, one can use $\boldsymbol{f}(0)$ and $\boldsymbol{x}(0)$, which are known before solving for $\boldsymbol{h}(0)$ (as shown in Figure 3.1), to make the first inequality, $\boldsymbol{x}(1) < \boldsymbol{x}_{max}$, deterministic. The rest of the inequalities can be replaced with their expected values [77], i.e., $|\mathbb{E}[x_k(i)]| \leq x_{k,max}$. While this results in a less conservative control law, it still guarantees constraints fulfilment [131].

Other approaches are also possible, including imposing constraints to hold with a given sufficiently high probability, or using the worst case of a scenario tree. Both of these approaches may not hold for some situations (e.g., in the case of scenario trees, when the outcome is not covered by the generated scenarios), and suffer from higher computational complexity.

In this chapter, the expected value constraint is implemented. Rewritten in matrix form, and given $\mathbb{E}[\boldsymbol{X}] = \mathbb{E}[\boldsymbol{\Psi H} + \boldsymbol{Y}]$, the following constraints are imposed:

$$-\boldsymbol{\Psi H} \leq \boldsymbol{X}_{max} + \mathbb{E}[\boldsymbol{Y}]$$
$$\boldsymbol{\Psi H} \leq \mathbf{X}_{max} - \mathbb{E}[\boldsymbol{Y}]$$

3.6.3 Prediction Requirements

Solving (3.20) for $\mathbf{H}$ requires the following parameters:

- $\mathbb{E}[\boldsymbol{\Delta}]$ and $\mathbb{E}[\mathbf{D}]$: the expected value of market impact coefficients
- $\mathbb{V}\text{ar}[\mathbf{R}]$: FX rate returns covariance matrix
- $\mathbb{E}[\mathbf{D}\boldsymbol{y}(N+1)]$ and $\mathbb{E}[\boldsymbol{y}(N+1)^{\mathrm{T}}\mathbf{D}\boldsymbol{y}(N+1)]$: the expected value of transaction costs for closing the accumulated client flow
- $\mathbb{V}\text{ar}[\boldsymbol{Y}^{\mathrm{T}}\boldsymbol{R}]$: the variance of P&L in absence of hedging
- $\mathbb{C}\text{ov}[\boldsymbol{R}, \boldsymbol{Y}^{\mathrm{T}}\boldsymbol{R}]$: the covariance matrix of FX returns and P&L

Different approaches are available for forecasting these parameters:

1. Using an analytical model for each parameter.
2. Treating each parameter as a multi-dimensional time-series, and forecasting them using a statistical or ML technique.
3. Obtaining values using Monte Carlo techniques: stochastic models are defined for $\boldsymbol{F}$, $\boldsymbol{R}$ and $\boldsymbol{\delta}$, and fit to the observation from the market. η random scenarios are generated using these models, and the expected values and covariance matrices are approximated from the scenarios.

Each method has its own advantages or disadvantages:

1. Simple analytical models are easy to explain and solve analytically. However, better modelling of real-world events requires more sophisticated models, adding to the complexity of any analytical solution.
2. Forecasting techniques are well studied. However, six different matrices and vectors need to be forecast, requiring one model per each matrix element, which adds to the model tuning and computational complexity.
3. The Monte Carlo methods require a large number of scenarios to improve their approximations, and are computationally expensive. Nevertheless, Monte Carlo methods are not bound to analytical models for generating scenarios, and are able to directly sample historical data [67].

3.6.4 Simplification of the Hedging Model

If the client flow is uncorrelated to the market conditions (i.e., the FX rate returns, volatility and market impact coefficient), (3.20) can be further simplified to

$$\begin{aligned} J = &\boldsymbol{H}^{\mathrm{T}}\left(\bar{\boldsymbol{\Delta}} + \boldsymbol{\Upsilon}\bar{\mathbf{D}}\boldsymbol{\Upsilon}^{\mathrm{T}} + \lambda\boldsymbol{\Psi}^{\mathrm{T}}\boldsymbol{\Sigma}\boldsymbol{\Psi}\right)\boldsymbol{H} + \\ &2\boldsymbol{H}^{\mathrm{T}}\left(\boldsymbol{\Upsilon}\bar{\mathbf{D}}\bar{\boldsymbol{y}}(N+1) + \lambda\boldsymbol{\Psi}^{\mathrm{T}}\boldsymbol{\Sigma}\bar{\boldsymbol{Y}}\right) \end{aligned} \tag{3.22}$$

where $\boldsymbol{\Sigma} = \mathbb{V}\mathrm{ar}[\boldsymbol{R}]$ is the returns covariance matrix and for every variable the expected value $\mathbb{E}[x]$ is denoted by $\bar{x}$. Notice that the constant terms are also dropped.

Therefore, the only requirements are

- expected position of the dealer in absence of hedging $\bar{\boldsymbol{Y}} = \mathbb{E}[\boldsymbol{Y}]$, or the expected client flow $\bar{\boldsymbol{F}} = \mathbb{E}[\boldsymbol{F}]$ as $\mathbb{E}[\boldsymbol{Y}] = \boldsymbol{\Psi}\mathbb{E}[\boldsymbol{F}] + \boldsymbol{x}(0)$,
- expected market impact coefficient $\bar{\boldsymbol{\delta}} = \mathbb{E}[\boldsymbol{\delta}]$, which is then formed into diagonal matrices $\bar{\boldsymbol{\Delta}}$ and $\bar{\mathbf{D}}$, and
- volatility covariance matrix $\boldsymbol{\Sigma}$.

3.7 The Risk Management System Architecture

An FX risk management system can be implemented using the SMPC hedging model proposed in Section 3.6.1. The internal design of this system is divided to three major components:

- A SMPC controller, optimising the cost function defined in (3.20).
- A customisable time-series prediction engine, forecasting parameters required by the SMPC controller. Implementing a single customisable time-series forecasting engine, instead of multiple task specific models, allows system scalability: manually tuning a task specific model is time-consuming and costly. Furthermore, the customisable engine can automatically discover the best model for prediction and dynamically adapt to changes. Details of designing this subsystem will be discussed in Chapter 5.
- A hedging action manager, which queries the SMPC controller and executes the recommended hedging action.

Figure 3.3 illustrates the relationship of the proposed risk management system and its components to the dealer and the FX market.

While in a normal system the SMPC controller is responsible for predictive modelling and executing the results as well as the optimisation, in our proposed risk management system these tasks are assigned into different modules. Additionally, the hedging action

management subsystem is arranged as a part of the dealer. This modular layout has the following advantages compared to a monolithic design:

- External circuit breaker: Due to regulatory issues, automated financial systems may require an external circuit breaker to monitor their behaviour, and issue a trading halt if an unexpected decision is made. A modular design allows external audits to be included without affecting the SMPC controller and the prediction engine.
- Fault tolerance: Financial systems have to be highly reliable. In the proposed system, fault tolerance is achievable through N-module redundancy, where multiple identical SMPC controllers and prediction engine instances are run on different computers. The action manager queries all instances and selects the correct results on the basis of a quorum, and thus discarding results from hardware failures or errors.
- High performance computing: A modular design allows different subsystems to be executed on separate processors. For example, instead of serially running the prediction engine for each parameter, several engines are executed in parallel on different processors, tuning unrelated time-series prediction models separately.

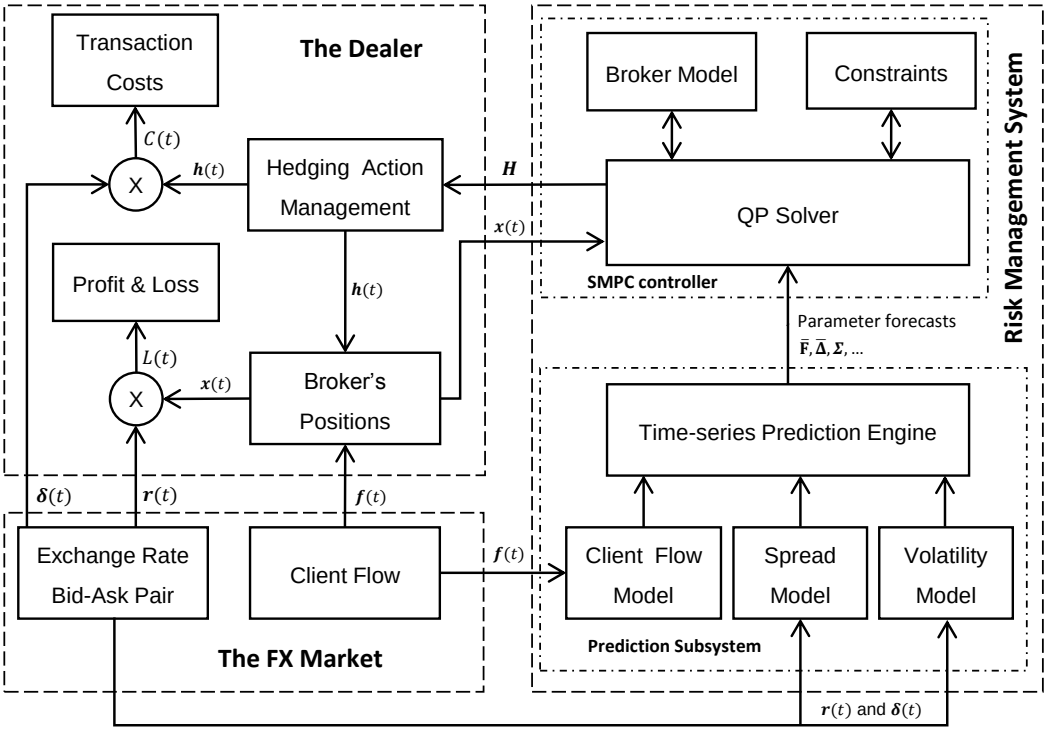


FIGURE 3.3: The proposed FX risk management system and its components.

3.7.1 Shrinking Horizon Hedging

In real-world systems, exact modelling of the client flow and the FX market is impossible due to incomplete data and approximations, and consequently the accuracy of hedging actions is lost over time. To improve hedging performance, one can include new information in the optimiser using a shrinking horizon scheme, as summarised in Algorithm 3.1:

1. At each time-step the prediction subsystem updates the client flow, FX rate returns, and market impact coefficient models, based on the observed market conditions.
2. The prediction subsystem applies statistical or machine learning techniques to forecast the future client flow and market conditions, and estimate the cost function parameters accordingly.
3. The estimated parameters are then given to the SMPC controller, which uses a quadratic programming (QP) solver to obtain the optimal hedging action $\boldsymbol{H}$ by minimising (3.20), subject to the constraints of Section 3.6.2.
4. The hedging action management system hedges the positions by $\boldsymbol{h}(0)$, which adds to the transaction costs.
5. At the next time-step, FX volatility may change the FX rates and consequently the value of the dealer's open positions, and thus generating profit or loss.
6. The hedging algorithm is repeated from the first step, with the horizon shortened one time-step.
7. The algorithm terminates when the end-of-trading time is reached.

Algorithm 3.1: SMPC Hedging Algorithm

$t \leftarrow 0$
while $t \leq N$ **do**
  Update `client flow models` using new trades data.
  Update `FX rate volatility models` from market conditions.
  Update `market impact coefficient models` from market conditions.
  Predict the hedging model's parameters, $\mathbb{E}[\mathbf{F}]$, $\mathbb{E}[\boldsymbol{\delta}]$, $\mathbb{V}\mathrm{ar}[\mathbf{R}]$, $\cdots$.
  Compute $\boldsymbol{H} = [\boldsymbol{h}(t), \boldsymbol{h}(t+1), \ldots, \boldsymbol{h}(N)]$ by minimising (3.20).
  Hedge by $\boldsymbol{h}(t)$.
  Update positions $\boldsymbol{x}(t)$.
  $t \leftarrow t + 1$.
end
Close all position using $\boldsymbol{h}(N) = -(\boldsymbol{x}(N) + \boldsymbol{f}(N))$.

3.8 Summary

In this chapter, FX risk hedging from a dealer's point of view was studied. Realistic assumptions were made to model the dealer dynamics, considering the stochastic nature of client flow, FX rate, and transaction cost. Rule-based hedging approaches were then studied, and disadvantages of each were noted.

The main contribution of this chapter is a SMPC approach to FX hedging. A finite horizon optimisation was formulated using the proposed dealer model, and an algorithm was devised to periodically update the model using feedback. The model's cost function was constructed such that it can be efficiently minimised via quadratic programming, making the scheme suitable for real-time implementations.

The forward looking multi-period formulation of the proposed model uses several stochastic variables to describe future client flow and market states. The following chapters will discuss machine learning based time-series forecasting techniques to model and predict these variables.

Chapter 4

A Machine Learning Approach to Time-series Prediction

4.1 Introduction

A *time-series* is defined as a set of observations, recorded successively at specific points in time [166]. Time-series are used in any domain which involves temporal measurements, including finance and econometrics, signal processing, control engineering, the energy industry, astronomy, climate sciences, biomedical sciences, and management.

Forecasting (or prediction[1]) is an important time-series analytics tool; success or failure of many systems relies on an accurate prediction of future.

Forecasting algorithms, either as general purpose prediction tools or adapted to specific applications, have been subject to extensive research [36, 166]. In this chapter, based on the state-of-the-art techniques in literature, a generic machine learning (ML) based approach to time-series prediction is studied.

4.2 Time-series Prediction as a Regression Task

In this thesis, as with many other applications employing digital computers, time-series are studied only in a discrete-time framework: for a time-series denoted by $\{x(t)\}$, values are only sampled at equally spread intervals, i.e., the finite set $t \in [0, 1, \ldots, N]$.

From a machine learning perspective, time-series prediction is commonly realised as a regression task. The objective is to find a regression function, $f(\cdot)$, to model future

[1] Forecasting and prediction are used interchangeably within this thesis.

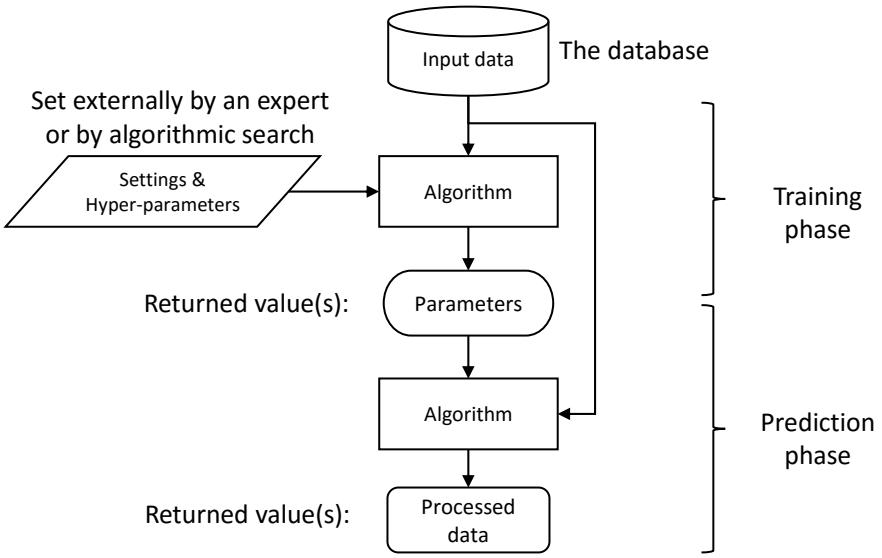


FIGURE 4.1: Inputs and outputs in a generic machine learning algorithm.

values of a time-series using its past observations. This can be expressed formally as

$$\hat{x}(t+h) = f\big(x(t), x(t-1), x(t-2), \dots\big) \tag{4.1}$$

where $h \geq 1$ is the *horizon* in future to predict, and $\hat{x}(t+h)$ is the estimate (i.e., the forecast or prediction) of x at time $t+h$, i.e., $x(t+h)$.

Time-series prediction is a *supervised* machine learning task [167]: the algorithm learns from the provided examples (in this case, historical time-series), and its results are validated against the already observed future of that time-series. In the rest of this section, details of this learning process are discussed in depth.

4.2.1 Terminology

Supervised machine learning algorithms are divided into two separate phases:

- Training phase, where available data is applied to the model to fit model parameters.
- Prediction phase, where new data is applied to the model to estimate future forecasts.

It is noteworthy to differentiate the two types of *parameters* used (Figure 4.1):

- Ordinary parameters, which are built-in arguments within the internal formulation of the mathematical model. These values are determined by the model fitting algorithm and returned as its output.
- Hyper-parameters, which are set by user, and act as inputs or settings.

Consider fitting a finite impulse response (FIR) filter, expressed as

$$x(t+1) = \sum_{i=0}^{p} \alpha_i x(t-i) + \epsilon(t) \tag{4.2}$$

to model and predict a black box system from data. Here

- $x(t)$ is the input data.
- $\epsilon(t)$ is the error, modelled as Gaussian noise.
- p, the FIR model order, is the hyper-parameter. It is set by the user, and tuned using a model validation technique.
- α_i, the coefficients, are the ordinary parameters. These coefficients are automatically determined during the model fitting process, which is commonly implemented using ordinary least squares.

After fitting the model and obtaining the α_i parameters, the same coefficients are used on any new data to obtain $x(t+1)$.

Care must be taken in implementation of training and prediction phases, as some algorithms are often mentioned without separating these two steps. For example, consider *feature scaled* normalisation, defined by

$$\bar{\boldsymbol{x}} = \frac{\boldsymbol{x} - \min(\boldsymbol{x})}{\max(\boldsymbol{x}) - \min(\boldsymbol{x})} \tag{4.3}$$

where $\boldsymbol{x}$ is a vector of data, and $\bar{\boldsymbol{x}}$ is the normalised data.

Here, the first step is to compute the maximum and minimum of the data. The second step applies these values to centre and scale the data. In real-world problems, where normalisation is itself the first phase in a multi-step processing algorithm, these parameters are extracted from the larger set of *observed data* at the training stage, stored, and later applied to a smaller set of *new data* during the prediction stage. Otherwise, if the parameters are extracted from the new data for its own scaling, considering the smaller size of the new data set, the statistical accuracy of parameters will be reduced, and thus possibly invalidating the rest of the multi-step processing algorithm's assumptions.

4.3 Data Flow for Time-series Prediction

In time-series prediction, the training phase is commonly [13, 167, 168] comprised of the following sequence (Figure 4.2):

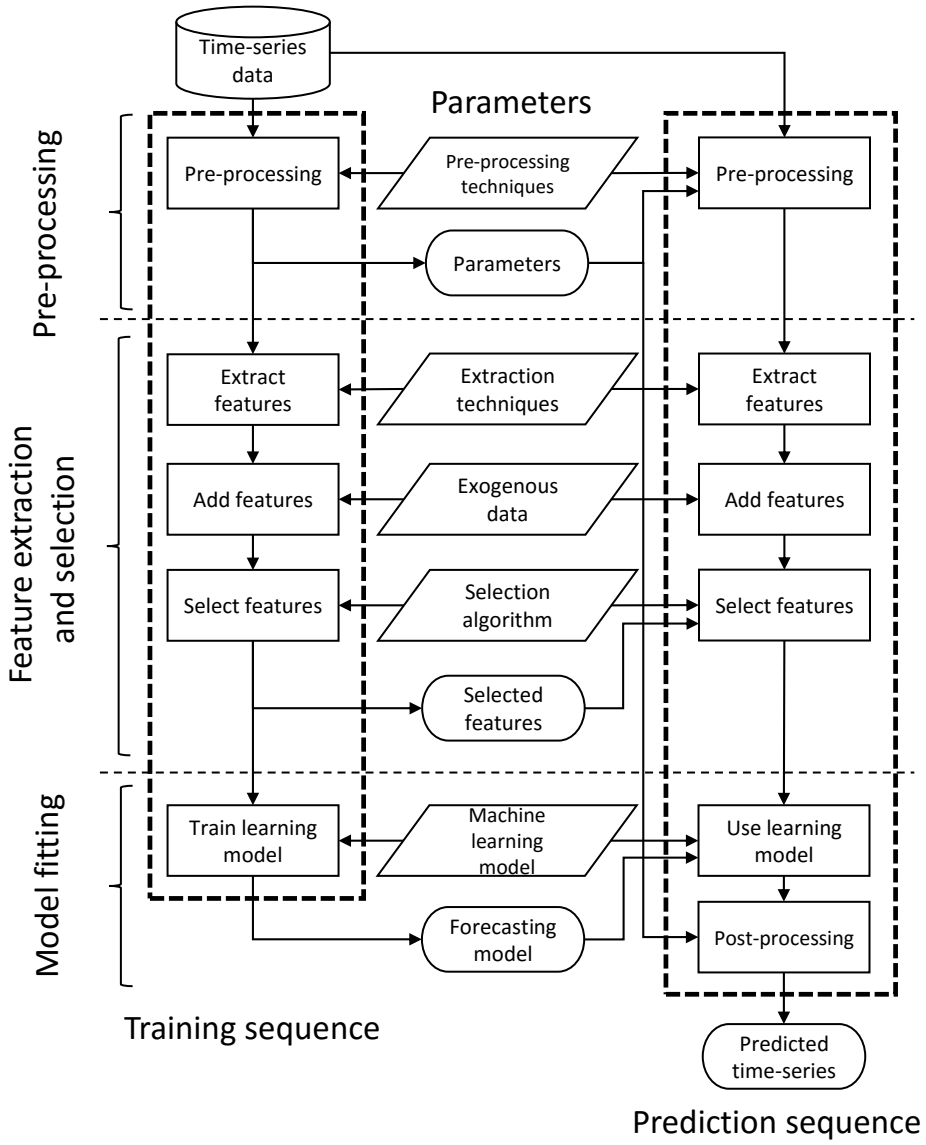


FIGURE 4.2: Data flow in machine learning based time-series prediction.

1. Pre-processing: Machine learning algorithms commonly assume a certain data distribution (e.g., zero-mean unit-variance Gaussian). Consequently, an *adjustment* step is required to adapt the statistics of the data.
2. Feature extraction: A regression function typically requires a multi-variate vector of inputs, with enough information to model and predict the output. In contrast, a time-series is a long single dimensional vector. The goal of feature extraction is to convert the time-series data to an informative vector, so as to be used as the input to the learning algorithm. This data may additionally include other time-series, which we refer to as *exogenous* data.
3. Feature selection: Redundancies may occur during the feature extraction phase, and some features may be irrelevant to the output. Hence, a feature selection step is commonly applied to only keep the informative features.

4. Model training: A machine learning algorithm is applied to the features to create a predictive model. Many machine learning models are available, ranging from linear regression using ordinary least squares (OLS), to artificial neural networks (ANN) and kernel based techniques such as support vector regression (SVR) [169] and kernel recursive least squares (KRLS) [170]. Each model has its own assumptions, advantages, and disadvantages.

The prediction sequence is similar to the training phase, as the regression model requires input features similar to what it has been trained on:

1. Pre-processing: New time-series data is pre-processed (normalised, power transformed, seasonally adjusted) using the parameters that were obtained in the training phase.
2. Feature extraction: From the pre-processed time-series (and also the other exogenous time-series), features are generated using the same techniques used during the training.
3. Feature selection: The same features deemed relevant in training phase are selected.
4. Prediction: The features are used with the regression model obtained in the training phase to predict the future values.
5. Post-processing: The prediction step is actually forecasting the *pre-processed* time-series. To forecast the original time-series, pre-processing has to be reversed.

4.3.1 Pre-processing for Time-series Prediction

Several pre-processing algorithms are available for time-series:

- Normalisation: The mean and variance of a time-series is adjusted to create a zero-mean unit-variance data.
- Scaling: An alternative to normalisation is feature scaling, implemented using (4.3) as explained in Section 4.2.1.
- Power transform: A power transform adjusts the data distribution to create a more Gaussian-like distribution by *stabilising* its variance. The simplest power transform is logarithm. More advanced and parametrisable techniques exist, such as the Box–Cox transformation [171], defined by

$$x^{(\lambda)} = \begin{cases} \frac{x^{\lambda}-1}{\lambda} & \text{if } \lambda \neq 0 \\ \log(x) & \text{if } \lambda = 0 \end{cases} \tag{4.4}$$

  where λ is the scaling parameter.

- Seasonal decomposition: Many time-series show a *seasonal* variation or a *trend*. For example, solar radiation (and consequently PV electricity generation) has a daily cycle (with peak at noon) and a yearly cycle (as the sun elevation changes with the season). Electricity consumption may show an upward trend due to continuous population growth. Separating these variations improves predictability by reducing the complexity of the models required to fit each part of the time-series. Several techniques are available, including STL (Seasonal and Trend decomposition using Loess) [172].
- Differencing: In addition to decomposition techniques, one can use *differencing*, i.e., subtracting lagged values from the current value, to remove the trends and seasonality. This effectively works by removing the common trend or seasonal element which is repeated through the time-series.
  For example, consider an AR time-series with trend and drift,

$$x(t) = c + \delta t + \alpha x(t-1) + \epsilon(t),$$

  where c is the drift, δt is the trend, and $\epsilon(t)$ is an independent and identically distributed (i.i.d) Gaussian noise.
  The first order difference is

$$x'(t) = x(t) - x(t-1) = \delta + \alpha(x(t-1) - x(t-2)) + \epsilon(t) - \epsilon(t-1)$$

  and the second order difference, $x''(t) = x'(t) - x'(t-1)$, becomes

$$\begin{aligned} x''(t) &= x(t) - 2x(t-1) + x(t-2) \\ &= \alpha(x(t-1) - 2x(t-2) + x(t-3)) + \epsilon(t) - 2\epsilon(t-1) + \epsilon(t-2). \end{aligned}$$

  Let $e(t) = \epsilon(t) - 2\epsilon(t-1) + \epsilon(t-2)$ be a Gaussian random noise (which is true by the assumption of $\epsilon(t)$ being i.i.d), and $y(t) = x(t) - 2x(t-1) + x(t-2)$. The above can be rewritten as

$$y(t) = \alpha y(t-1) + e(t)$$

  which is a first order AR model, and easily predictable using available tools.
  A common way of selecting the optimal differencing order is to test for *stationarity*. A *stationary* time-series is one "whose properties do not depend on the time at which the series is observed" [36]. In other terms, the time-series does not show any trend or seasonal behaviour. Several tests are available, mostly testing for existence of a *unit root*. If the AR model polynomial (or in other terms, the FIR filter) contains roots outside the unit circle, it shows non-stationary (i.e., unstable)

behaviour. Commonly used tests include:

- Kwiatkowski–Phillips–Schmidt–Shin test (KPSS) [173],
- Augmented Dickey-Fuller test (ADF) [174],
- Phillips-Perron test (PP) [175],
- Canova-Hansen (CH) test [176] with null hypothesis of deterministic seasonality, and
- Osborn-Chui-Smith-Birchenhall (OCSB) test [177] with null hypothesis that a seasonal unit root exists.

Tests differ on their assumptions, e.g., the null hypothesis that x has a unit root against a stationary root alternative, versus the null hypothesis that x has a stationary root against a unit root alternative. Hence, they may result in various differencing order suggestions.

4.3.2 Features for Time-series Prediction

Features are informative variables, used as inputs for a learning model. A good time-series prediction algorithm employs a combination of the feature extraction process and learning model to closely mimic the physical process that generates the time-series. As a result, features have to be designed around the assumptions about this process.

A commonly used time-series process model is an infinite impulse response (IIR) model,

$$x(t+1) = c_0 + \sum_{i=0}^{p} \alpha_i x(t-i) + \sum_{i=0}^{q} \beta_i u(t-i) + \epsilon(t),$$

where $x(t)$ is the time-series, $u(t)$ is an exogenous input to the system, c_0 is a constant, p and q are the model orders, and ϵ_t is white noise.

This assumption is the basis of many forecasting techniques, with the simplest being the auto-regressive (AR) model:

$$\hat{x}(t+1) = \sum_{i=0}^{p} \alpha_i x(t-i) \tag{4.5}$$

Here, the features are simply a *lagged window* of the time-series.

The simple AR model has been extended in order to use a non-linear regression function, and include a lagged window of another time-series as its exogenous input. This extension is referred to as the non-linear auto-regressive with exogenous input (NARX)

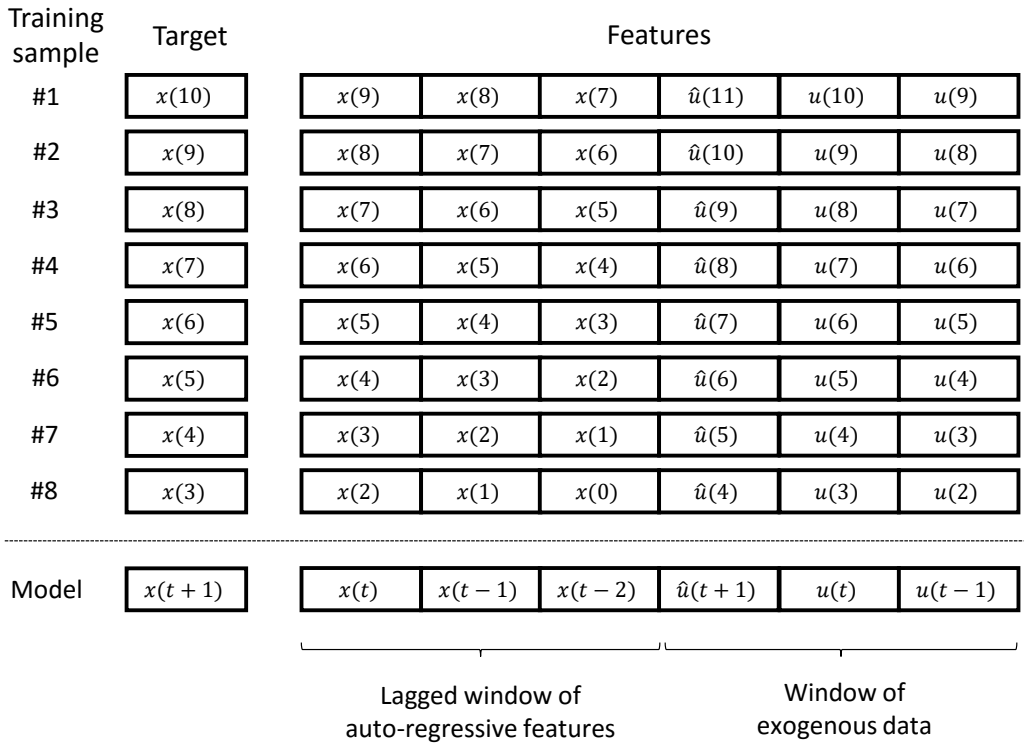


FIGURE 4.3: Example of endogenous and exogenous features for time-series prediction. Notice that $u(t+1)$ is not available for inferring $x(t)$, as it is yet to be observed at time t. As a result, its forecast $\hat{u}(t+1)$ is used instead.

model:

$$\hat{x}(t+1) = f\Big(x(t), x(t-1), \ldots, x(t-p), u(t), u(t-1), \ldots, u(t-q)\Big) \tag{4.6}$$

where $f(\cdot)$ is the non-linear regression function.

4.3.2.1 Endogenous vs. Exogenous Features

As mentioned above, features are commonly divided into two groups:

- Endogenous features, which are extracted from historical observations of the same time-series data.
- Exogenous features, which are constructed from other time-series.

Figure 4.3 illustrates an example of this type of features for time-series.

Exogenous features are usually helpful, especially if they are predicted using a more accurate technique. For example, renewable energy generation is highly correlated with weather conditions, and future weather conditions are available due to scientific understanding of atmospheric processes and the high level of investment in monitoring current

states of the atmosphere. As a result, predicting renewable energy generation benefits from using weather forecasts as exogenous features.

Unfortunately, exogenous features are not available for all time-series due to either lack of data, or lack of understanding about the processes driving the time-series.

4.3.2.2 Seasonality Features

In addition to seasonal decomposition, auto-regressive models (either in linear form or non-linear form) can be readily extended to take seasonality into account.

This is realised by including *seasonal* lags of the time-series in addition to the ordinary lags [36, 178], as in

$$\begin{aligned}\hat{x}(t+1) = f\Big(&x(t), x(t-1), \ldots, x(t-p), \\ &x(t-T), x(t-2T), \ldots, x(t-PT)\Big)\end{aligned} \tag{4.7}$$

where P is the order of the seasonal features and T is the period of seasonality.

The seasonality period is not always known beforehand. Several methods are available for estimating it, which mainly include finding the peak frequency from a periodogram plot or the Fourier transform [179].

4.3.2.3 Argument against Linearly Dependent Features

Some publications employing time-series prediction have used features such as Discrete Fourier Transform (DFT) coefficients, Wavelet coefficients, and different moving averages (simple, exponentially weighted, and custom weighted). All of these features only offer an affine transformation over the original time-series:

$$\boldsymbol{X} = \mathbf{W}\boldsymbol{x}$$

For example, kth coefficient of DFT is defined by

$$X_k \overset{\text{def}}{=} \sum_{t=0}^{N-1} x(t) \cdot e^{-2\pi i k t/N}.$$

A four point DFT matrix, i.e., $\boldsymbol{x} \in \mathbb{R}^4$, $\boldsymbol{X} \in \mathbb{C}^4$, $\mathbf{W} \in \mathbb{C}^{4\times 4}$ is realised by

$$\mathbf{W} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{bmatrix}$$

Similarly, a discrete wavelet transform (DWT) with Haar wavelets for $N = 4$ is realised by a similar affine transformation with

$$\mathbf{W} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix}.$$

Additionally, all moving averages, and other concepts such as technical indicators [180, 181] are different variations of an FIR filter:

- A simple moving average (SMA) is realised by $\alpha_i = \frac{1}{p}$ in (4.2), where p is the model order.
- Exponentially weighted average (EMA) uses $\alpha_0 = a$ and $\alpha_i = a(1-a)^i$, where $0 < a < 1$ is a constant parameter.
- A *momentum* technical indicator, defined as $momentum = x(t) - x(t-1)$, is constructed by setting $\alpha_0 = 1, \alpha_1 = -1$, and $\alpha_i = 0, i > 1$.

Considering this affine relationship, using the aforementioned transformations as features is redundant; even a linear regression function will find the optimal affine coefficients for extracting useful features from the time-series, and using another layer of affine transformation will not add any useful information. Formally, let $\boldsymbol{x}$ be the input vector and $\boldsymbol{y}$ the targets. Solving for $\mathbf{B}$ in $\boldsymbol{y} = \mathbf{B}\boldsymbol{f}$ (where $\boldsymbol{f} = \mathbf{W}\boldsymbol{x}$ is the features vector) is equivalent to finding $\mathbf{A}$ in $\boldsymbol{y} = \mathbf{A}\boldsymbol{x}$, where $\mathbf{A} = \mathbf{B}\mathbf{W}$.

Additionally, moving averages create a chance of *peeking* if incorrectly implemented. A simple moving average, defined by

$$\bar{x}(t) = \frac{1}{2n+1} \sum_{i=-n}^{n} x(t+i)$$

is using information from $x(t+1), x(t+2), \cdots$, which, while available during the training phase and simulations, are not observed in a real-world setting at time t.

TABLE 4.1: Dummy variables for weekly seasonality. Notice the absence of a seventh variable, removed due to linear dependence.

	V_1	V_2	V_3	V_4	V_5	V_6
Monday	1	0	0	0	0	0
Tuesday	0	1	0	0	0	0
Wednesday	0	0	1	0	0	0
Thursday	0	0	0	1	0	0
Friday	0	0	0	0	1	0
Saturday	0	0	0	0	0	1
Sunday	0	0	0	0	0	0

Nevertheless, these transformations can be beneficial if used as a pre-processing step. For example, applying a moving average to the input time-series can filter noise, and thus improve results in feature selection and modelling steps.

4.3.2.4 Seasonality Dummy Regression Variables

Instead of directly using the Fourier transform, one can include periodic information using *dummy* variables. This is commonly implemented by including the Fourier series transform coefficients, i.e., $\cos(2\pi\omega_i t)$ and $\sin(2\pi\omega_i t)$, as a feature. This effectively helps by allowing *harmonic regression* [182], in the form of

$$\hat{x}(t) = \cdots + \sum_{i=1}^{p} A_i \cos(2\pi\omega_i t) + \sum_{i=1}^{p} B_i \sin(2\pi\omega_i t). \tag{4.8}$$

Alternatively, a similar technique can be used to add dummy variables to differentiate time-periods (e.g., seasons, or days of a week) which show different attributes, but do not exhibit a sinusoid behaviour. For example electricity consumption shows a weekly cycle (higher consumption on weekdays, less consumption on weekends). This can be modelled by assigning a 0–1 variable to each day of the week. This is demonstrated in Table 4.1. Note that only six variables are generated, as the seventh variables is linearly dependent on the others and therefore redundant:

$$V_7 = 1 - \sum_{i=1}^{6} V_i$$

4.3.3 Feature Selection and Dimensionality Reduction

While feature selection and dimensionality reduction are commonly implemented as separate techniques, they both follow the common goal of choosing the most appropriate inputs for the learning algorithm. The logic behind both is to avoid over-fitting by reducing the model's degrees of freedom. Additionally, the *curse of dimensionality* [183] prohibits using high dimensional data in many algorithms by exponentially increasing their time complexity. Hence, reducing the number of features is essential.

Dimensionality reduction commonly takes the form of transforms, such as principal component analysis (PCA), independent component analysis (ICA), or autoencoder neural networks; the data is first transformed to another domain space, and only parts of the data with a significant weight are selected as features [184].

Feature selection directly selects what features are to be used as inputs, and is commonly implemented using one of the following approaches:

- Filter methods, where the relationship between the feature and the output is scored by similarity measures such as correlation or linear regression, and the features with the highest similarity scores are selected. In has been shown that these techniques do not necessarily lead to improvement in predictability [185, 186].
- Wrapper methods, where the features are selected based on the learning algorithm's validation score. The learning algorithm is effectively *wrapped* inside a loop, and evaluates the feature sets based on the prediction results. An outer loop algorithm tries to find the best combination of features, in order to maximise the learner's prediction score. It has been shown that this approach is NP hard [187].

Details of filter and wrapper methods for time-series are discussed extensively in [167].

4.3.4 Choice of Learning Algorithm

Time-series prediction, using a machine learning approach, as formulated in (4.6), requires a supervised regression technique. Some popular techniques include:

- Linear regression: Due to its simplicity and speed, linear regression is widely used in statistics and machine learning. AR models [36] are an example of linear regression for time-series prediction. Various techniques are used to fit the linear model, including ordinary least squares (OLS) and maximum likelihood estimation.
- Artificial neural networks (ANN): ANNs are *universal function approximators*, i.e., any continuous function defined over compact subsets of $\mathbb{R}^n$ can be approximated by one [188]. As a result, ANNs have been extensively used for time-series prediction, and the N in the term NARX originally stood for neural networks [189].

However, because of technical problems, specially the non-convex optimisation formulation of ANNs which is prone to getting stuck in local minimas, ANNs were abandoned in favour of convex algorithms such as SVM. Recently, introduction of extensions such as deep learning techniques [190] and extreme learning machines (ELM) [191] have renewed interest in ANNs.

- Kernel based techniques: Support vector regression machines (SVR or SVM) [169], and kernel recursive least squares (KRLS) [170] are supervised regression algorithms. These techniques are linear in nature (e.g., kernel recursive least squares is a special case of linear adaptive filters), but by using the *kernel trick* they are able to perform in non-linear spaces [192]. The kernel trick, combined with the convex formulation of their optimisation problem, results in fast and versatile non-linear regression techniques.
- Probabilistic techniques: Probabilistic models, such as Gaussian mixture models (GMMs) [193] and Gaussian processes (GP) [194], are able to model data as stochastic processes. By using a custom *covariance function* to model the relationship between different dimensions of the model (i.e., multidimensional inputs and outputs), these techniques are not only able to perform regression, but also provide confidence intervals of the prediction using the distribution's variance. This, however, is at the expense of computational complexity, as these methods commonly use Monte Carlo integration techniques to compute conditional expectations of the underlying stochastic processes.

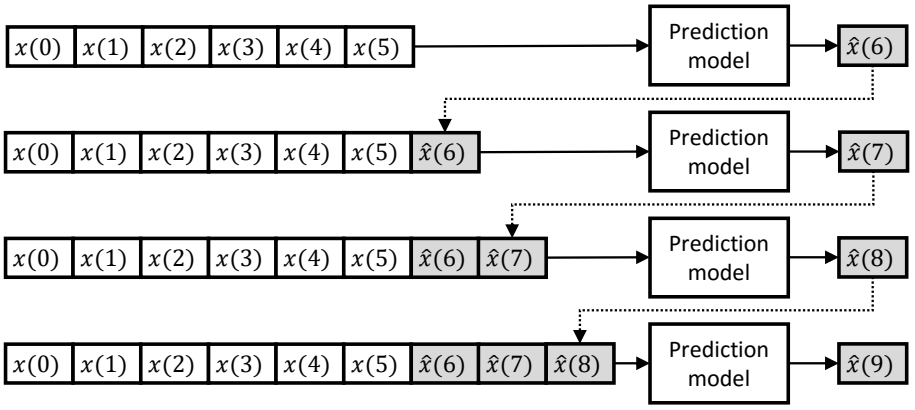
It must be noted that, except for linear regression, most supervised machine learning algorithms require hyper-parameters, such as:

- Neural Networks: number of hidden layers, the number of neurons at each hidden layer, and the choice of activation function.
- Kernel-based techniques: choice of the kernel (including linear, polynomial, radial, sigmoid, $\cdots$) and kernel parameters.
- Probabilistic techniques: choice of the covariance function (including exponential, squared exponential, Matérn, $\cdots$) and its parameters.

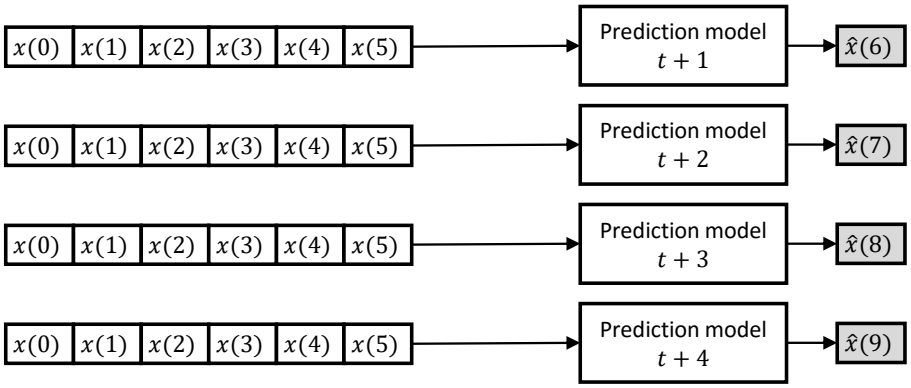
Choosing an optimal hyper-parameter is vital for the performance of the learner. This choice is commonly performed by a model validation technique, namely *cross-validation*, which will be explained in depth in Section 4.5.

4.4 Multi-horizon Forecasts

The machine learning approach of regression, as with the (4.1), only predicts the hth horizon. Many applications, such as a finite horizon control, require forecasting m next



(A) Iterative forecasting.



(B) Direct forecasting.

FIGURE 4.4: Iterative versus direct multi-horizon forecasting.

values of the time-series, i.e., $x(t+1)$ to $x(t+m)$. Two common approaches are used to predict these values [195] (Figure 4.4):

1. Direct method, where m different models are fitted for each horizon.
2. Iterative method, where one single model is fitted and its own previous predictions are used iteratively to compute the next forecast.

Performance of these approaches is data dependent, either favouring a direct method [195] or an iterative approach [196]. Model validation on data for choosing one, or a hybrid approach by combining their results using boosting techniques are recommended [197, 198].

It must be noted, however, that if the prediction horizon is long and the learning process is costly, using an iterative model might be advantageous with regards to the training time.

4.5 Cross-validation for Time-series Prediction

Considering the number of choices (e.g., model order for selecting past lags, the hyper-parameters of the learning algorithm, etc.), model validation is crucial for choosing a good predictor.

Different techniques are available for validating models, including goodness of fit, R^2 (coefficient of determination), Akaike information criterion (AIC), and Bayesian information criterion (BIC) [36]. However, comparing different learning models using these criterion is not possible, as each are built around model dependent assumptions and thus are incompatible.

An alternative technique for comparing model accuracy is *cross-validation* (the term used in statistics and machine learning) or *backtesting* (as referred to in finance). In cross-validation, the model is tested using a simulation with available real-data; the data is divided to several *folds*, each comprised of a training and a testing set. For each fold, the model is fit to the training set, and its output is compared with the testing set. The prediction errors are measured and accumulated, and the model with the least error is selected.

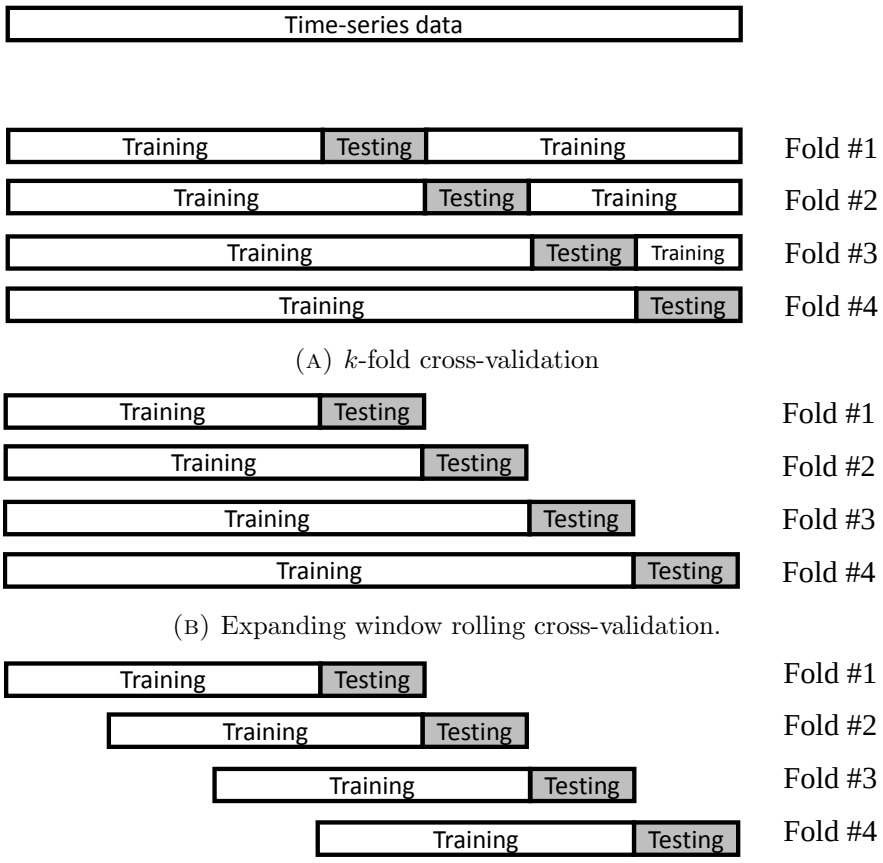
It must be noted that a time-series cross-validation is unlike an ordinary k-fold or leave-p-out cross-validation, where the testing set is selected from *in-between* training data (Figure 4.5a). In time-series cross-validation, only the training data before observation of the testing period has to be used (Figure 4.5b and 4.5c). This is because of the dependency between steps in a time-series, which invalidates the results of ordinary k-fold cross-validation by using future data for modelling the past.

Additionally, when predicting time-series, one might want to include possible non-stationariness, i.e., changes in the dynamics of the process generating this data. As a result, two variants of cross-validation are generally used for time-series:

- Expanding window, where the training window expands with each fold. This method uses the most available data for training the model (Figure 4.5b).
- Fixed window, where the size of the training window is fixed. This technique helps by discarding older training data, where the dynamics of samples are not relevant to the current process any more (Figure 4.5c).

4.5.1 Error Measures for Time-series

Choice of the error measure for cross-validation is problem dependent; however, several commonly used error measures have been proposed, each with their own advantages and



(A) k-fold cross-validation

(B) Expanding window rolling cross-validation.

(C) Fixed window rolling cross-validation.

FIGURE 4.5: Cross-validation strides for time-series prediction.

disadvantages.

Some of these measures include means square error (MSE), defined as

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(x(i) - \hat{x}(i))^2,$$

and root mean square error (RMSE), realised by

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x(i) - \hat{x}(i))^2},$$

which are used specially to penalise larger errors more using a quadratic equation.

In comparison, mean absolute error (MAE), formalised as

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|x(i) - \hat{x}(i)|,$$

treats the errors uniformly.

MSE and MAE are useful for comparing results only on the same time-series. To allow comparisons between different time-series, which may differ in their scale (i.e., minimum and maximum), mean absolute percentage error (MAPE), defined by

$$MAPE = \frac{100}{N} \sum_{i=1}^{N} \frac{|x(i) - \hat{x}(i)|}{x(i)},$$

and symmetric MAPE (sMAPE), realised by

$$sMAPE = \frac{200}{N} \sum_{i=1}^{N} \frac{|x(i) - \hat{x}(i)|}{x(i) + \hat{x}(i)},$$

have been proposed.

A recent measure is mean absolute scaled error (MASE) [199], proposed as

$$MASE = \frac{1}{N} \sum_{t=1}^{N} \frac{|x(t) - \hat{x}(t)|}{\frac{1}{N-1} \sum_{i=2}^{N} |x(i) - x(i-1)|},$$

which is designed to compare the prediction error $|x(t) - \hat{x}(t)|$ against the average *naïve* forecast error. In a naïve forecast, $\hat{x}(i) = x(i-1)$, and consequently its error is $|x(i) - x(i-1)|$. Thus, this measure can be used to show if the forecast is effective compared to a naïve approach.

4.6 Existing Approaches to Automatic Forecasting

Creating a successful forecasting model requires a careful selection of components and hyper-parameters. This is commonly performed manually by an expert who chooses these components and devises cross-validation strategies according to the properties of the target time-series.

To the best of our knowledge, the only completely automatic and peer-reviewed approach to time-series forecasting is the methodology proposed by Hyndman and Khandakar [200], and implemented as the package *forecast* [201] in R [202]. Two main techniques provided are the auto-regressive integrated moving average (ARIMA) and exponentially smoothing (ETS). The automation methodology deploys a step-wise search algorithm to select the ARIMA parameters, and uses an error penalty term to compare and select the appropriate additive/multiplicative model for ETS. The implementations are highly configurable with regards to the parameters of ARIMA and ETS models.

No equivalent ML-based time-series prediction libraries were found. The nearest approach is *tsDyn* package [203] in R, and *narxnet* in MATLAB, both implementing NARX using neural networks. Selecting the model orders, adding exogenous features, and applying pre/post-processing techniques are left out as the user's responsibility.

General purpose machine learning libraries are available, which offer functionality for pre-processing, feature selection, and cross-validation, including *caret* [204] package in R, *Weka* [205] in Java, and *scikit-learn* [206] in Python. Each component is provided separately, and none of these libraries offer a systematic way for creating a complete time-series prediction model.

4.6.1 ARIMA

An ARIMA(p,d,q) process [36] is expressed in polynomial form by

$$\left(1 - \sum_{i=1}^{p} \phi_i L^i\right) (1 - L)^d x(t) = \left(1 + \sum_{i=1}^{q} \theta_i L^i\right) \varepsilon_t \tag{4.9}$$

where L is the lag operator (e.g., $Lx(t) = x(t-1)$ and $L^2 x(t) = x(t-2)$).

The ARIMA model is essentially:

1. a pre-processing step, removing trends by *differencing*. This is parameterised by d, using $(1 - L)x(t) = x(t) - x(t-1)$.
2. an *auto-regressive* component, as in (4.5), parameterised by ϕ_i with model order p.
3. a *moving average* component, i.e., a FIR filter (4.2) parameterised by θ_i with model order q applied to white noise $\epsilon(t)$.

4.6.2 Exponential Smoothing

ETS models [36] use *exponential smoothing* of the past observations as a means of forecasting the future. These models are non-linear in the sense that they cannot be learnt using linear regression techniques, and are interpreted based on state-space representation of the time-series [207, 208]. For example, a simple ETS model, without trend or seasonal component, is realised by

$$\begin{aligned} s_0 &= x(0) \\ s_t &= \alpha x(t) + (1 - \alpha) s_{t-1}, \ t > 0 \end{aligned} \tag{4.10}$$

where s_i is the ith state, α is the smoothing parameter, and the future is forecast using

$$\hat{x}(t+1) = s_t.$$

By adding trend or seasonal components through multiplicative or additive forms, and careful tuning of the parameter, ETS can become comparable or even better than proprietary and case-specific models [207].

4.7 Summary

In this chapter, data flow and ML algorithms for time-series prediction were studied. Prediction was formulated as a regression task based on past observation, and different pre-processing, feature selection, and dimensionality reduction techniques were reviewed. Additionally, methodologies for multi-horizon forecasting and model validation were discussed.

These techniques will be used in Chapter 5 to create an automatic time-series forecasting methodology based on machine learning. Existing approaches to automatic time-series forecasting, namely ARIMA and ETS, were also examined, which will be later compared to the proposed forecasting methodology.

Chapter 5

Grammatical Evolution for Time-series Prediction

5.1 Introduction

Despite more than 50 years of study [209], time-series forecasting is still a cumbersome operation, requiring an experienced *data scientist* to design and build models. This becomes problematic for systems which require a fast response to changing time-series dynamics, as redesigning and validating a new prediction model is time-consuming. Additionally, many businesses face datasets with thousands of different time-series, and adapting a separate model to each becomes impossible. Consequently, with the ever increasing complexity of modern systems and amounts of data being generated each year, it becomes more challenging to create effective time-series prediction software.

In this chapter, a systematic approach to automating time-series prediction using grammatical evolution is proposed. Based on the techniques reviewed in Chapter 4, a generic template for describing machine learning algorithms for time-series prediction is formalised using context-free grammars. Subsequently, the template is utilised within a computational intelligence framework, to automatically create the optimal forecasting model for any given time-series.

5.2 Motivation

In Chapter 3, a model predictive control framework was proposed to manage an FX dealer's risk. While the dealer is modelled using a simple linear formulation, its predictive accuracy relies on several exogenous parameters, including client flow, market volatility,

and the bid-ask spreads. An estimation of the future values of these parameters is required, and the controller's efficiency depends on the quality of this estimation.

As explained in Section 3.6.3, one of the techniques for estimating these parameters is time-series prediction, where each time-varying parameter is treated as a time-series, and prediction algorithms are applied to forecast their future values.

In addition to FX, renewable energy management techniques also benefit from time-series prediction. The amount of electricity consumption, as well as its generation using renewable resources, are not known in advance; as a result, any forward planning, either short or long-term, requires an accurate forecast of generation and consumption.

5.3 A Template for Time-series Prediction

In this section, we formalise a structured algorithmic approach to time-series prediction based on the techniques and requirements examined in Chapter 4.

The proposed method is composed of two major components:

- A customisable template for prediction algorithms.
- An optimisation scheme to choose the best template configuration for a given time-series.

The rest of this section discusses the customisable algorithm in more depth, and the optimisation scheme is discussed in the next.

5.3.1 A Customisable Prediction Algorithm

We propose a time-series prediction algorithm, comprised of a *training* function which fits a ML model using historical data, and a *prediction* function that uses the ML model to forecast the future over a horizon.

The training function, summarised in Algorithm 5.1, is divided to three major parts:

1. Pre-processing: A <*pre-processing techniques list*>, containing selected techniques and their parameters, is applied to the input time-series x.
2. Feature extraction: The `Feature Extraction` function is used to extract features from the pre-processed time-series.
3. Machine learning model training: A <*regression algorithm*> is used to create the prediction `model`, which can map the features to the time-series.

Algorithm 5.1: The proposed time-series prediction training algorithm.

Input: time-series x.
Result: Prediction model

```
// pre-processing
pre-processing techniques ← <pre-processing techniques list>
foreach (technique, hyper-parameters) in pre-processing techniques do
    x, parameters ← Apply technique with hyper-parameters to x
    pre-processing techniques ←(Append)— parameters
end

// Feature extraction
features ← Apply Feature Extraction to x

// ML Training
learner ← <regression algorithm>
model ← Apply learner on <m> last features

return model, pre-processing techniques
```

Notice that the <*pre-processing techniques list*>, <*regression algorithm*>, and <m> are covered in *angle brackets*. We use this notation, similar to non-terminal elements in a context-free grammar (CFG) in Backus–Naur form, to show that these components are later replaced by actual pre-processing methods and regression algorithms.

The prediction function, summarised in Algorithm 5.2, has four components:

1. Pre-processing: The same pre-processing techniques are applied to the time-series x.
2. Feature extraction: The same feature extraction function is used to obtain features.
3. Prediction: The `model` created in the training phase is used for prediction. The prediction is able to forecast h steps ahead by iteratively using its own output as new observations.
4. Post-processing: Reverses the pre-processing step on the prediction results.

This function, unlike the training function, is not customisable, and the information regarding the choice of techniques and models are set via function arguments.

The proposed training and prediction methodologies follow the discussion in Section 4.3, specifically the data flow explained in Figure 4.2, but with a major difference; feature extraction, addition of exogenous features, and feature selection are combined and replaced with the `Feature Extraction` function. This is explained in detail in Section 5.3.3.

Ordinarily, the training and the prediction functions are executed back to back after the observation of each new sample. This is to update the forecasting model using the new

Algorithm 5.2: The proposed forecasting algorithm.

Input: time-series x, forecasting horizon h, pre-processing techniques, ML model, feature window size m.
Result: future forecasts $\hat{x}$

```
// pre-processing
foreach (technique, hyper-parameters) in pre-processing techniques do
    x, parameters ← Apply technique with hyper-parameters to x
    pre-processing techniques ←(Append)— parameters
end

// Rolling window prediction
x̂ ← x
for i = 1 to h do
    features ← Apply Feature Extraction on x̂
    x̂ ←(Append)— Predict model with the last m features
end

// Reverse pre-processing Phase
foreach (technique, hyper-parameters, parameters) in pre-processing techniques do
    x̂ ← Reverse technique on x̂ with hyper-parameters and parameters
end
```

information in order to obtain better prediction results. However, in several cases one may want to separate these processes:

- If time-series dynamics do not change often, re-fitting the model can be avoided to save on computational costs.
- If real-time or high throughput prediction is required, computational load of model training can be offloaded to a low-priority background process or a separate computing device, and prediction is performed on a high priority foreground process.
- Embedded systems with low computational capacity can offload model training to a more capable computer. For example, training is performed on a cloud computing system, and model parameters are transferred to the local low-power embedded system. While the cloud computing system can be also responsible for forecasting, storing the prediction model locally allows continuous operation even when the communication to the cloud breaks down.

5.3.2 Pre-processing Grammar

The CFG in Table 5.1 is proposed to replace the <*pre-processing techniques list*> in Algorithm 5.2. The terminal and non-terminal elements used in this grammar are:

TABLE 5.1: Production rules for the time-series prediction pre-processing grammar.

⟨*techniques list*⟩	::= {<*techniques list*>, <*technique*>} | <*technique*>	(1.a) (1.b)
⟨*technique*⟩	::= (normalise) | (feature scale) | (diff, <*ndiff*>) | (sdiff, <*nsdiff*>, <T>) | (log) | (BoxCox, <λ>)	(2.a) (2.b) (2.c) (2.d) (2.e) (2.f)
⟨*ndiff*⟩	::= <n> | KPSS | ADF | PP	(3.a) (3.b) (3.c) (3.d)
⟨*nsdiff*⟩	::= <n> | CH | OCSB	(4.a) (4.b) (4.c)
⟨T⟩	::= known frequency | FFT	(5.a) (5.b) (5.c)
⟨λ⟩	::= Guerrero | λ_{min} | $\cdots$ | λ_{max}	(6.a) (6.b) (6.c) $\cdots$
⟨n⟩	::= 1 | 2 | $\cdots$ | n_{max}	(7.a) (7.b) (7.c) $\cdots$

- <*techniques list*>: A list of selected pre-processing techniques. This element is recursive, which can extend to include one or multiple <*technique*>s.
- <*technique*>: Contains a pair of (method, parameters). Several pre-processing techniques are included:
  - mean-variance normalisation, or feature scaling. These techniques are non-parametric.
  - differencing (diff) and seasonal differencing (sdiff), to create a stationary time-series by removing trends and seasonality.
  - power transforms, including logarithm and Box–Cox transform.
- <*ndiff*> and <*sndiff*>: Order of ordinary and seasonal differencing. These values can be obtained using unit root tests, as explained in Section 4.3.1. Tests include ADF, KPSS, and PP for ordinary differences, and CH and OCSB for seasonal differences. An alternative option is to directly choose the order from a list of integer numbers, <n>.
- <T>: The time-series periodicity. If this frequency is known (for example weekly or daily) it can be directly used. Otherwise, a spectrogram using FFT can extract the dominant frequency of the time-series.
- <λ>: The λ parameter of the Box–Cox transform, as explained in (4.4). The value of λ is either chosen from a user specified set, $[\lambda_{min}, \cdots, \lambda_{max}]$, or obtained using the Guerrero's variance-stabilising method [210].
- <n>: A positive integer. The maximum value is controllable by user specified value $n_{max} \in \mathbb{N}$.

The statistical techniques used above are well studied, and are available from several statistical software packages [201, 202, 211].

5.3.3 Feature Extraction Grammar

The training and prediction algorithms use the `Feature Extraction` function to prepare the machine learning model's input. Similar to the pre-processing step, it contains several customisable options as it can select from a pool of different endogenous and exogenous features. The customisability has two different goals. First, the end user can easily update the pool with new features. Second, this unifies feature selection and extraction algorithms: *wrapper* feature selection algorithms work by comparing different feature combinations. If the customisable feature extraction can also create different feature combinations, only one external validation step would be required.

The `Feature Extraction` function, summarised in Algorithm 5.3, has three main steps:

1. Endogenous features are extracted according to (4.7) (i.e., seasonal AR model). The features are parameterised by
   - $<AR\ order>$, i.e., the model order p in (4.7), which determines the window size of lagged values used as features,
   - $<seasonal\ AR\ order>$, the model order P in (4.7), which determines the window size of seasonally lagged values, and
   - $<T>$, similar to $<T>$ in Table 5.1 and the T in (4.7), which determines the frequency of the lagged seasonal components.
2. Exogenous features are added in accordance with the NARX model of (4.6). These features are parameterised by
   - the $<additional\ time\text{-}series>$, which contains a list of related time-series, suggested by the user, and
   - the $<window\ size>$, which determines the order of the lags and leads for each time-series included in the $<additional\ time\text{-}series>$. The parameter q^- is used to determine the order of past lags (similar to the order q in (4.6)), and q^+ sets the order of future leads (e.g., $q^+ = 1$ for $\hat{u}(t+1)$ in Figure 4.3). One must note that the selected window of the exogenous time-series can only include *available* forecasts, and thus q^+ should be limited to the number of available step-ahead forecasts for that exogenous time-series.
3. Time-dependent features are added as *dummy variables*. These include calendar based variables (e.g., day-of-week as in Table 4.1), and Fourier transform coefficients as in (4.8).

Table 5.2 lists the production rule for customising the `Feature Extraction` function:

- $<AR\ order>$, $<seasonal\ AR\ order>$, and $<X\ order>$: Determine the candidate orders for the auto-regressive, seasonal auto-regressive, and exogenous features.

Algorithm 5.3: The proposed feature extraction algorithm for time-series prediction.

Input: Time-series x
Result: Extracted features

Function *Feature Extraction* **is**
  `// Extract endogenous features`
  $p = <AR\ order>$
  $P = <seasonal\ AR\ order>$
  $\tau = <T>$
  features $\leftarrow [x(t-1), x(t-2), \cdots, x(t-p)]$
  features $\xleftarrow{Append}$ $[x(t-\tau), x(t-2\tau), \cdots, x(t-P\tau)]$
  `// Extract exogenous features`
  additional time-series $\leftarrow <additional\ time\text{-}series>$
  window sizes $\leftarrow (q^-, q^+) \in <X\ order>$ for each additional time-series
  foreach y, q^-, q^+ **in** additional time-series *and* window sizes **do**
    $q^+ \leftarrow \min(q^+,$ max available future forecasts for $y)$
    features $\xleftarrow{Append}$ $[y(t-q^-), \cdots, y(t-1), y(t), y(t+1), \cdots, y(t+q^+)]$
  end
  `// Add seasonality dummy variables`
  features $\xleftarrow{Append}$ $<dummy\ variables\ list>$
  return features
end

These values are limited by the user suggested orders p_{max}, P_{max}, and q_{max} respectively.

- $<T>$: The time-series periodicity. If this frequency is known (for example weekly or daily) it can be directly used. Otherwise, a spectrogram using FFT can extract the dominant frequency of the time-series.
- $<exogenous\ time\text{-}series>$: A list of related exogenous time-series, supplied by the user.
- $<additional\ time\text{-}series>$: A recursive list, selecting time-series form the $<exogenous\ time\text{-}series>$ to be used as features.
- $<dummy\ variables>$: A list of explanatory dummy variables, as described in Section 4.3.2.4.
- $<dummy\ variables\ list>$: A recursive list, selecting form $<dummy\ variables>$ to be added as features.

TABLE 5.2: Production rules for the feature extraction grammar.

$\langle AR\ order \rangle$	::= 1 | 2 | $\cdots$ | p_{max}
$\langle X\ order \rangle$	::= 0 | 1 | 2 | $\cdots$ | q_{max}
$\langle seasonal\ AR\ order \rangle$	::= 0 | 1 | 2 | $\cdots$ | P_{max}
$\langle T \rangle$	::= known frequency | FFT
$\langle additional\ time\text{-}series \rangle$	::= {<*additional time-series*>, <*exogenous time-series*>} | <*exogenous time-series*> | None
$\langle exogenous\ time\text{-}series \rangle$	::= {List of exogenous time-series}
$\langle dummy\ variables\ list \rangle$	::= {<*dummy variables list*>, <*dummy variable*>} | <*dummy variable*> | None
$\langle dummy\ variable \rangle$	::= hour-of-day | week-of-day | month-of-year | $\cdots$ | calendar holidays | $sin(2\pi \frac{t}{<T>})$ | $cos(2\pi \frac{t}{<T>})$

5.3.4 Learning Model Grammar

The final component for training and predicting time-series is a machine learning model. While the prediction algorithm re-uses the same model that was used during the training, the user is responsible for choosing the machine learning model in the training algorithm. Additionally, the user has to tune the model's hyper-parameters, either manually, or using a search method such as random search or grid search.

To automate and unify learning model and hyper-parameter selection, we propose the grammar in Table 5.3:

- <*learner*> selects the learning model, and determines what hyper-parameters are required for configuring it.
- Other production rules customise the selected model's hyper-parameters. For example, the ANN is parameterised by the number of <*hidden layers*> and <*hidden nodes per layer*>, while the kernel methods are configured with type of <*kernel*> and its parameters such as <γ> (the Gaussian kernel width) or d (the order of polynomial kernel). Each kernel method has other additional hyper-parameters, such as the soft margin cost C for SVM, and the regularisation parameter λ for KRLS. The range of each parameter is controllable by the user, allowing the algorithm to search within a fine-grained or course-grained list.
- As explained in Section 4.5, the training can be performed on all available data to maximise training samples, or just on the recent features to reject older and non-relevant samples. The parameter <m> is used to set the number of features for this purpose. This can vary from a minimal number set by the user in m_{min}, up to all of the available features.

TABLE 5.3: Example of production rules for the hyper-parameter selection grammar.

$\langle learner \rangle$	::=	(ANN, <*hidden layers*>, <*neurons per layer*>) | (SVM, <*kernel*>, <ϵ>, <C>) | (KRLS, <*kernel*>, <λ>)
$\langle hidden\ layers \rangle$	::=	$1 \mid 2 \mid \cdots \mid L_{max}$
$\langle neurons\ per\ layer \rangle$	::=	$1 \mid 2 \mid \cdots \mid N_{max}$
$\langle kernel \rangle$	::=	Linear | {Radial, <γ>} | {Polynomial, <d>} $\mid \cdots$
$\langle C \rangle$	::=	$C_{min} \mid \cdots \mid C_{max}$
$\langle \epsilon \rangle$	::=	$\epsilon_{min} \mid \cdots \mid \epsilon_{max}$
$\langle \lambda \rangle$	::=	$\lambda_{min} \mid \cdots \mid \lambda_{max}$
$\langle \gamma \rangle$	::=	$\gamma_{min} \mid \cdots \mid \gamma_{max}$
$\langle d \rangle$	::=	$d_{min} \mid \cdots \mid d_{max}$
$\langle m \rangle$	::=	$m_{min} \mid \cdots \mid$ all

5.4 Grammatical Evolution for Time-series Prediction

In Section 5.3, a customisable template for non-linear time-series prediction was proposed. While any custom configuration of this template results in a valid predictor, the choice of components directly determines the forecasting quality.

In this section, we use grammatical evolution (GE) to search for the optimal forecasting configuration. GE, introduced in Section 2.6, is an evolutionary search technique used for generating complete programs optimised towards a certain task. Implementing GE for any domain-specific task requires 3 major components:

1. A grammar: Context-free grammars (CFG) are used to determine the structure of the program.
2. A cost function: The cost (or fitness) function compares different configurations of the program.
3. An evolutionary search technique: Considering the cost function is usually non-smooth and non-convex, gradient-based optimisation algorithms are unable to find the optimal grammar configuration. As a result, evolutionary optimisation techniques such as genetic algorithm (GA) are used.

Specific to the time-series prediction, these components are already explored:

- The grammar, which was proposed in Section 5.3.
- The cost function: The best technique to compare time-series prediction algorithms is cross-validation on the target data, as introduced in Section 4.5.

5.4.1 The Evolutionary Search

Evolutionary optimisation algorithms, such as GA, are inspired by natural evolution. These algorithms alter a *population* of candidate solutions, searching within the feasible solution space for an *individual* solution with the minimum *cost* (or maximum *fitness*).

We implement the grammatical evolution using GA with integer chromosomes:

1. GA creates a population of integer sequences, called *chromosomes*.
2. A *grammar mapping* engine, maps each chromosome to a configured forecasting function. This is performed by replacing each component in the prediction template algorithms, with a rule from the grammar tables, selected according to the value in chromosomes. Similar to translation of DNA to a complete organism in biology, this step is referred to as *genotype to phenotype mapping*.
3. Each prediction algorithm is cross-validated using the target time-series.
4. The cross-validation error is returned to the GA as the fitness score or the cost.
5. Ordinary GA operators are applied to the population to create a new *generation* of population. These operators include:
   - Selection: Only predictors with low cross-validation error (i.e., high fitness and low cost) are kept.
   - Cross-over: Combines the integer sequence from two chromosomes to create new chromosomes.
   - Mutation: Some chromosomes are randomly perturbed, and thus the search is extended to their nearby solution space.
6. The algorithm is repeated for the new generation from step 2, until a termination condition is reached. Termination conditions include reaching a desired level of cross-validation score, or a maximum number of generations.

The proposed implementation of the GE optimisation is presented in Figure 5.1.

One must note that as GE is a *stochastic* optimisation, i.e., it randomly searches the solution space. As a result, on some occasions it might be unable to find the best prediction function. Alternatively, an exhaustive search can be performed to find the global minima at the expense of computation time. This technique, however, might not be practical for larger problems.

5.4.2 The Chromosome Structure

Unlike other program generating evolutionary algorithms, such as genetic programming (GP) which require a specific structure for their chromosomes, GE only uses a string

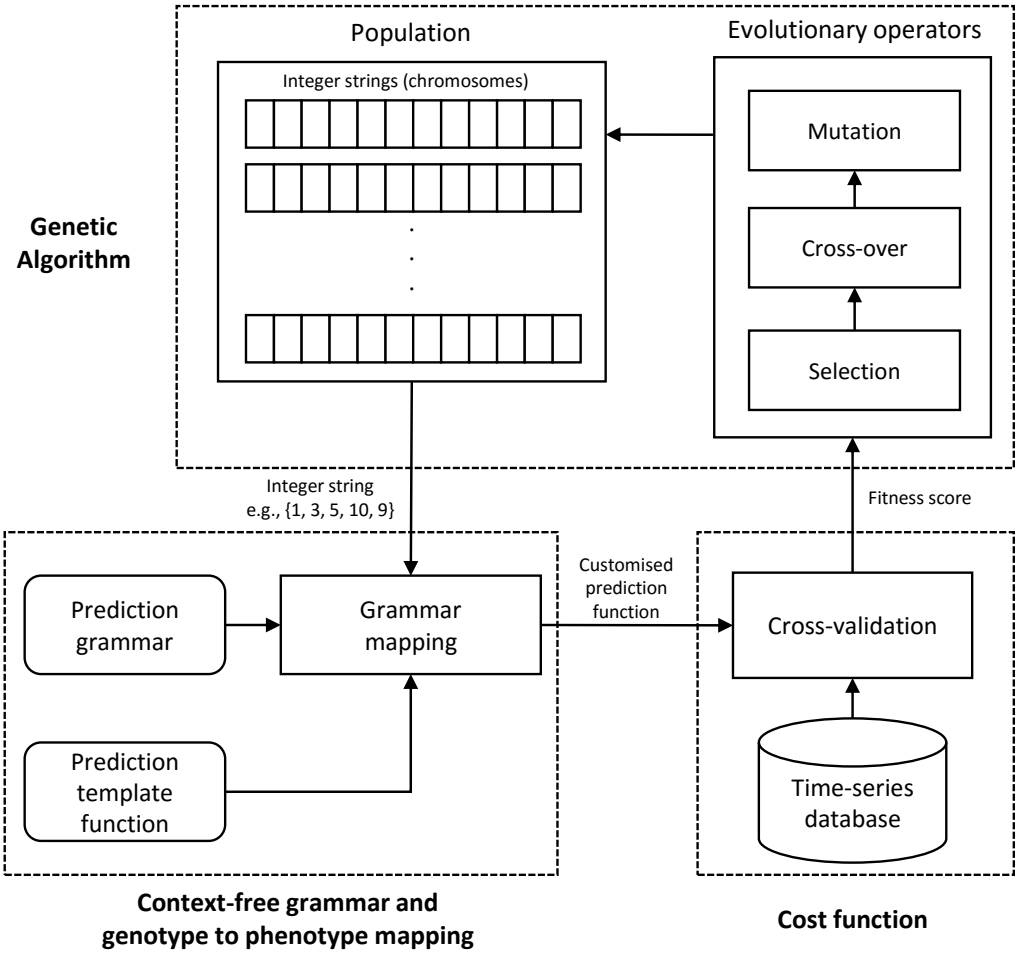


FIGURE 5.1: Grammatical evolution for time-series prediction.

of integer numbers. Creating a prediction function is performed by replacing each non-terminal element of Algorithms 5.1, 5.2, and 5.3 with an according rule defined by the grammars in Tables 5.1, 5.2, and 5.3.

Consider the example in Table 5.4, where the <*pre-processing techniques list*> of Algorithm 5.1 is transformed into two pre-processing functions, namely normalisation and differencing with the order determined by ADF test. The chromosome in this example is $\{0, 1, 0, 2, 2\}$. The mapping starts from the *start rule*, which in this case is the <*techniques list*>. The first *codon* (i.e., the integer value 0) is used with a zero-indexing scheme to choose rule (1.a). The non-terminal is then replaced by this rule, i.e., {<*techniques list*>, <*technique*>}. Mapping continues by replacing the first non-terminal element in the new state (i.e., a new <*techniques list*>), using the next codon (i.e., the integer value 1). When all non-terminals are replaced, as in step 6, the mapping terminates.

TABLE 5.4: Genotype to phenotype mapping example using the pre-processing grammar.

Step	Current State	Sequence	Rule	Replacement
0			Start	<*techniques list*>
1	**<*techniques list*>**	0	(1.a)	{<*techniques list*>, <*technique*>}
2	{**<*techniques list*>**, <*technique*>}	1	(1.b)	<*technique*>
3	{**<*technique*>** , <*technique*>}	0	(2.a)	(normalise)
4	{(normalise), **<*technique*>** }	2	(2.c)	(diff, <*ndiff*>)
5	{(normalise), (diff, **<*ndiff*>**) }	2	(3.c)	ADF
6	{(normalise), (diff, ADF) }			

5.4.3 Fitness Function

The fitness function requires a model-independent technique to compare the forecasting quality of different time-series prediction algorithms. Cross-validation, studied in Section 4.5, satisfies these requirements.

Determining cross-validation details are task dependent, and the following parameters have to be determined by the user:

- h: The forecasting horizon.
- N_{folds}: The number of folds to use in cross-validation. More folds result in more testing, but the data available for training will be reduced, and vice versa.
- Error measure for comparing the predictions against the observed time-series. Common error measure used for comparing prediction results are listed in Section 4.5.1. It must be noted an appropriate measure depends on the application of prediction, and how the prediction error affects that application's results. This, focusing on a linear-quadratic (LQ) control application, is further studied in Chapter 6.

Given this information, the cross-validation can be applied to each prediction algorithm in the following sequence:

- The data is divided to N_{folds} training and testing fold, considering that test folds have to be of length h.
- For each fold, the model is trained on the training data.
- The prediction function is applied iteratively to predict h steps ahead.
- The prediction is compared against the testing data, and its error is measured.
- The errors are accumulated and returned as the cost (i.e., the inverse of fitness).

This fitness function is summarised as pseudocode in Algorithm 5.4

Algorithm 5.4: The proposed fitness function for time-series prediction evaluation.

Input: Time-series x, forecasting horizon h, number of folds N_{folds}, error measure, training function, prediction function
Result: Cross-validation score

```
1 e_total ← 0
2 folds ← divide x into N_folds training and testing folds
3 foreach (train data, test data) in folds do
4 |   model ← Apply training function on train data
5 |   forecast ← Apply prediction function on model with forecast horizon h
6 |   e ← error measure(forecast, test data);
7 |   e_total ← e_total + e
8 end
9 return e_total
```

5.4.3.1 Speeding-up the Fitness Function

Computing the fitness is the most time-consuming part of any evolutionary algorithm. For time-series cross-validation, the main speed bottleneck is the training and testing algorithms applied to different folds of data. By improving fitness computation speed, more grammar space can be explored, which leads to finding better prediction functions in a limited time.

In this thesis, a combination of these techniques will be used to speed up fitness function computation:

- Search parallelisation: In an evolutionary search algorithm, such as genetic algorithm, computing fitness function of each chromosome is independent of the others. As a result, each chromosome can be evaluated on a different processor, and the communication overhead would be negligible.
- Cross-validation parallelisation: The time-series can be broken into different sections, with cross-validation of each section assigned to a different processor. One must note that as time-series samples are sequential in nature, some of the results for samples around the breaking points will be lost. Hence, a mechanism is required to divide the time-series into *overlapping* sections, and re-assemble cross-validation results into the final cost [168].
- Fast breaks: Some prediction functions may show a considerably high prediction error, or even terminate in error (e.g., logarithmic pre-processing used with negative numbers). If the error is identified to surpass the best fitness early in the cross-validation process, it is unnecessary to continue the cross-validation.

5.5 Summary

In this chapter, grammatical evolution (GE) was proposed to automate non-linear time-series prediction. Based on the state-of-the art techniques in literature, a customisable template for describing a machine learning (ML) methodology for time-series prediction was formalised using context-free grammars (CFGs). Subsequently, an optimisation framework based on GE was proposed, which allows finding the optimal template configuration for any given time-series requirements.

An important feature of the proposed method is its extensibility. If any new information regarding the time-series is known, it can be added to one of the forecasting steps; furthermore, additional methods are easily added as another rule, and known poor performing techniques are removed before executing the GE search.

This technique will be used to automatically find the optimal time-series prediction algorithms, for the applications where:

- time-series dynamic change quickly, e.g., financial markets, or
- the number of time-series to predict is large enough for an analyst to study, e.g., the energy market.

Chapter 6

Time-series Forecasting Error Measures for Finite Horizon Control

6.1 Introduction

In Chapter 3, an FX dealer's risk was modelled using a linear-quadratic (LQ) system, and a shrinking horizon MPC controller was proposed for hedging it. This model requires an estimation of future system parameters, and the controller's efficiency depends on the accuracy of these estimations.

This problem is not limited to hedging. Many real-world systems are described by an LQ model (i.e., a set of linear differential equations and a quadratic objective function), and finite horizon optimal control regulators are used for controlling them. Applications include chemical engineering [212], electric power systems [213], and inventory management [125].

In finite horizon control (FHC), as with other optimal control techniques, a state-space model is defined to allow *predictive modelling* of the system's future states to any input. Based on this model, controllable inputs of the system are determined such that the objective cost function is minimised, and thus the system is controlled towards the desirable outcome.

Classic FHC formulations model the system's uncontrollable exogenous inputs analytically as a part of the state-space equations. This is performed by dividing these inputs into *measurable* and *non-measurable external disturbances*, and using an analytical model to describe them [8]. However, due to the complexity of real-world applications,

developing an accurate analytical model for the system's exogenous inputs is not always possible.

One relatively recent solution is the use of statistical models and machine learning (ML) algorithms as a part of the predictive model, where the time-varying exogenous inputs are treated as a time-series, and estimated via a forecasting algorithm [214–218]. In time-series forecasting, error measures such as mean square error (MSE) or mean absolute error (MAE) are used for model validation, by comparing the effectiveness of different forecasting techniques and their parameters for a certain prediction task [199]. However, these assume that the error of each sample is independent of other time-steps, which is not generally true. To obtain full accuracy for model selection, one must validate a model by *backtesting* over the control horizon. Alternative techniques exist, such as including the conditional distribution of the exogenous inputs in the dynamic programming problem [121]. Unfortunately, none of these approaches are computationally efficient.

In this chapter, the criteria for evaluating and selecting an appropriate forecasting model for discrete-time LQ FHC regulators are discussed. As will be demonstrated, this problem has its own unique challenges.

6.2 A Notation for Optimal Control Problems

A discrete linear time-invariant (LTI) system can be described using the following state-space model:

$$\boldsymbol{x}(t+1) = \mathbf{A}\boldsymbol{x}(t) + \mathbf{B}\boldsymbol{u}(t) + \mathbf{C}\boldsymbol{v}(t) \tag{6.1}$$

Here,

- $\boldsymbol{x}(t) \in \mathbb{R}^k$ is the state vector at time $t \in [0, 1, \ldots, n]$,
- $\boldsymbol{u}(t) \in \mathbb{R}^l$ is the vector of controllable inputs,
- $\boldsymbol{v}(t) \in \mathbb{R}^m$ is the vector of exogenous inputs,

and $\mathbf{A} \in \mathbb{R}^{k\times k}$, $\mathbf{B} \in \mathbb{R}^{k\times m}$ and $\mathbf{C} \in \mathbb{R}^{k\times n}$ are the system and input matrices respectively and are controllable.

The objective is to find controllable inputs, $\boldsymbol{u}(t)$, to minimise a quadratic cost function defined by

$$J = \sum_{t=1}^{n} \boldsymbol{x}(t)^{\mathrm{T}}\mathbf{Q}_t\boldsymbol{x}(t) + \sum_{t=0}^{n-1} \boldsymbol{u}(t)^{\mathrm{T}}\mathbf{P}_t\boldsymbol{u}(t) \tag{6.2}$$

where positive-semidefinite matrix $\mathbf{Q}_t \in \mathbb{R}^{k\times k}$ and positive-definite matrix $\mathbf{P}_t \in \mathbb{R}^{l\times l}$ are stage costs of $\boldsymbol{x}(t)$ and $\boldsymbol{u}(t)$ respectively.

6.2.1 Matrix Form Solution

A common way to find the solution to argmin J is to explicitly express $\boldsymbol{x}(t), \forall t > 0$ as a function of inputs through matrix form, and then minimise J to find all $\boldsymbol{u}(t)$ in a *batch* approach [8].

In matrix form, the state-space equation (6.1) and the cost function (6.2) are represented by

$$\boldsymbol{X} = \mathbf{S}_A\boldsymbol{x}(0) + \mathbf{S}_B\boldsymbol{U} + \mathbf{S}_C\boldsymbol{V} \tag{6.3}$$

$$J = \boldsymbol{X}^{\mathrm{T}}\bar{\mathbf{Q}}\boldsymbol{X} + \boldsymbol{U}^{\mathrm{T}}\bar{\mathbf{P}}\boldsymbol{U} \tag{6.4}$$

where

$$\begin{aligned}
\boldsymbol{X} &= [\boldsymbol{x}(1)^{\mathrm{T}}\ \boldsymbol{x}(2)^{\mathrm{T}}\ \cdots\ \boldsymbol{x}(n)^{\mathrm{T}}]^{\mathrm{T}},\\
\boldsymbol{U} &= [\boldsymbol{u}(0)^{\mathrm{T}}\ \boldsymbol{u}(1)^{\mathrm{T}}\ \cdots\ \boldsymbol{u}(n-1)^{\mathrm{T}}]^{\mathrm{T}},\\
\boldsymbol{V} &= [\boldsymbol{v}(0)^{\mathrm{T}}\ \boldsymbol{v}(1)^{\mathrm{T}}\ \cdots\ \boldsymbol{v}(n-1)^{\mathrm{T}}]^{\mathrm{T}},
\end{aligned}$$

$$\mathbf{S}_A = \begin{bmatrix} \mathbf{A} \\ \mathbf{A}^2 \\ \vdots \\ \mathbf{A}^n \end{bmatrix},$$

$$\mathbf{S}_B = \begin{bmatrix} \mathbf{B} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{AB} & \mathbf{B} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}^{n-1}\mathbf{B} & \mathbf{A}^{n-2}\mathbf{B} & \cdots & \mathbf{B} \end{bmatrix},$$

$$\mathbf{S}_C = \begin{bmatrix} \mathbf{C} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{AC} & \mathbf{C} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}^{n-1}\mathbf{C} & \mathbf{A}^{n-2}\mathbf{C} & \cdots & \mathbf{C} \end{bmatrix},$$

$$\begin{aligned}
\bar{\mathbf{Q}} &= \mathrm{blockdiag}\{\mathbf{Q}_1, \mathbf{Q}_2, \cdots, \mathbf{Q}_n\},\\
\bar{\mathbf{P}} &= \mathrm{blockdiag}\{\mathbf{P}_0, \mathbf{P}_1, \cdots, \mathbf{P}_{n-1}\},
\end{aligned}$$

and blockdiag$\{\mathbf{M}_1, \mathbf{M}_2, \cdots, \mathbf{M}_N\}$ creates a block diagonal matrix from matrices $\mathbf{M}_1$ to $\mathbf{M}_N$.

We define $\boldsymbol{Y}$ as the vector of uncontrollable variables, i.e., the accumulated initial state and exogenous inputs, formalised as

$$\boldsymbol{Y} = \mathbf{S}_C^{\dagger}\mathbf{S}_A\boldsymbol{x}(0) + \boldsymbol{V} \tag{6.5}$$

where † is the Moore–Penrose pseudoinverse operator.

Theorem 6.1. *Let*

$$\begin{aligned}
\mathbf{J}_A &= \bar{\mathbf{P}} + \mathbf{S}_B^{\mathrm{T}}\bar{\mathbf{Q}}\mathbf{S}_B,\\
\mathbf{J}_B &= \mathbf{S}_B^{\mathrm{T}}\bar{\mathbf{Q}}\mathbf{S}_C,\\
\mathbf{J}_C &= \boldsymbol{Y}^{\mathrm{T}}\mathbf{S}_C^{\mathrm{T}}\bar{\mathbf{Q}}\mathbf{S}_C\boldsymbol{Y}, \textit{ and}\\
\mathbf{H} &= -\mathbf{J}_A^{\dagger}\mathbf{J}_B.
\end{aligned}$$

The cost of the system as a function of controllable and uncontrollable inputs $\boldsymbol{U}$ and $\boldsymbol{Y}$ is realised by

$$J(\boldsymbol{U}) = \boldsymbol{U}^{\mathrm{T}}\mathbf{J}_A\boldsymbol{U} + 2\boldsymbol{U}^{\mathrm{T}}\mathbf{J}_B\boldsymbol{Y} + \mathbf{J}_C. \tag{6.6}$$

Proof. We begin by replacing $\boldsymbol{X}$ in (6.4) with (6.3):

$$\begin{aligned}
J &= \boldsymbol{X}^{\mathrm{T}}\bar{\mathbf{Q}}\boldsymbol{X} + \boldsymbol{U}^{\mathrm{T}}\bar{\mathbf{P}}\boldsymbol{U}\\
&= (\mathbf{S}_A\boldsymbol{x}(0) + \mathbf{S}_B\boldsymbol{U} + \mathbf{S}_C\boldsymbol{V})^{\mathrm{T}}\bar{\mathbf{Q}}(\mathbf{S}_A\boldsymbol{x}(0) + \mathbf{S}_B\boldsymbol{U} + \mathbf{S}_C\boldsymbol{V})\\
&\quad + \boldsymbol{U}^{\mathrm{T}}\bar{\mathbf{P}}\boldsymbol{U}
\end{aligned}$$

Considering the definition of $\boldsymbol{Y}$ from (6.5), this can be simplified to

$$J = \ (\mathbf{S}_B\boldsymbol{U} + \mathbf{S}_C\boldsymbol{Y})^{\mathrm{T}}\bar{\mathbf{Q}}(\mathbf{S}_B\boldsymbol{U} + \mathbf{S}_C\boldsymbol{Y}) + \boldsymbol{U}^{\mathrm{T}}\bar{\mathbf{P}}\boldsymbol{U},$$

which is subsequently expanded and reordered by $\boldsymbol{U}$:

$$\begin{aligned}
J = &\boldsymbol{U}^{\mathrm{T}}(\bar{\mathbf{P}} + \mathbf{S}_B^{\mathrm{T}}\bar{\mathbf{Q}}\mathbf{S}_B)\boldsymbol{U}\\
&+ \boldsymbol{U}^{\mathrm{T}}(\mathbf{S}_B^{\mathrm{T}}\bar{\mathbf{Q}}\mathbf{S}_C)\boldsymbol{Y}\\
&+ \boldsymbol{Y}^{\mathrm{T}}\mathbf{S}_C^{\mathrm{T}}\bar{\mathbf{Q}}\mathbf{S}_C\boldsymbol{Y}
\end{aligned}$$

The proof is complete.

□

Finally, using the optimality condition, namely $\frac{dJ}{d\boldsymbol{U}} = 0$, the optimal control input $\boldsymbol{U}^* = \operatorname{argmin} J$ can be obtained:

$$\begin{aligned} \boldsymbol{U}^* &= -\mathbf{J}_A^{\dagger}\mathbf{J}_B\boldsymbol{Y} \\ &= \mathbf{H}\boldsymbol{Y}. \end{aligned} \tag{6.7}$$

6.2.2 Finite Horizon Control

The goal of an FHC regulator is to dynamically minimise J over control period $t \in [0, 1, \ldots, N]$. Due to practical considerations, at each time-step, (6.7) is solved over a finite horizon of length $n < N$ to obtain the optimum control sequence $\boldsymbol{U}$, and only the first action of this sequence, $\boldsymbol{u}(0)$, is applied. As time moves forward, the model is updated based on observations, and the finite horizon optimisation is repeated. This update allows better control of the system in presence of external disturbances and model misspecification, at the expense of computational power required for repeated optimisation at each time-step.

Updating the horizon at the next time-step takes two major forms:

1. The horizon is either moved forward, becoming the *receding* horizon control (RHC).
2. The same termination time is held, resulting in the *shrinking* horizon control (SHC).

In this chapter, both methodologies are studied.

6.3 Closed-form Analysis of the Final Cost

6.3.1 Prediction Error

In a real system, the exogenous inputs $\boldsymbol{v}(t)$ are not observed before time t and must be substituted by $\hat{\boldsymbol{v}}(t) = \boldsymbol{v}(t) + \boldsymbol{\epsilon}(t)$, where $\hat{\boldsymbol{v}}(t)$ is their prediction and $\boldsymbol{\epsilon}(t)$ is the additive prediction error.

In a finite horizon approach (either shrinking or receding), at each time-step t, $\hat{\boldsymbol{v}}(t+h)$ is re-estimated (i.e., predicted again using the latest available information) for $h \in [1, 2, \cdots, n]$. We denote these re-estimations and their prediction error by $\hat{\boldsymbol{v}}_t(t+h)$ and $\boldsymbol{\epsilon}_t(t+h)$ respectively (Figure 6.1).

Similarly, their matrix form counterparts are defined as

$$\hat{\boldsymbol{V}}_t = [\hat{\boldsymbol{v}}_t(t+1)^{\mathrm{T}}\ \hat{\boldsymbol{v}}_t(t+2)^{\mathrm{T}}\ \cdots\ \hat{\boldsymbol{v}}_t(t+n)^{\mathrm{T}}]^{\mathrm{T}}$$

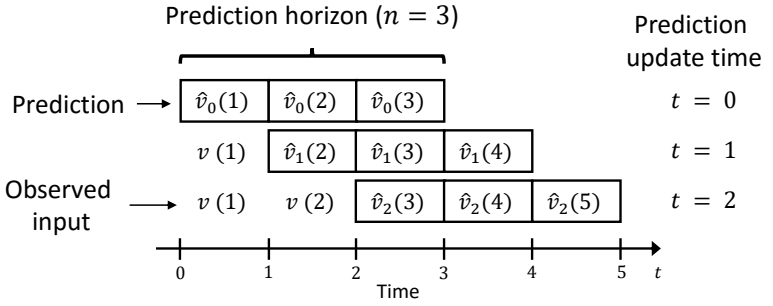


FIGURE 6.1: Prediction updates in FHC. To differentiate between predictions performed at different times, the estimation of $v(t+h)$ predicted at time-step t is denoted with $\hat{v}_t(t+h)$.

and

$$\boldsymbol{E}_t = [\boldsymbol{\epsilon}_t(t+1)^{\mathrm{T}} \ \boldsymbol{\epsilon}_t(t+2)^{\mathrm{T}} \ \cdots \ \boldsymbol{\epsilon}_t(t+n)^{\mathrm{T}}]^{\mathrm{T}}.$$

To denote all prediction errors over all control horizons,

$$\boldsymbol{E} = [\boldsymbol{E}_0^{\mathrm{T}} \ \boldsymbol{E}_1^{\mathrm{T}} \ \ldots \ \boldsymbol{E}_{N-1}^{\mathrm{T}}]^{\mathrm{T}}$$

is defined.

6.3.2 Optimal Input in Presence of Prediction Error

Theorem 6.2. *The applied controllable input $\boldsymbol{U}$, obtained from the FHC regulator, is a linear function of the accumulated exogenous inputs $\boldsymbol{Y}$ and their prediction errors $\boldsymbol{E}$,*

$$\boldsymbol{U} = \boldsymbol{\Psi}\boldsymbol{Y} + \boldsymbol{\Phi}\boldsymbol{E} \tag{6.8}$$

where

$$\boldsymbol{\Psi} = \sum_{i=0}^{N-1} \boldsymbol{\Phi}_i \mathbf{M}_Y(i), \tag{6.9}$$

$$\boldsymbol{\Phi} = [\boldsymbol{\Phi}_0 \ \boldsymbol{\Phi}_1 \ \cdots \ \boldsymbol{\Phi}_{N-1}], \tag{6.10}$$

$$\boldsymbol{\Phi}_i = \sum_{j=0}^{N-1} \mathbf{M}_\Phi(j)\mathbf{H}_j\boldsymbol{\Gamma}(j,i), \tag{6.11}$$

$$\mathbf{\Gamma}(t,\tau) = \begin{cases} t > \tau & \mathbf{0}_{kn_t \times kn_\tau} \\ t = \tau & \mathbf{I}_{kn_t \times kn_t} \\ t < \tau & \sum_{i=0}^{t-1} \mathbf{C}^\dagger \mathbf{A}^{t-i} \mathbf{B} \mathbf{M}_U(i) \mathbf{H}_i \mathbf{\Gamma}(i,\tau) \end{cases}$$

$$\mathbf{M}_\Phi(t) = [\mathbf{K}_{i,j}]_{N \times n_t}, \begin{cases} i = t+1, j = 1 & \mathbf{K}_{i,j} = \mathbf{I}_{l \times l} \\ else & \mathbf{K}_{i,j} = \mathbf{0}_{l \times l} \end{cases}$$

$$= \begin{bmatrix} \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{I} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \end{bmatrix}$$

$$\mathbf{M}_U(t) = [\mathbf{K}_{i,j}]_{n_t \times 1}, \begin{cases} i = j = 1 & \mathbf{K}_{i,j} = \mathbf{I}_{l \times l} \\ else & \mathbf{K}_{i,j} = \mathbf{0}_{l \times l} \end{cases}$$

$$= \begin{bmatrix} \mathbf{I} & \mathbf{0} & \cdots & \mathbf{0} \end{bmatrix}$$

$$\mathbf{M}_Y(t) = [\mathbf{K}_{i,j}]_{n_t \times N}, \begin{cases} j - i = t & \mathbf{K}_{i,j} = \mathbf{I}_{k \times k} \\ i = 1, j \leq t & \mathbf{K}_{i,j} = \mathbf{C}^\dagger \mathbf{A}^{t+1-j} \mathbf{C} \\ else & \mathbf{K}_{i,j} = \mathbf{0}_{k \times k} \end{cases}$$

$$= \begin{bmatrix} \mathbf{C}^\dagger \mathbf{A}^t \mathbf{C} & \cdots & \mathbf{C}^\dagger \mathbf{A}^2 \mathbf{C} & \mathbf{C}^\dagger \mathbf{A} \mathbf{C} & \mathbf{I} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I} & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{I} \end{bmatrix},$$

n_t is the length of horizon at time-step t, $\mathbf{0}_{f \times g}$ and $\mathbf{I}_{f \times g}$ are $f \times g$ zero and identity matrices respectively, and $[\mathbf{K}_{i,j}]_{f \times g}$ denotes a block matrix of $f \times g$ sub-block matrices.

Here, $\mathbf{H}_t$ is the matrix used to derive the control law, akin to $\mathbf{H}$ in (6.7), computed considering the horizon length of the optimisation at time-step t. Here, the horizon length has been denoted with n_t, as the horizon for $\mathbf{H}$ was denoted with n. This formulation allows (6.8) to be used with receding horizon, shrinking horizon, or hybrid control schemes. For a receding horizon scheme, $\mathbf{H}_t = \mathbf{H}$, while for a shrinking horizon scheme, it has to be recomputed.

Proof. The system state in the presence of prediction error is obtained by

$$\begin{aligned}\boldsymbol{X} &= \mathbf{S}_A\boldsymbol{x}(0) + \mathbf{S}_B\boldsymbol{U} + \mathbf{S}_C(\boldsymbol{V} + \boldsymbol{E}) \\ &= \mathbf{S}_B\boldsymbol{U} + \mathbf{S}_C\boldsymbol{Y} + \mathbf{S}_C\boldsymbol{E}.\end{aligned}$$

The cost function can be rewritten as

$$\begin{aligned}J &= \boldsymbol{X}^{\mathrm{T}}\bar{\mathbf{Q}}\boldsymbol{X} + \boldsymbol{U}^{\mathrm{T}}\bar{\mathbf{P}}\boldsymbol{U} \\ &= (\mathbf{S}_B\boldsymbol{U} + \mathbf{S}_C\boldsymbol{Y} + \mathbf{S}_C\boldsymbol{E})^{\mathrm{T}}\bar{\mathbf{Q}}(\mathbf{S}_B\boldsymbol{U} + \mathbf{S}_C\boldsymbol{Y} + \mathbf{S}_C\boldsymbol{E}) + \boldsymbol{U}^{\mathrm{T}}\bar{\mathbf{P}}\boldsymbol{U} \\ &= \boldsymbol{U}^{\mathrm{T}}(\bar{\mathbf{P}} + \mathbf{S}_B^{\mathrm{T}}\bar{\mathbf{Q}}\mathbf{S}_B)\boldsymbol{U} + 2\boldsymbol{U}^{\mathrm{T}}\mathbf{S}_B^{\mathrm{T}}\bar{\mathbf{Q}}\mathbf{S}_C(\boldsymbol{Y} + \boldsymbol{E}) + \\ &\quad \left(\boldsymbol{Y}^{\mathrm{T}}\mathbf{S}_C^{\mathrm{T}}\bar{\mathbf{Q}}\mathbf{S}_C\boldsymbol{Y} + 2\boldsymbol{E}^{\mathrm{T}}\mathbf{S}_C^{\mathrm{T}}\bar{\mathbf{Q}}\mathbf{S}_C\boldsymbol{Y} + \boldsymbol{E}^{\mathrm{T}}\mathbf{S}_C^{\mathrm{T}}\bar{\mathbf{Q}}\mathbf{S}_C\boldsymbol{E}\right) \\ &= \boldsymbol{U}^{\mathrm{T}}\mathbf{J}_A\boldsymbol{U} + 2\boldsymbol{U}^{\mathrm{T}}\mathbf{J}_B(\boldsymbol{Y} + \boldsymbol{E}) + \\ &\quad \left(\boldsymbol{Y}^{\mathrm{T}}\mathbf{S}_C^{\mathrm{T}}\bar{\mathbf{Q}}\mathbf{S}_C\boldsymbol{Y} + 2\boldsymbol{E}^{\mathrm{T}}\mathbf{S}_C^{\mathrm{T}}\bar{\mathbf{Q}}\mathbf{S}_C\boldsymbol{Y} + \boldsymbol{E}^{\mathrm{T}}\mathbf{S}_C^{\mathrm{T}}\bar{\mathbf{Q}}\mathbf{S}_C\boldsymbol{E}\right).\end{aligned}$$

The solution to $\underset{\boldsymbol{U}}{\operatorname{argmin}}\, J$ is given by

$$\boldsymbol{U} = \mathbf{H}(\boldsymbol{Y} + \boldsymbol{E}).$$

In finite horizon control, controllable inputs are re-evaluated at each time-step. For example, at time-step t,

$$\boldsymbol{U}_t = \mathbf{H}_t(\boldsymbol{\Upsilon}_t + \boldsymbol{E}_t) = \mathbf{H}_t\hat{\Upsilon}_t, \tag{6.12}$$

where $\boldsymbol{U}_t$ is the input optimised using information available at time t, $\boldsymbol{\Upsilon}_t \in \mathbb{R}^{n_t m}$ is constructed by accumulating previous exogenous inputs (i.e., $\boldsymbol{v}(i), i < t$) and the initial state $\boldsymbol{x}(0)$ into the first value of $\boldsymbol{Y}_t$, the tth horizon *window* of $\boldsymbol{Y}$. $\boldsymbol{Y}_t$ can be obtained by the affine mapping

$$\boldsymbol{Y}_t = \mathbf{M}_Y(t)\boldsymbol{Y}. \tag{6.13}$$

$\hat{\boldsymbol{\Upsilon}}_t$ is the estimate (i.e., prediction) of $\boldsymbol{\Upsilon}_t$ and includes the prediction error at time t, i.e., $\hat{\boldsymbol{\Upsilon}}_t = \boldsymbol{\Upsilon}_t + \boldsymbol{E}_t$.

At any time-step, $\boldsymbol{\Upsilon}_{t+1}$ can be computed by accumulating the observed values of $\boldsymbol{u}(i)$ for $i < t$, to the $\boldsymbol{Y}_{t+1}$:

$$\hat{\boldsymbol{\Upsilon}}_{t+1} = \boldsymbol{Y}_{t+1} + \boldsymbol{E}_{t+1} + \mathbf{C}^{\dagger}\sum_{i=0}^{t}\mathbf{A}^{t+1-i}\mathbf{B}\mathbf{M}_U(i)\boldsymbol{U}_i$$

Here, $\mathbf{M}_U$ is used to select the *applied* input control by separating the first input vector of $\boldsymbol{U}_t$; then $\mathbf{A}^{(t+1)-i}\mathbf{B}$ is used in a recursive approach (similar to applying $\mathbf{S}_B$) to accumulate its effect on the system's states. Finally, $\mathbf{C}^\dagger$ is applied, similar to $\mathbf{S}_C^\dagger$ in (6.5), to reshape the resulting new states into the new uncontrollable inputs vector.

The equation above can be rewritten in a recursive form by replacing $\boldsymbol{U}_i$ with (6.12),

$$\hat{\boldsymbol{\Upsilon}}_{t+1} = \boldsymbol{Y}_{t+1} + \boldsymbol{E}_{t+1} + \sum_{i=0}^{t} \mathbf{C}^\dagger \mathbf{A}^{t+1-i} \mathbf{B} \mathbf{M}_U(i) \mathbf{H}_i \hat{\boldsymbol{\Upsilon}}_i$$

which can be expanded recursively and then factored to

$$\hat{\boldsymbol{\Upsilon}}_{t+1} = \sum_{i=0}^{t+1} \boldsymbol{\Gamma}(t+1, i)(\boldsymbol{Y}_i + \boldsymbol{E}_i).$$

The final $\boldsymbol{U}$ is constructed using $\mathbf{M}_\Phi(j)$ which places the first inputs of each $\boldsymbol{U}_j$ into the jth position of $\boldsymbol{U}$:

$$\begin{aligned}
\boldsymbol{U} &= \sum_{j=0}^{N-1} \mathbf{M}_\Phi(j) \boldsymbol{U}_j \\
&= \sum_{j=0}^{N-1} \mathbf{M}_\Phi(j) \mathbf{H}_j \sum_{i=0}^{N-1} \boldsymbol{\Gamma}(j, i)(\boldsymbol{Y}_i + \boldsymbol{E}_i)
\end{aligned}$$

By factoring $\boldsymbol{\Phi}_i$ using (6.11), considering that $\mathbf{M}_U$ is non-zero only for $i = j$, and substituting $\boldsymbol{Y}_i$ by (6.13), this simplifies to

$$\begin{aligned}
\boldsymbol{U} &= \sum_{j=0}^{N-1} \sum_{i=0}^{N-1} \mathbf{M}_\Phi(j) \mathbf{H}_j \boldsymbol{\Gamma}(j, i)(\boldsymbol{Y}_i + \boldsymbol{E}_i) \\
&= \sum_{i=0}^{N-1} \boldsymbol{\Phi}_i (\boldsymbol{Y}_i + \boldsymbol{E}_i) \\
&= \sum_{i=0}^{N-1} \boldsymbol{\Phi}_i (\mathbf{M}_Y(i) \boldsymbol{Y} + \boldsymbol{E}_i),
\end{aligned}$$

which can be then reorganised by substituting for $\boldsymbol{\Psi}$ and $\boldsymbol{\Phi}$ from (6.9) and (6.10), into

$$\boldsymbol{U} = \boldsymbol{\Psi} \boldsymbol{Y} + \boldsymbol{\Phi} \boldsymbol{E}.$$

The proof is complete.

□

Notice that $\boldsymbol{\Psi}$ in (6.8) is an extended form of $\mathbf{H}$ in (6.7). In Section 6.2, $\mathbf{H}$ was designed for a constant horizon length. In contrast, $\boldsymbol{\Psi}$ supports variable horizon lengths. This is a result of (6.8)'s goal not being *control*, but an accurate simulation of what would have happened if the control was to be performed with either a receding or shrinking horizon. For shrinking only horizons (i.e., $n_{t+1} = n_t - 1$), due to the principle of optimality, $\boldsymbol{\Psi} = \mathbf{H}$.

6.3.3 Effects of Prediction Error on Cost

Theorem 6.3. *Let $\boldsymbol{U}^* = \boldsymbol{\Psi}\boldsymbol{Y}$ be the optimal input and $J^* = J(\boldsymbol{U}^*)$ the optimal (i.e., minimum) cost obtained by using a prescient forecaster. The increase in cost from optimal due to prediction error $\boldsymbol{E}$, $\Delta J = J(\boldsymbol{U}) - J^*$, is given by*

$$\Delta J = \boldsymbol{E}^{\mathrm{T}}\boldsymbol{\Theta}\boldsymbol{E} + \boldsymbol{E}^{\mathrm{T}}\boldsymbol{\Omega}\boldsymbol{Y} \tag{6.14}$$

where

$$\boldsymbol{\Theta} = \boldsymbol{\Phi}^{\mathrm{T}}\mathbf{J}_A\boldsymbol{\Phi} \tag{6.15}$$

$$\boldsymbol{\Omega} = 2\boldsymbol{\Phi}^{\mathrm{T}}(\mathbf{J}_A\boldsymbol{\Psi} + \mathbf{J}_B). \tag{6.16}$$

Proof. Let

$$\boldsymbol{U}_e = \boldsymbol{U} - \boldsymbol{U}^* = \boldsymbol{\Phi}\boldsymbol{E} \tag{6.17}$$

be the difference of the controllable input computed based on the prediction and the optimal input (i.e., without prediction error). The cost difference between minimum cost using $\boldsymbol{U}^*$ (i.e., $J(\boldsymbol{U}^*)$) and the cost with prediction error using $\boldsymbol{U}$ (i.e., $J(\boldsymbol{U})$), is given by

$$\begin{aligned}
\Delta J &= J(\boldsymbol{U}) - J(\boldsymbol{U}^*) \\
&= \boldsymbol{U}^{\mathrm{T}}\mathbf{J}_A\boldsymbol{U} + 2\boldsymbol{U}\mathbf{J}_B\boldsymbol{Y} + \mathbf{J}_{\mathbf{C}} - \boldsymbol{U}^{*\mathrm{T}}\mathbf{J}_A\boldsymbol{U}^* - 2\boldsymbol{U}^*\mathbf{J}_B\boldsymbol{Y} - \mathbf{J}_C \\
&= \boldsymbol{U}^{\mathrm{T}}\mathbf{J}_A\boldsymbol{U} + 2\boldsymbol{U}\mathbf{J}_B\boldsymbol{Y} - \boldsymbol{U}^{*\mathrm{T}}\mathbf{J}_A\boldsymbol{U}^* - 2\boldsymbol{U}^*\mathbf{J}_B\boldsymbol{Y}.
\end{aligned}$$

Expanding $\boldsymbol{U} = \boldsymbol{U}^* + \boldsymbol{U}_e$ results in

$$\begin{aligned}\Delta J &= (\boldsymbol{U}^* + \boldsymbol{U}_e)^{\mathrm{T}}\mathbf{J}_A(\boldsymbol{U}^* + \boldsymbol{U}_e) - \boldsymbol{U}^{*\mathrm{T}}\mathbf{J}_A\boldsymbol{U}^* \\ &\quad + 2(\boldsymbol{U}^* + \boldsymbol{U}_e)^{\mathrm{T}}\mathbf{J}_B\boldsymbol{Y} - 2\boldsymbol{U}^{*\mathrm{T}}\mathbf{J}_B\boldsymbol{Y} \\ &= \boldsymbol{U}^{*\mathrm{T}}\mathbf{J}_A\boldsymbol{U}^{\mathrm{T}} + \boldsymbol{U}_e^{\mathrm{T}}\mathbf{J}_A\boldsymbol{U}_e + 2\boldsymbol{U}_e^{\mathrm{T}}\mathbf{J}_A\boldsymbol{U}^* - \boldsymbol{U}^{*\mathrm{T}}\mathbf{J}_A\boldsymbol{U}^* \\ &\quad + 2(\boldsymbol{U}_e + \boldsymbol{U}^*)^{\mathrm{T}}\mathbf{J}_B\boldsymbol{Y} - 2\boldsymbol{U}^{*\mathrm{T}}\mathbf{J}_B\boldsymbol{Y} \\ &= \boldsymbol{U}_e^{\mathrm{T}}\mathbf{J}_A\boldsymbol{U}_e + 2\boldsymbol{U}_e^{\mathrm{T}}\mathbf{J}_A\boldsymbol{U}^* + 2\boldsymbol{U}_e^{\mathrm{T}}\mathbf{J}_B\boldsymbol{Y}.\end{aligned}$$

By replacing $\boldsymbol{U}^*$ with its definition from (6.8) (i.e., $\boldsymbol{U}^* = \boldsymbol{\Psi}\boldsymbol{Y}$ considering $\boldsymbol{E} = 0$), we can simplify the above to

$$\begin{aligned}\Delta J &= \boldsymbol{U}_e^{\mathrm{T}}\mathbf{J}_A\boldsymbol{U}_e + 2\boldsymbol{U}_e^{\mathrm{T}}\mathbf{J}_A(\boldsymbol{\Psi}\boldsymbol{Y}) + 2\boldsymbol{U}_e^{\mathrm{T}}\mathbf{J}_B\boldsymbol{Y} \\ &= \boldsymbol{U}_e^{\mathrm{T}}\mathbf{J}_A\boldsymbol{U}_e + 2\boldsymbol{U}_e^{\mathrm{T}}(\mathbf{J}_A\boldsymbol{\Psi} + \mathbf{J}_B)\boldsymbol{Y}.\end{aligned}$$

Substituting $\boldsymbol{U}_e$ with (6.17) results in

$$\Delta J = \boldsymbol{E}^{\mathrm{T}}(\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{J}_A\boldsymbol{\Phi})\boldsymbol{E} + 2\boldsymbol{E}^{\mathrm{T}}\boldsymbol{\Phi}^{\mathrm{T}}(\mathbf{J}_A\boldsymbol{\Psi} + \mathbf{J}_B)\boldsymbol{Y}.$$

This can be factored using (6.15) and (6.16) to

$$\Delta J = \boldsymbol{E}^{\mathrm{T}}\boldsymbol{\Theta}\boldsymbol{E} + \boldsymbol{E}^{\mathrm{T}}\boldsymbol{\Omega}\boldsymbol{Y}.$$

The proof is complete. □

According to the principle of optimality [8], $\boldsymbol{U}^* = \mathbf{H}\boldsymbol{Y}$ is universal for all steps in a shrinking horizon control, and as a result $\boldsymbol{\Psi} = \mathbf{H}$. Consequently, for an approach with only shrinking horizons, ΔJ simplifies to

$$\Delta J = \boldsymbol{E}^{\mathrm{T}}\boldsymbol{\Theta}\boldsymbol{E}.$$

6.4 Time-series Prediction and Model Selection

The objective of time-series prediction is to find a function $p(\cdot)$ to estimate the future of time-series $v(t)$ over prediction horizon $h \in [1, 2, \cdots, n]$ using available data, i.e.,

$$\hat{v}_t(t+h) = p(v(t), v(t-1), v(t-2), \ldots).$$

As explained in Chapter 5, forecasting techniques are divided into two phases:

1. A learning phase, where for a chosen model and its hyper-parameters, the training data is fitted to minimise a p-norm error (i.e., $||v(t) - \hat{v}_t(t)||_p$). Commonly, an L_2 norm is used as error, as in the cases of ordinary least squares (OLS) regression, perceptron neural networks [219], and support vector regression (SVR) [220].
2. A cross-validation or backtesting phase, where performance of different models and hyper-parameters across a separate set of testing data are compared.

The first phase implicitly minimises $||\boldsymbol{E}||_p$, which also reduces the controller's error. However, the second phase is commonly implemented using ordinary error measures which utilise an identity matrix instead of $\boldsymbol{\Theta}$ (e.g., MSE is $\boldsymbol{E}^{\mathrm{T}}\boldsymbol{E}$), and thus ignore possible dependencies between errors and also between the errors and the inputs. To include this knowledge of the controller, a *step-by-step simulation* of FHC has to be performed: for each discrete time-step, the control law is first applied based on predictions, and then the system states and costs are updated according to the actual inputs. Equivalently, (6.14) can be used to evaluate the final cost of prediction error in a single step.

6.5 Cost Matrix Dimensionality Reduction

Computing ΔJ using (6.14) is of $O(N^2)$ time complexity, compared to $O(N)$ of using MSE and MAE. Consequently, the efficiency of computing ΔJ has to be improved before being used in data-intensive problems.

In many real-world problems, matrices $\boldsymbol{\Theta}$ and $\boldsymbol{\Omega}$ prove to be sparse, or even diagonal. In such cases, numerical techniques can be used to improve efficiency of computing (6.14).

In other cases, assuming repeated evaluation of (6.14) on a fixed system, one can precompute $\boldsymbol{\Theta}$ and $\boldsymbol{\Omega Y}$, and calculate ΔJ efficiently for different values of $\boldsymbol{E}$, the latter coming from different prediction models.

With this assumption, matrix decomposition can also be used to further reduce computation complexity by approximating matrix $\boldsymbol{\Theta} \in \mathbb{R}^{M \times M}$ with a matrix of lower rank [221]. Here, M is the total number of predictions and is obtained from the sum of horizon lengths and size of the inputs vector, i.e.,

$$M = m \sum_i n_i.$$

Let $\boldsymbol{\Theta} = \boldsymbol{V}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathrm{T}}$, where $\boldsymbol{\Sigma}$ is the diagonal matrix of eigenvalues and $\boldsymbol{V}$ is the matrix of eigenvectors of $\boldsymbol{\Theta}$. By only keeping the $0 < L < M$ largest eigenvalues of $\boldsymbol{\Theta}$ and their

corresponding eigenvectors, $\boldsymbol{\Theta}$ can be approximated with

$$\boldsymbol{\Theta}' = \boldsymbol{V}'\boldsymbol{\Sigma}'\boldsymbol{V}'^{\mathrm{T}},$$

where $\boldsymbol{V}' \in \mathbb{R}^{M\times L}$ and $\boldsymbol{\Sigma}' \in \mathbb{R}^{L\times L}$.

Let $\mathbf{W} = \boldsymbol{V}'\sqrt{\boldsymbol{\Sigma}'}$, where $\sqrt{\boldsymbol{\Sigma}'}$ computes the root square of the diagonal eigenvalue matrix. An approximation to (6.14) is

$$\Delta J' = \boldsymbol{E}^{\mathrm{T}}\mathbf{W}'\mathbf{W}'^{\mathrm{T}}\boldsymbol{E} + \boldsymbol{E}^{\mathrm{T}}\boldsymbol{\Omega}\boldsymbol{Y}. \tag{6.18}$$

Exploiting the symmetric structure of the new cost function and assuming precomputed $\mathbf{W}'$ and $\boldsymbol{\Omega}\boldsymbol{Y}$, evaluating (6.18) is reduced to time complexity $\mathrm{O}(ML)$.

The choice of L is problem dependent. A general guideline is to select L such that $\mathrm{tr}(\boldsymbol{\Sigma}') \geq \lambda \mathrm{tr}(\boldsymbol{\Sigma})$, where tr is the trace operator and $0 \leq \lambda \leq 1$ determines how much of the matrix's *energy* is to be conserved. In practice, $\lambda > 0.99$ is commonly used.

6.6 Numerical Examples and Simulation Results

In this section, two finite horizon problems with time series forecasting are analysed, and the proposed error measure ΔJ is compared with MSE for predictor selection. MSE was chosen as it offers a quadratic error function, similar to the controller's cost function, and is not scaled against the magnitude of inputs.

For each problem, first the system dynamics are defined and used to formulate an FHC control problem. The best predictor models are then selected over a set of *training* data using different error measures. Consequently, a simulation is performed over a separate set of *testing* data, and performance results of models selected using ΔJ and MSE are compared.

6.6.1 Pre-ordering Problem

Inventory management and supply chain planning techniques play an essential role in managing supply and demand, and are widely studied and used in practice [222]. Recent developments in this regard have shown that forward-looking optimisation-based policies, such as using optimal control in combination with forecasting, significantly outperform other rule-based decision policies [215, 223].

In this example, we study the problem of meeting a fluctuating demand for a perishable commodity, similar to the problem discussed in [216]. In this task, one can either pre-order the perishable item with different lead times and discounts, or buy it on the spot market at a higher price. The objective is to minimise the ordering costs by utilising prediction.

6.6.1.1 Problem Formulation

We formalise this problem as follows:

1. The demand is denoted with $v(t) \in \mathbb{R}^+$, and the spot price is depicted with p.
2. It is possible to pre-order κ steps ahead, where at each time-step a discount of d is applied.
3. A pre-order can be adjusted at any time before delivery; however, an adjustment penalty equal to the discounted price is applied. Similarly, a penalty is applied for over-supplied orders (i.e., discarded perishables).
4. Pre-orders are denoted with $\boldsymbol{u}(t) = [u_{t+1}(t) \ \cdots \ u_{t+\kappa-1}(t) \ u_{t+\kappa}(t)]^{\mathrm{T}}$, where $u_{t+h}(t)$ is the order (or adjustment to the order) at time t for delivery at time $t+h$.
5. An order book is maintained, in form of the vector $\boldsymbol{x}(t) = [x_t(t) \ x_{t+1}(t) \ \cdots \ x_{t+\kappa}(t)]^{\mathrm{T}}$, where $x_{t+h}(t)$ is the total of orders expected at time t to be delivered at time $t+h$.

Notice that $\boldsymbol{u}(t) \in \mathbb{R}^{\kappa}$, and $\boldsymbol{x}(t) \in \mathbb{R}^{\kappa+1}$ as it includes a state for the *delivery* at time-step t in addition to future pre-orders.

The problem can be formulated using the state-space equation (6.1), as updating the order book using

$$x_{t+h}(t+1) = x_{t+h}(t) + u_{t+h}(t)$$

when pre-ordering (i.e., $h > 1$), and

$$x_{t+1}(t+1) = x_{t+1}(t) - v(t)$$

during delivery.

Assuming $\kappa = 3$, the system dynamics in matrix form are

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

$$\mathbf{B} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

and

$$\mathbf{C} = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

Eq. (6.2) is used as the cost function, with the stage cost of $\boldsymbol{u}(t)$ (i.e., pre-ordering prices and discounts) defined as

$$\mathbf{P}_t = \begin{bmatrix} d^3p & 0 & 0 \\ 0 & d^2p & 0 \\ 0 & 0 & dp \end{bmatrix}$$

and the stage cost of $\boldsymbol{x}(t)$ (i.e., the penalty for unmet demand) as

$$\mathbf{Q}_t = \begin{bmatrix} p & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

For numeric simulation, the spot price was set to $p = 4$, and the discount to $d = 0.7$.

6.6.1.2 Analysis of the Cost Equation

An analysis of $\boldsymbol{\Theta}$ and $\boldsymbol{\Omega}$ from (6.14) for the current problem reveals the matrix elements associated with prediction of the current demand, and prediction of steps more than κ step ahead, are zero. The former is a result of the observed demand (and not the prediction) being used for spot market ordering. The latter is because any prediction beyond κ steps is not used for ordering in the current horizon. Consequently, choosing a horizon length beyond the number of pre-orders (such as with the methodology used in [216]) is redundant.

Furthermore, $\boldsymbol{\Theta}$ is a diagonal matrix for optimisations ending with shrinking horizons (e.g., as in Figure 6.2). In addition, the diagonal elements for each set of predictions

FIGURE 6.2: A hybrid receding-shrinking horizon control scheme, with a maximum horizon length of 3.

decay exponentially. For example, for $\kappa = 3$ and a horizon of $n = 5$, the block of $\mathbf{\Theta}$ associated with the first vector of predictions is

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 2.98 & 0 & 0 & 0 \\ 0 & 0 & 0.35 & 0 & 0 \\ 0 & 0 & 0 & 0.13 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

This simplifies ΔJ to a weighted MSE.

6.6.1.3 Simulation and Results

The simulation was set up to run for 10 time-steps, with pre-ordering allowed for three steps ahead (i.e., $N = 10$ and $\kappa = 3$). Based on the analysis of $\mathbf{\Theta}$ in Section 6.6.1.2, the control horizon n was limited to the number of pre-orders κ (i.e., $n_t \leq \kappa = 3, \forall t$). A finite horizon controller governs the system, with seven receding windows of length 3, followed by three shrinking windows (Figure 6.2).

The hth step in the future, $v(t+h)$, was predicted using an AR model,

$$\hat{v}_t(t+h) = \sum_{i=0}^{q} \phi_i v(t-i), \tag{6.19}$$

where q is the model order, and ϕ_i are the model parameters. For each future step in a horizon, a different model was selected and fitted to data using ordinary least squares, resulting in a total of three models. The range of orders was limited to $q \in [1, 2, \ldots, 8]$.

To simulate real-world demand, time-series from the M3 competition dataset [224] were used. Only time-series longer than 100 samples were selected for simulation. For each time-series, the first 80 values were assigned to in-sample testing, with the starting 60 time-steps used for training the predictor, and the next 20 values for model validation. The last 20 were used to report out-of-sample prediction errors, as well as the controller's cost performance, i.e., the out-of-sample ΔJ.

Three different approaches were used for model order selection using cross-validation:

1. A *random search* was performed to select the best model using in-sample MSE. A subset of the models was sampled from the model space using a uniform distribution, and the model with the least cross-validation error was selected.
2. A random search using in-sample ΔJ was performed.
3. A *hybrid search* was undertaken. The best model for the first step (i.e., $h = 1$) was chosen using in-sample MSE through an exhaustive search. The model orders for $h = 2$ and $h = 3$ were selected using in-sample ΔJ through a random search.

The first two methods compare MSE and ΔJ model selection when the computational resources are limited. The design rationale of the third technique is based on the $\mathbf{\Theta}$ weights, where the first step contributes 84% of the total error. Hence, half of the computational capacity is solely allocated to the first predictor's model selection.

For a fair comparison, only 16 model evaluations were allowed per each approach. For the hybrid approach, this translated to 8 evaluations exhaustively searching for the best first model order, and 8 additional random evaluations for the second and third model orders.

The test was repeated for each of the 1020 selected time-series. As each time-series exhibits different mean-variance characteristics, the resulting MSE and ΔJ are not directly comparable. Consequently, these errors were normalised to the MSE and ΔJ of a *naïve* predictor respectively, where the predictor simply repeats the last observed value, i.e., $\hat{v}_t(t+h) = v(t), \forall h$. The normalised results were then averaged and reported.

TABLE 6.1: Run-time and speed-up comparison for the pre-ordering problem.

Measurement Technique	Run-time (s)	Speed-up
Step-by-step simulation	2340.7	1
Closed-form ΔJ using (6.14)	11.730	199.5
Diagonal ΔJ	0.9130	2563.7
MSE	0.8940	2619.0

TABLE 6.2: Mean normalised error for different model selection methods in the pre-ordering problem. Values within parentheses denote standard variation.

Selection Method	In-sample MSE	In-sample ΔJ	Out-of-sample MSE	Cost Performance
Random Search using MSE	**0.9078** (± 0.4497)	0.8953 (± 0.4044)	**0.9261** (± 0.4562)	0.9208 (± 0.4338)
Random Search using ΔJ	0.937 (± 0.4668)	0.867 (± 0.3922)	0.941 (± 0.4745)	0.9109 (± 0.414)
Hybrid Search based on ΔJ	0.9081 (± 0.4515)	**0.8533** (± 0.3851)	0.9321 (± 0.4704)	**0.9072** (± 0.4117)

To reduce the computation time of ΔJ, considering the diagonal nature of $\mathbf{\Theta}$, (6.14) was also numerically implemented as a weighted MSE. Table 6.1 compares the run-times of different methods. It is observed that computing ΔJ using (6.14), even in its full matrix form, offers a significant speed-up compared to a step-by-step FHC simulation. Furthermore, the diagonal only implementation is almost as efficient as an ordinary MSE, achieving a speed-up of more than 2500×.

In Table 6.2, prediction errors for models selected using MSE and ΔJ are summarised. While the models selected using MSE offered a better in-sample and out-of-sample MSE, they were outperformed in the cost performance by the models selected using ΔJ. Overall, a cost improvement of 1.5% was obtained simply by prioritising the first step's model selection over other steps. A paired sample t-test of results rejected the null hypothesis of improvement not being significant with p-value of 0.0027.

Considering the independence of the three model orders and their errors, the best global model would have minimised both the MSE and ΔJ. This example demonstrates that when a global search is not possible (e.g., limited computational resources), the search can be concentrated on the most influential factor, as analysed by the proposed cost measure, to improve model selection.

6.6.2 Stock Portfolio Management

We extend the example of Section 6.6.1 to financial markets, where a dealer keeps a portfolio of stocks to trade on behalf of his or her clients. The dealer wishes to reduce the costs associated with:

1. the market risk, i.e., the loss of portfolio value due to market price changes, and
2. trading with other dealers, when the client's requests can not be fulfilled using what is available in the portfolio.

The first issue forces the dealer to minimise the inventory to avoid risk, while the second obligates keeping all client trades in the portfolio, such that opposing client trades (i.e., buys and sells) are neutralised without referring to other dealers.

The problem of finding the optimal portfolio subject to cost and risk considerations has been extensively researched. Recent studies of this problem for stock options and FX market, using MPC but neglecting the effects of time-series prediction error, were discussed in Section 2.5.2.

6.6.2.1 Problem Formulation

We simplify the problem by assuming a single-stock inventory with the following rules and notations:

1. $x(t)$ denotes dealer's inventory.
2. The demand is denoted with $v(t)$.
3. The inter-dealer trades are determined using $u(t)$.
4. Short-selling is allowed.
5. To consider market impact, the inter-dealer brokering cost is modelled using a quadratic function of trades, $P_t u^2(t)$.
6. The risk is modelled using $Q_t x^2(t)$, where Q_t is the market volatility, i.e., the variance of the price process as used in modern portfolio optimisation [61].

The dealer's dynamics can be formulated using the state-space equation (6.1), with $A = 1$, $B = 1$, and $C = 1$. The dealer begins with a zero-balance, i.e., $x(0) = 0$. To concentrate on demand prediction, we assume the cost of trading with inter-dealer brokers and the market volatility are available and constant in time, $P_t = 1$ and $Q_t = 1$ respectively. The objective is to minimise the overall cost, as defined by (6.2).

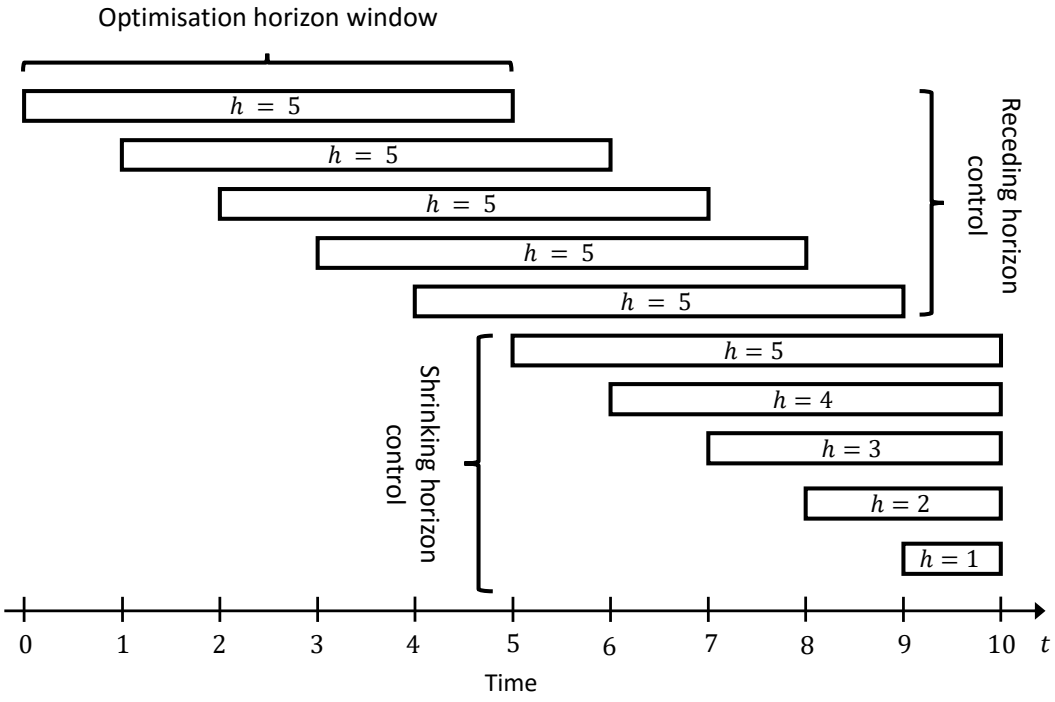


FIGURE 6.3: A hybrid receding-shrinking horizon control scheme, with a maximum horizon length of 5.

6.6.2.2 Simulation and Results

Similar to the controller used in Section 6.6.1, a receding/shrinking horizon controller was designed to govern the system for 10 time-steps. The controller uses five receding windows of length 5 followed by five windows with shrinking lengths (Figure 6.3).

For predicting v_t, five different linear models were used to predict each future time-step, using the AR model defined in (6.19). An exhaustive search was performed to cross-validate the model orders $q \in [2, 3, \ldots, 8]$, using MSE and ΔJ.

The clients' trades were simulated via a 5th order auto-regressive (AR) model,

$$v_{t+1} = 2.76v_t - 3.13v_{t-1} + 1.79v_{t-2} - 0.50v_{t-3} + 0.05v_{t-4} + \epsilon_t \tag{6.20}$$

where $\epsilon_t \sim N(0, 1)$ is a zero-mean unit-variance Gaussian noise.

For each test, a time-series of length 100 was generated. The first 80 values were assigned to in-sample testing, with the starting 60 time-steps used for training the predictor, and the next 20 values for model validation. The last 20 were used to report out-of-sample prediction errors, as well as the controller's cost performance, i.e, the out-of-sample ΔJ. The test was repeated 10 times.

TABLE 6.3: Run-time and accuracy comparison for the stock portfolio management problem.

Measurement Technique	Run-time (s)	Speed-up	Measured ΔJ
Step-by-step simulation	1427.664	1	96.8476
Closed-form ΔJ using (6.14)	12.136	117.6	96.8476
Approximated ΔJ using (6.18)	2.826	598.1	96.8476
MSE	1.228	1376.0	N/A

TABLE 6.4: Mean prediction error for different model selection methods in the stock portfolio management problem. Values within parentheses denote standard variation.

Measure for selection	In-sample MSE	In-sample ΔJ	Out-of-sample MSE	Cost Performance
MSE	**32.4888** (± 15.0229)	86.9027 (± 25.7024)	**36.3447** (± 12.3691)	98.1846 (± 34.2707)
ΔJ	32.4892 (± 15.0266)	**86.7015** (± 26.4899)	38.0916 (± 13.0867)	**94.7870** (± 35.3964)

Our first concern is the dimensionality reduction for $\mathbf{\Theta} \in \mathbb{R}^{40\times 40}$. Despite its simpler formulation compared to the problem of Section 6.6.1, $\mathbf{\Theta}$ is not diagonal, and consequently the eigenvalue method has to be used. The 10 largest eigenvalues hold more than 99.9% of eigenvalue energy, and thus a fourfold order reduction is possible using $\mathbf{W}' \in \mathbb{R}^{40\times 10}$. Table 6.3 compares the time required for computing each error measure and the value of errors. It is observed the proposed dimensionality reduction technique results in near 5× speed-up, with less than 0.001% loss of accuracy. Comparing run-times of Table 6.3, an overall 598× speed-up over a step-by-step FHC simulation is observed. While MSE is still 2.3× faster, the difference in execution is 1.6 s, which is negligible compared to the time spent for training the model.

Figure 6.4 compares in-sample MSE error against ΔJ for different model orders. It is evident that these errors are not strongly correlated. Numerical value of the correlation coefficient for MSE and ΔJ was measured to be 0.423, compared to that of MSE and MAE being 0.893. As a result, model selection using MSE does not necessarily improve ΔJ.

Performance results for the selected models are summarised in Table 6.4. It can be seen that using ΔJ as a model selection measure has reduced the controller's cost compared to selections using the standard MSE measure. While this reduction is not significant for in-sample ΔJ results, the controller's final cost improvement, from 98.18 to 94.79, is considerable. Additionally, a paired sample t-test rejected the null hypothesis of improvement not being significant with $p = 0.009409$.

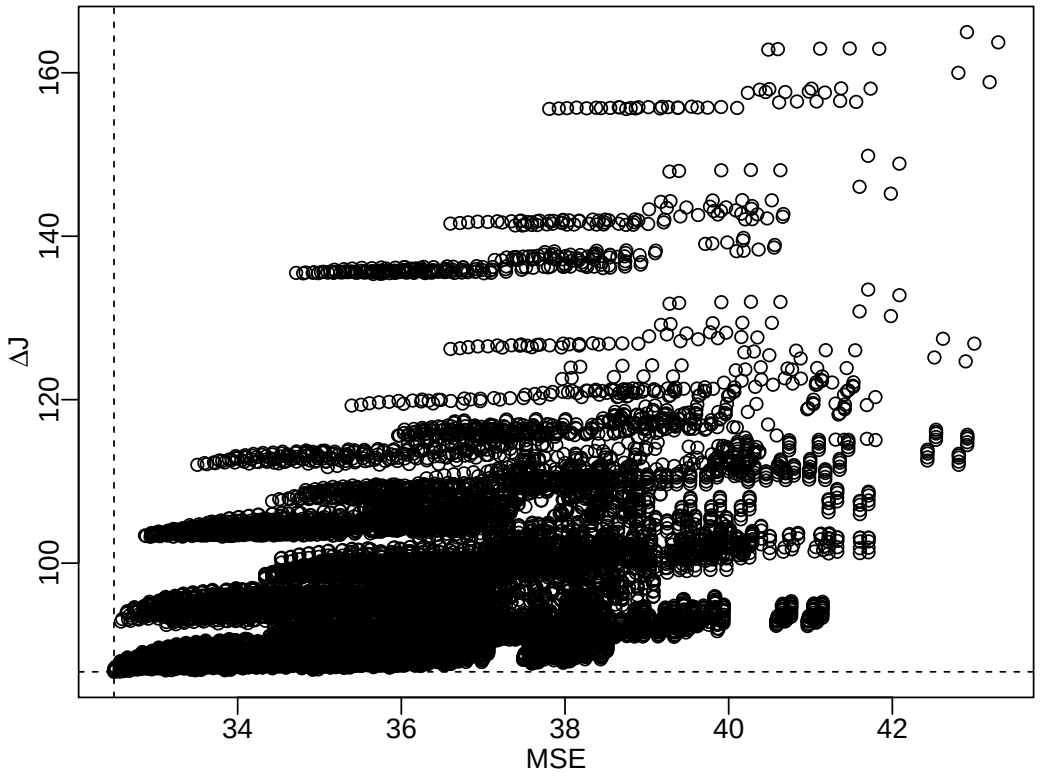


FIGURE 6.4: MSE versus ΔJ for different model orders in the stocks portfolio problem.

6.6.2.3 Dimensionality Reduction and Accuracy

To test the effects of dimensionality reduction on the accuracy of $\Delta J'$, the number of chosen eigenvalues of $\boldsymbol{\Theta}$ (i.e., L) was varied and the cost performance was measured using (6.18). Only prediction errors from one AR model with $q = 4$ for all horizons were used. The results, including the energy of the remaining eigenvalues (λ), run-time and speed-up of (6.18) against using the full rank $\boldsymbol{\Theta}$ matrix, the measured value of $\Delta J'$, and the accuracy against ΔJ defined as

$$\text{Accuracy } (\%) = \frac{\Delta J'}{\Delta J} \times 100,$$

are shown in Table 6.5.

It can be seen that as expected, increasing L improves accuracy at the cost of run-time. Additionally, selecting the smallest L where $\lambda > 0.99$ (as proposed in Section 6.5 and performed for the stocks portfolio example) results in near 100% accuracy, while enabling the highest possible speed-up.

TABLE 6.5: Effects of dimensionality reduction on ΔJ accuracy and run-time.

L	λ	Run-time (s)	Speed-up	$\Delta J'$	Accuracy (%)
2	0.216	2.77	3.650	32.53	37.54
3	0.324	2.83	3.572	39.70	45.81
4	0.432	2.97	3.404	48.53	56.00
5	0.540	3.03	3.337	54.28	62.64
6	0.649	3.15	3.210	63.32	73.07
7	0.756	3.27	3.092	75.58	87.22
8	0.861	3.36	3.009	82.48	95.19
9	0.954	3.46	2.922	86.27	99.56
10	1.000	3.62	2.793	86.65	100.00
20	1.000	4.75	2.128	86.65	100.00
40	1.000	10.11	1.000	86.65	100.00

6.7 Application to the FX Risk Management Problem

As noted in this chapter's introduction, and similar to the stock portfolio problem studied in Section 6.6.2, the FX risk management system proposed in Chapter 3 uses an LQ formulation:

- Hedging state-space equation (3.1) is a specific case of linear state-space model (6.1), with $A = 1$, $B = 1$, and $C = 1$.
- Hedging cost is a quadratic function of the current state (i.e., positions) and the controllable inputs (i.e., hedging actions).

Parameters of the cost function, including bid-ask spread and volatility, are determined by predicting the market conditions. These parameters are deeply explored in financial literature and well established techniques exist to model and forecast them using the available market data [225, 226].

Client flow data, on the other hand, is considered confidential by financial institutions and has been subject to fewer studies. Forecasting the client flow, which is represented by $\boldsymbol{f}$ in (3.1), and $\boldsymbol{v}$ in (6.1), is possible using the techniques introduced in Chapter 5. Assuming that the constraints on positions and hedging actions are set high enough not to influence the hedging, one can use the proposed error measure (6.14) in cross-validation. This improves the accuracy compared to using common error measures by precisely replicating the final hedging results, while allowing the predictors to be evolved with a faster speed compared to a complete backtesting.

6.7.1 Numerical Examples

In this section, three numeric examples are presented to show the distribution of error weights for different hedging objectives. It is assumed that the hedging has three steps (i.e., $N = 3$), starting at $t = 1$ and ending at $t = 3$. Consequently, testing the client forecaster algorithm requires six forecasts, i.e.,

$$\boldsymbol{v} = [\hat{f}_1(1)\ \hat{f}_1\ (2)\ \hat{f}_1(3) \quad \hat{f}_2(2)\ \hat{f}_2(3) \quad \hat{f}_3(3)].$$

Here, $\hat{f}_\tau(n)$ denotes the forecast of the client flow $f(n)$ at time $t = \tau$.

In the first case, minimum risk optimisation is studied with $P_t = 0$ and $Q_t = \sigma$ (i.e., the bid-ask spreads are set to 0 and the volatility is set to σ). Using (6.15) and (6.16), and letting $\sigma = 1$, results in $\boldsymbol{\Omega} = \mathbf{0}$, and

$$\boldsymbol{\Theta} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

The sparse $\boldsymbol{\Theta}$ has 1's on its diagonal, relevant to the locations of the first time-step of each horizon. In other words, only the errors for $\hat{f}_1(1)$, $\hat{f}_2(2)$, and $\hat{f}_3(3)$ are considered in the final ΔJ. This was expected from the analysis of Section 3.4.1, where it was shown that the best strategy for minimising the risk is closing all of the current position. This strategy only considers the current client flow and does not use the information from future steps.

In the second case, minimum cost hedging is considered by using $Q_t = 0,\ t < N$, which dismisses the risk of holding position $x(t)$. The transaction costs, and also the end-of-day closing are modelled by $P_t = \delta$ and $Q_N = \delta$ respectively, where δ is the market impact

coefficient. Letting $\delta = 1$ results in $\mathbf{\Omega} = \mathbf{0}$, and

$$\mathbf{\Theta} = \begin{bmatrix} 0.083 & 0.083 & 0.083 & 0 & 0 & 0 \\ 0.083 & 0.083 & 0.083 & 0 & 0 & 0 \\ 0.083 & 0.083 & 0.083 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.167 & 0.167 & 0 \\ 0 & 0 & 0 & 0.167 & 0.167 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.5 \end{bmatrix}.$$

In this case, the error weights are equal in each horizon (i.e., each block in the matrix), but they increase towards the end-of-day. This is because, as analysed in Section 3.4.4 for $\lambda = 0$, the trades are based on the *total* estimation of the future flow, and divided by the number of available hedging time-steps. Therefore, in each block, where there is no difference between individual forecasts at that time, the weights are equal. As the number of available hedging steps decreases, the controller has less time to compensate for forecast errors in future actions, and as a result the impact of errors is intensified. It can be shown that the error weight for each element is $w = \frac{1}{(n)(n+1)}$, where $n = N - t + 1$ is the number of hedging actions available from the current block. For example for $t = 1$, three hedging actions remain, and thus $w = \frac{1}{3\times 4} = 0.0833$. For $t = 3$, only one final hedging action can be applied, hence $w = \frac{1}{1\times 2} = 0.5$. In Appendix B, using sensitivity analysis of the cost function, an analytical model for this case is derived, yielding the same results.

For the third example, both cost and risk are minimised with $\delta = 1$ and $\sigma = 1$ by letting $P_t = Q_t = 1$, resulting in $\mathbf{\Omega} = \mathbf{0}$, and

$$\mathbf{\Theta} = \begin{bmatrix} 0.985 & 0.369 & 0.123 & 0 & 0 & 0 \\ 0.369 & 0.138 & 0.046 & 0 & 0 & 0 \\ 0.123 & 0.046 & 0.015 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.9 & 0.3 & 0 \\ 0 & 0 & 0 & 0.3 & 0.1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.5 \end{bmatrix}.$$

In this case, analogous to the examples of Section 6.6, at each time-step (i.e., a forecast block), the errors for the near future forecasts are more influential than later forecasts.

This knowledge can be used to allocate the computational effort to obtain more accurate forecasters for near term prediction, to which the LQ controller is more sensitive. This

technique was previously discussed and proven in the inventory management example in Section 6.6.1.

6.8 Summary

This chapter presented several observations regarding error measures for an LQ finite horizon controller in presence of a predictable exogenous input:

- First, an exact closed-form solution for ΔJ, i.e., the increase in cost function of a discrete LQ FHC system due to forecast error, was derived. It was shown that this increase follows a quadratic form.
- Second, it was demonstrated that by using dimensionality reduction, computational complexity of ΔJ can be reduced with minimal loss of accuracy. This leads to a significant speed-up in the case where different prediction algorithms and parameters need to be tested for fixed system parameters.
- Third, it was shown that using ΔJ to select forecasting models results in better predictors compared to common time-series error measures, such as MSE.

In addition to two benchmark problems, ΔJ was also analysed in the context of FX hedging, and it was shown that this measure can be used to quantify the effects of forecast error on hedging costs.

Chapter 7

FX Hedging Results

7.1 Introduction

In Chapter 3, a new model for hedging FX risk was established. In this chapter, this model is backtested using both synthetic and historical data.

To test the model independent of forecasting techniques, the scenario based approach explained in Section 3.6.3 is used. First, several statistical models are introduced which are used for creating synthetic data, as well as describing and modelling the real-world observations. A scenario generating oracle is then defined to simulate non-perfect predictive modelling with a controllable degree of accuracy.

Consequently, to improve hedging risk-cost profiles, the time-series forecasting methodology proposed in Chapter 5 is utilised to forecast the client flow, and comparisons are performed against stochastic models and other time-series forecasting techniques.

7.2 Data Models and Scenario Generation

As noted in Section 3.6.3, each hedging scenario consists of three correlated components:

- FX rate returns $\mathbf{R}$
- client flow volume $\mathbf{F}$
- market impact coefficients $\boldsymbol{\delta}$

In a multi-currency model, every scenario must include all of these components for each currency. In this section, stochastic models are introduced for each component to be used for scenario generation, as well as serving as benchmarks for forecasting the future behaviour of the clients and market.

7.2.1 FX Rate and Volatility Model

The FX logarithmic price process can be expressed as a continuous-time jump-diffusion process [84], formalised by

$$dp(\tau) = \mu_p(\tau)d\tau + v(\tau)dW(\tau) + k(\tau)dq(\tau), \tag{7.1}$$

where $p(\tau)$ is the logarithm of the FX rate for the continuous time τ, $\mu_p(\tau)$ (the drift coefficient) is risk free interest, $W(\tau)$ is the Wiener process, $v(\tau)$ is the diffusion coefficient (the volatility), $k(\tau)$ measures the jumps' intensity, and $q(\tau)$ is a counting process which is determined by the scheduled time of macroeconomic announcements. Here, we assume $\mu_p \approx 0$ as the optimisation horizon is too short for the interest rate to be effective.

Figure 7.1 shows an example of different FX rate scenarios generated for an event occurring at 12:00.

We model the discrete-time logarithmic returns $r(t)$ using

$$r(t) = p(t\Delta\tau) - p\left((t-1)\Delta\tau\right) \tag{7.2}$$

where $\Delta\tau$ is the time-step used in optimisation.

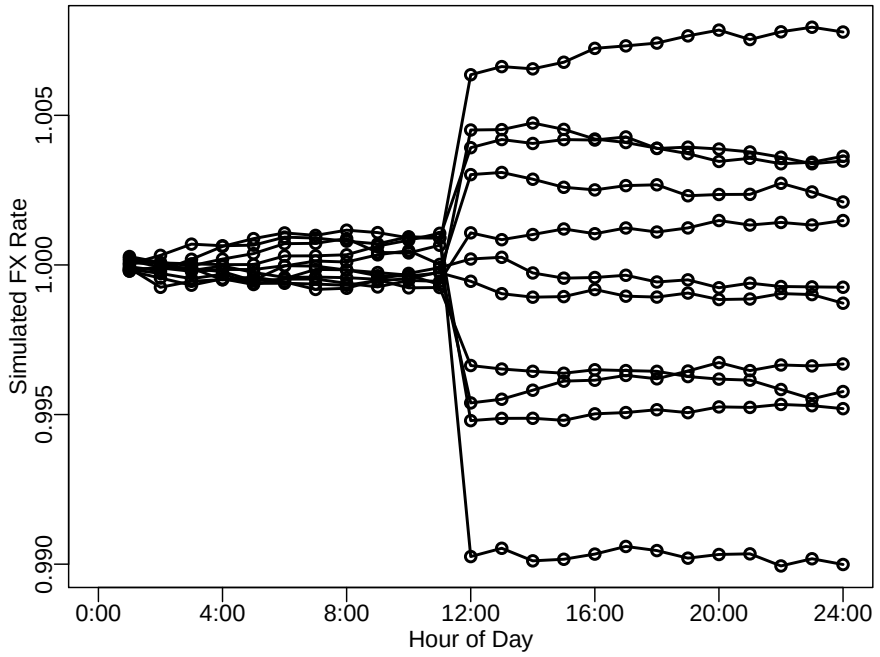


FIGURE 7.1: Simulated FX rate scenarios with an announced event at 12:00 PM.

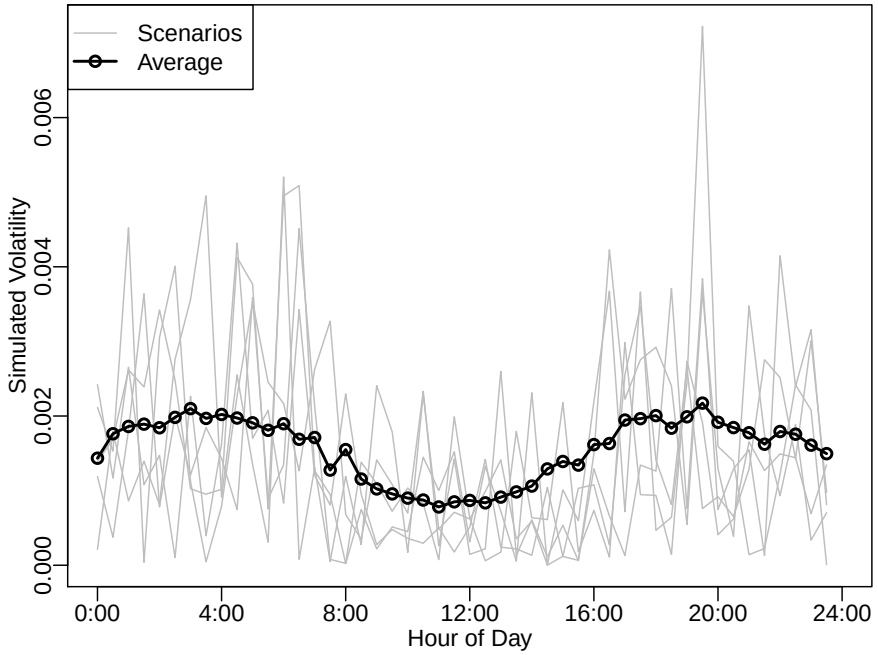


FIGURE 7.2: Simulated M-shaped volatility during the day.

The discrete-time form of volatility, $v(t)$, is modelled using an M-shaped pattern to account for daily pattern of liquidity variation [87]:

$$v(t) = \nu \left(1 + \frac{\rho - 1}{2}\left(1 + cos\left(2\pi \frac{t + \frac{N-w}{2} - t_{min}}{N - w}\right)\right)\right) \tag{7.3}$$

Here, ν is the minimum daily volatility, ρ is the ratio of maximum daily volatility to its minimum, N is the number of time-steps in the day, t_{min} is the time of volatility minima (i.e., maximum liquidity) and w determines the width of the M-shape peaks. Figure 7.2 presents an example of daily volatility scenarios generated using (7.3) and their average.

This model can be extended to a multivariate form, accounting for correlation between different FX rates. For a given correlation matrix $\mathbf{C}$, the following equation will generate correlated stochastic variables [136]:

$$\mathbf{W} = \mathbf{W}_{i.i.d.}\mathbf{L} \tag{7.4}$$

Here, $\mathbf{L}$ is obtained from a Cholesky decomposition $\mathbf{C} = \mathbf{L}\mathbf{L}^*$, $\mathbf{W}_{i.i.d.}$ is a matrix of independent and identically distributed stochastic variables, and $\mathbf{W}$ is the resulting matrix of correlated stochastic variables.

Correlated Wiener processes generated using (7.4) are then used in (7.1). Addition of events or drift is performed in a manner analogous to the univariate case.

7.2.2 Transaction Cost Model

The main component of transaction costs, as defined in (3.5), is the bid-ask spread, which in (3.4) was modelled as an affine function of trade size.

In addition to the size of trade, the bid-ask spread is also affected by liquidity. The liquidity increases during mid-trading hours, based on the geographic distribution of currency traders, and is reduced in non-trading hours [87].

We model this time-varying liquidity effect with a U-shaped market impact coefficient:

$$\delta(t) = \delta\left(1 + \frac{\rho - 1}{2}\left(1 + cos\left(2\pi\frac{t + \frac{N}{2} - t_{min}}{N}\right)\right)\right) \tag{7.5}$$

Here, $\delta(t)$ is the time-varying market impact coefficient, δ is the coefficient's daily minimum, ρ is the ratio of maximum daily market impact to its minimum, N is the number of time-steps in the day, and t_{min} determines the time of maximum liquidity.

Figure 7.3 shows an example of scenarios for daily bid-ask spread and their average generated by (7.5).

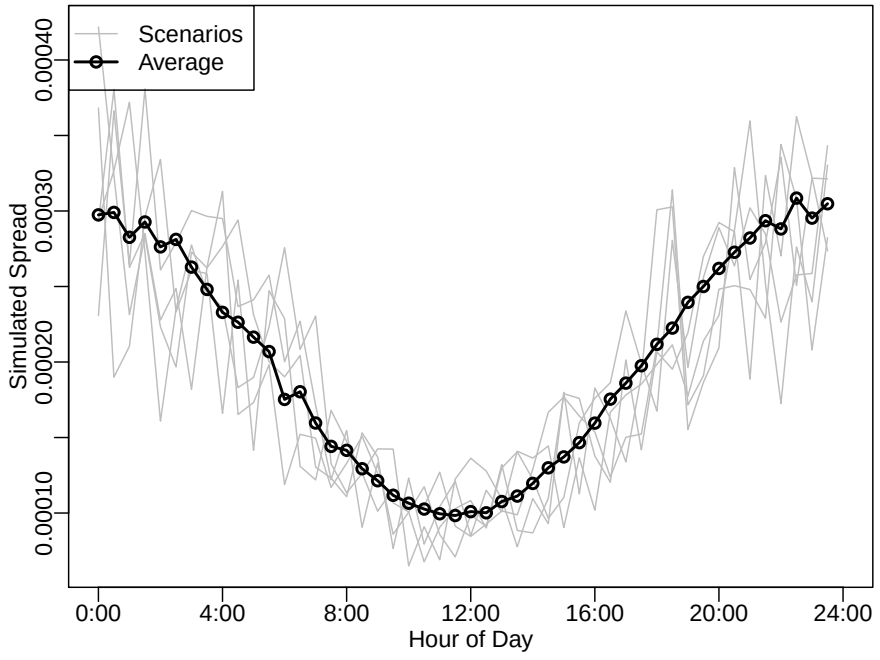


FIGURE 7.3: Simulated U-shaped bid-ask spreads during the day.

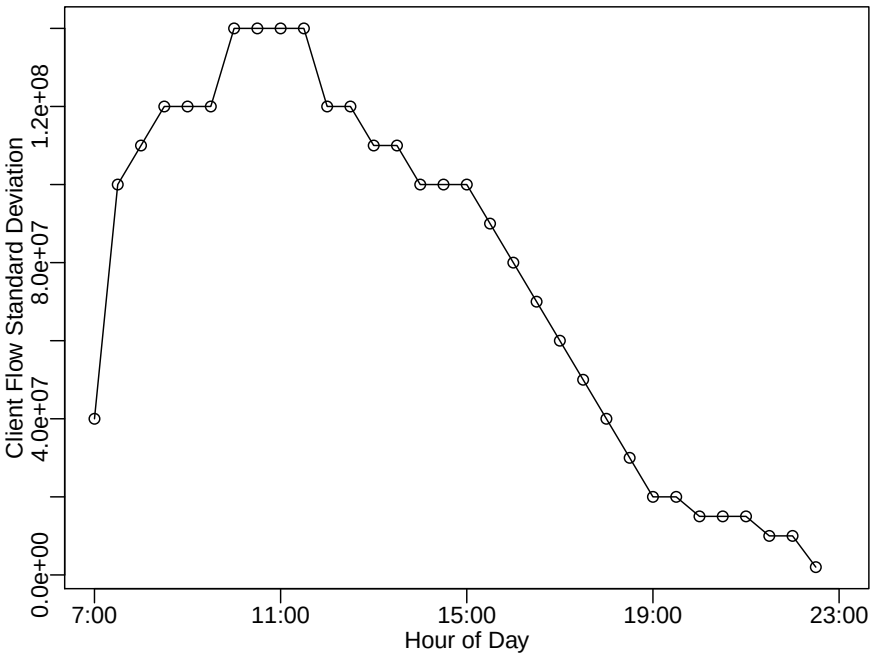


FIGURE 7.4: Standard deviation of synthetic client flow versus time.

7.2.3 Client Flow Model

The choice of client flow model is a dealer dependent task. Cerrato et al. [227] noted that in response to price moves, corporate and private clients act more as liquidity providers while profit-motivated speculative traders (e.g., hedge fund investors and asset managers) exhibit a directional behaviour as a result of being more informed. Furthermore, Chen et al. [228] found significant relationships between volatility, spreads, and the order flow of speculative traders. This information can be used by dealers to better model their client flow.

In this Section, we model private domestic clients who mainly trade during their own country's working hours. These clients exhibit a certain periodicity, e.g., more trades happen mid-day rather than at the end-of-day hours. Also, the dealers expect a bias in buys versus sells in a certain direction relative to their home currency, which is not influenced by price movement. Therefore, the client flow is modelled as a heteroskedastic Gaussian process, with a time-dependent mean μ_f and variance σ_f^2:

$$f(t) \sim \mathcal{N}(\mu_f(t), \sigma_f^2(t)) \tag{7.6}$$

Figure 7.4 shows an example of $\sigma_f(t)$ for a dealer trading from 7:00 to 23:00, with the assumptions presented earlier.

7.2.4 Scenario Generating Oracle

We define an oracle that generates scenarios by perturbing observations with alternative scenarios:

$$x^{(p)}(t) = \alpha e^{-\beta t} x(t) + (1 - \alpha e^{-\beta t}) x'(t) \tag{7.7}$$

Here, $x^{(p)}(t)$ is the oracle generated (i.e., perturbed) scenario used in optimisations, $x(t)$ is the actual future data used for backtesting, $x'(t)$ is an alternative scenario sampled from the models' statistical distribution, and $0 \leq \alpha \leq 1$ and $\beta \geq 0$ are initial accuracy and oracle decay rate parameters. To generate extreme cases, $\alpha = 1$ and $\beta = 0$ are used for prescient hedging, while $\alpha = 0$ tests the quality of data models with ordinary scenario generation. Furthermore, $\alpha > 0$ and $\beta > 0$ create an exponentially decaying accuracy, simulating a more realistic signal decay model.

7.3 Implementation and Simulation Details

Several tests using synthetic and real-world data were performed to validate and measure the performance of the SMPC hedging algorithm proposed in Section 3.6. This algorithm is compared against three benchmark strategies:

1. Limited position strategy (proposed in Section 3.4.3).
2. Gradual closing strategy (proposed in Section 3.4.4).
3. Single-stage shrinking horizon (SHC) hedging, which is a simplified form of SMPC hedging (proposed in Section 3.5).

In addition, a *prescient* hedging strategy is also demonstrated, by *peeking* into future data and directly solving the hedging objective (3.10).

The experiments were implemented in R programming language [202], with the optimisation function written in C++ using QuadProg++ [229] and Eigen linear algebra template libraries [230]. Run-time of the hedging algorithm for each trading session with 50 scenarios on a 3.4 GHz Intel Core i7-2600 processor was measured to be less than 50 ms, making it suitable not only for backtesting, but also for deployment in online hedging systems.

7.3.1 Synthetic Data

The synthetic data was generated according to the models described in Section 7.2. It is assumed the dealer worked from 7:00 to 23:00 and hedged at 30 minute steps.

Client flow was generated with standard deviation, σ_f, as depicted in Figure 7.4, and a positive mean, $\mu_f = \frac{1}{4}\sigma_f$.

For the FX rate volatility and market impact coefficients, each currency was assigned different δ, ν, ρ, ω, and t_{min} parameters. At the start of each trading session, the asset correlation matrix $\mathbf{C}$, and number, time, and impact of announced events $k(t)$ were regenerated randomly.

7.3.2 Real-world Data

For real-world tests, 16 weeks of actual FX client transaction data supplied by Westpac institutional bank was used[1]. AUD was chosen as the home currency and USD, EUR, NZD, and JPY were selected as foreign assets. This data was filtered and aggregated to create 32 half-hourly values per day, from 7:00 to 23:00 Australian Eastern Daylight Time (AEDT). The first six weeks (30 days) of data was used to obtain model parameters as described in Section 7.2.3 and the rest (i.e., 50 days) were used for out-of-sample backtesting.

Accordingly, half-hourly values of historical FX rates were fit to the model defined in Section 7.2.1. Individual variances ν and correlation matrix $\mathbf{C}$ were computed from covariances of the previous day, and jump component timings were extracted from the publicly[2] available DailyFX.com event calendar [232]. Jump intensity was set to $k = 5\nu$. Only events classified in DailyFX.com as high impact were considered in this simulation and the rest were discarded.

As the inter-bank market impact data was not available, U-shaped daily seasonality was generated synthetically. The market impact coefficient, δ, was selected as the square of average spread, 0.5, 1, 2 and 1 basis points (i.e., per 10000) for USD, EUR, NZD, and JPY respectively. The time of maximum liquidity for each currency was chosen as 12:00 PM of its main market's geographic distribution, i.e., New York, Frankfurt, Auckland and Tokyo respectively.

7.4 Single-asset Tests

To test the correctness of the proposed SMPC hedging scheme, first a simple test with one foreign currency asset for one trading session was performed. Data was synthetically

[1] This data was anonymised to protect Westpac client confidentiality.

[2] DailyFX terms and conditions permit personal, non-commercial use of their data without subscription [231].

generated, as described in Section 7.3.1. FX rate returns were created with only one announcement event occurring at 12:00, as shown in Figure 7.1.

Figure 7.5 compares the accumulation of positions in absence of hedging against different stochastic paths. Figure 7.6 and Figure 7.7 show the hedged positions and hedging actions respectively using the SMPC hedging with $\lambda = 0.01$. Figure 7.6 and 7.7 also show results of open-loop optimisation in grey: each line represents the results if hedging actions were not recalibrated again at each time-step according to Algorithm 3.1.

It can be seen from Figure 7.7 that the proposed method has gradually reduced the positions to zero at the end of trading session; this lessens the market impact and consequently transaction costs. Furthermore, comparing hedged and unhedged positions in Figure 7.8 shows that by using the knowledge of the announced event at 12:00, the open positions were gradually minimised before the event to reduce the possibility of losses in case of a major unfavourable FX rate movement.

7.5 Multi-asset Tests

To measure the hedging performance in a multi-asset portfolio, backtesting was performed with four foreign currencies using both synthetic and real-world data.

A total of 50 different trading sessions were backtested for the scenario based SMPC hedging, as well a the three benchmark strategies and prescient hedging. To generate risk-cost profiles, each run included different parametrisations of x_{max} in (3.15) for the limited position strategy and λ for other strategies. The end-of-trading costs and profits were computed according to (3.3) and (3.7) respectively and normalised by the daily total value of transactions $\sum |f(i)|$ to basis points (bps). The normalised values were then converted to risk-cost profiles.

Figure 7.9 shows the risk-cost profiles for the synthetic data test, while the real-world data's profiles are reported in Figure 7.10. It is shown that SMPC hedging outperforms all benchmark hedging strategies by offering a lower cost for any given risk in the synthetic test.

It was noted that in the real-word experiment, the cost saving improvements are noticeable only compared to the strategies of Section 3.4 which do not employ any information about the future. Furthermore, there is considerable room for improvement compared to the prescient hedging frontier in both tests. This problem can be traced to scenario generation, where the simple models of Section 7.2 are unable to capture certain features

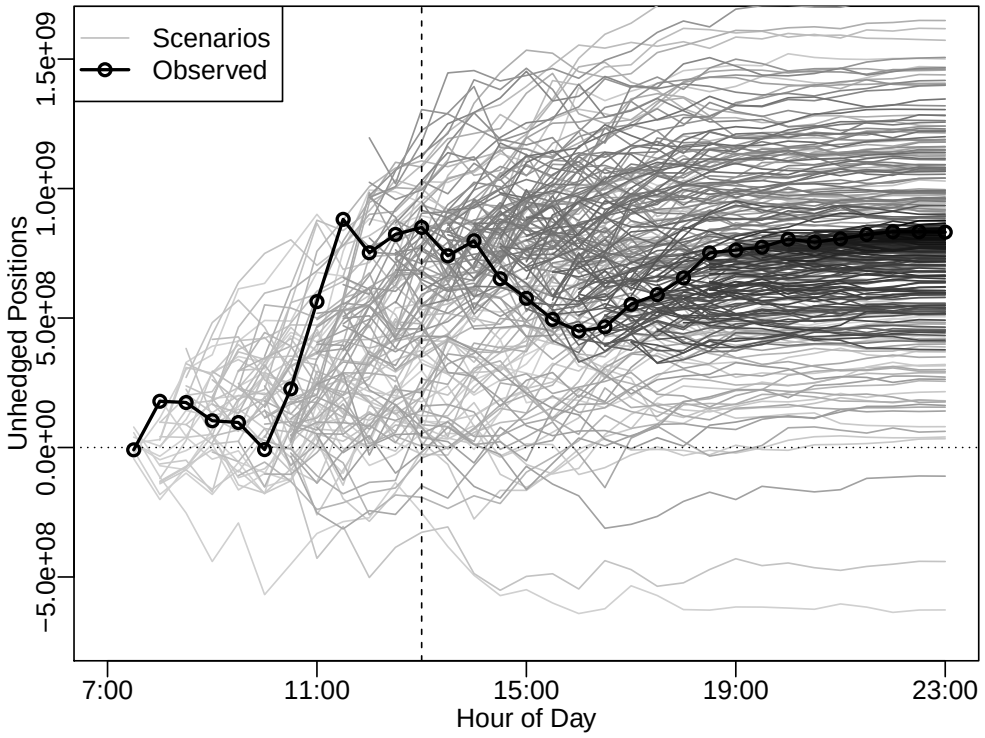


FIGURE 7.5: The dealer's unhedged position $y(t)$ through the trading session in the single-asset test. Each grey line shows a stochastic scenario estimated at various times of the trading session, with later scenarios coloured darker. The vertical line marks the location of the announcement event.

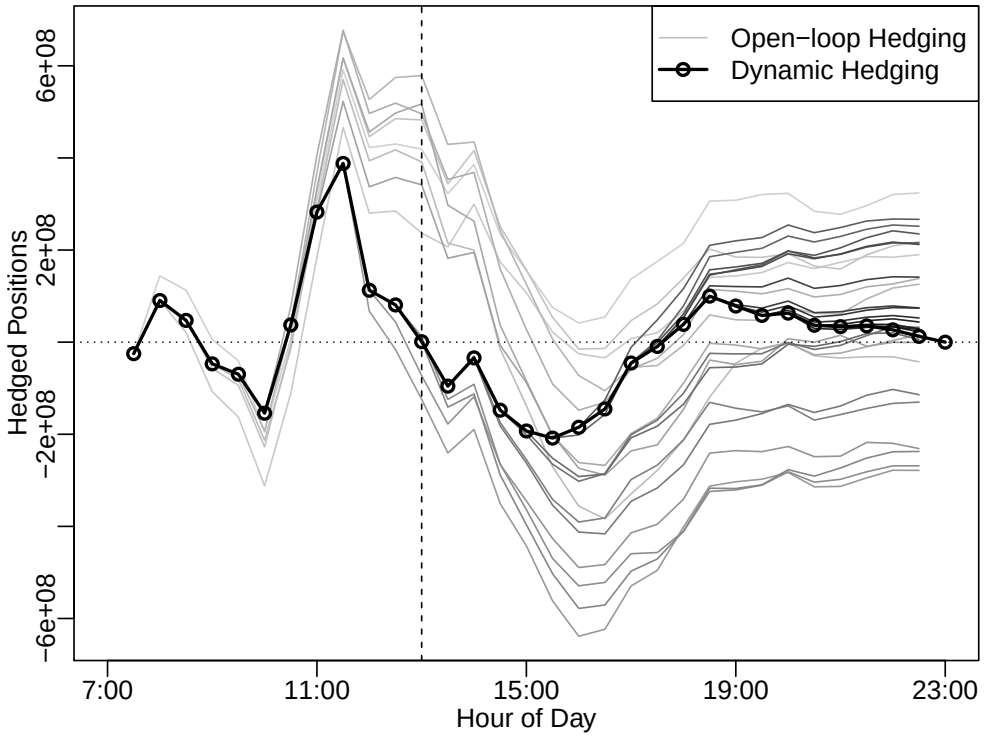


FIGURE 7.6: The dealer's hedged position $x(t)$ in the single-asset test. Each grey line shows result of open-loop hedging at various times of the trading session, with later results coloured darker. The vertical line marks the location of the announcement event.

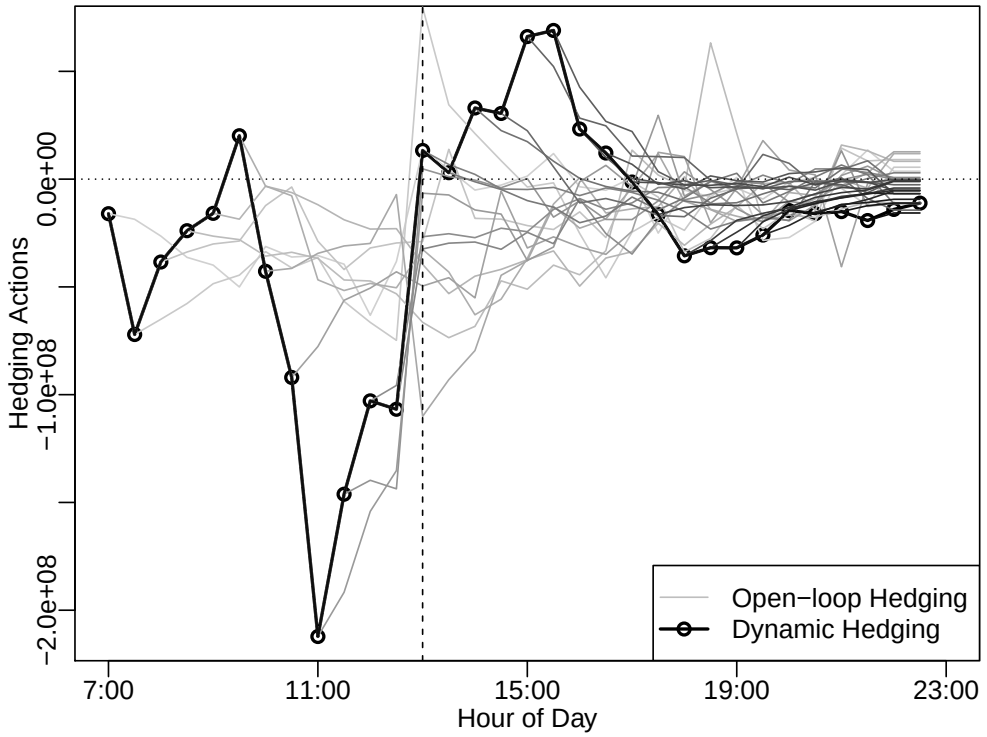


FIGURE 7.7: Hedging actions $h(t)$ in the single-asset test. Each grey line shows results of optimisation performed at various times of the trading session, with later optimisations coloured darker. The vertical line marks the location of the announcement event.

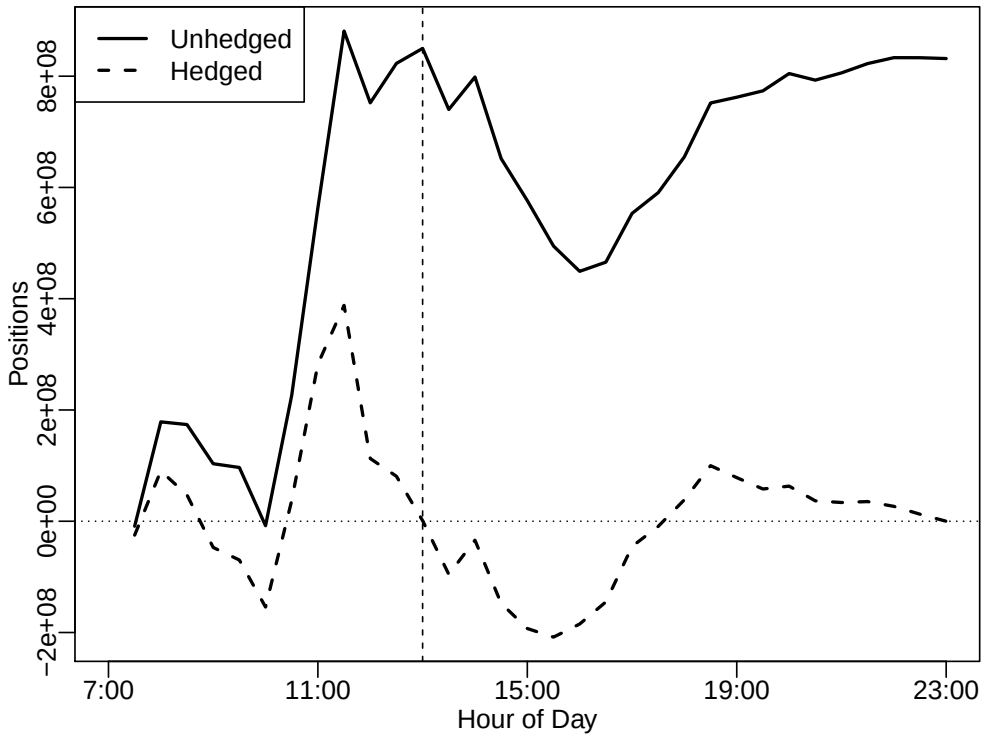


FIGURE 7.8: The dealer's positions in the single-asset test with and without hedging.

such as the fat-tailed distribution of data, correlation between FX rate and client flow, and variation of transaction cost around events, specially for the real-world data test.

To confirm the superiority of the proposed algorithm over the single-stage method regardless of the data model quality, the real-world test was repeated with prescient data in Figure 7.11. A comparison shows that the proposed method significantly outperforms other strategies when its input scenarios are accurate.

7.6 Scenario Quality and Oracle Tests

In the previous experiments, the impact of scenario generation accuracy on hedging was demonstrated. In this section, this impact is quantified using the oracle model of (7.7).

Figure 7.12 and Figure 7.13 compare hedging using scenarios generated by the oracle with $\alpha = 0.5, \beta = 0.9$ and $\alpha = 0.9, \beta = 0.5$ for all scenario components, against ordinary scenario generation and prescient hedging for synthetic and real-world data respectively. As expected, improved scenario generation (i.e., increasing initial accuracy α and reducing decay rate β) results in less cost for any chosen level of risk.

The effect of improving accuracy for individual components on risk management is compared in Figure 7.14 for synthetic and in Figure 7.15 for real-world data. Here, a high degree of accuracy ($\alpha = 1$, $\beta = 0.2$) is used to generate different scenarios for client flow, volatility and market impact coefficients separately.

In Table 7.1 and Table 7.2, this test is repeated with a wider range of α and β's for synthetic and real-world data respectively. The results were averaged over the number of backtesting sessions, and the normalised costs for risk = 10 bps were scaled to denote 0% for the single-stage hedging strategy (ie, the third benchmark) and 100% for prescient hedging. For both tests, it is observed from the tables that improving scenario generation produces better hedging profiles. Yet, hedging enjoys the most improvements with better client flow modelling, while any further market impact coefficient model enhancements have minor effects on the final results. For example in the synthetic test, the ordinary scenario generation reduces costs by 25.9%, and using a perfect market impact model and a perfect volatility model improves this to 26.5% and 38.2% respectively. In comparison, a perfect client flow modelling boosts the cost savings up to 87.8%. A similar phenomenon is noticeable for real-world data, improving cost reduction by 1.0%, 27.7% and 73.8% with perfect transaction cost, volatility, and client flow modelling respectively. A similar trend is also present in Figure 7.14 and Figure 7.15.

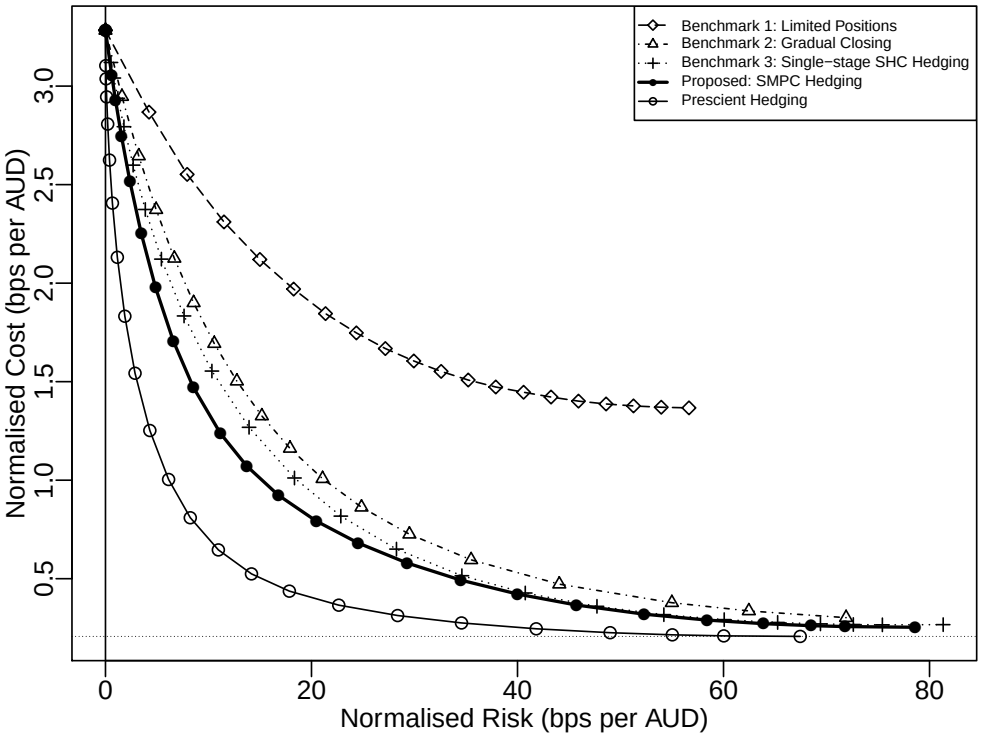


FIGURE 7.9: Risk-cost profiles for the synthetic data experiment.

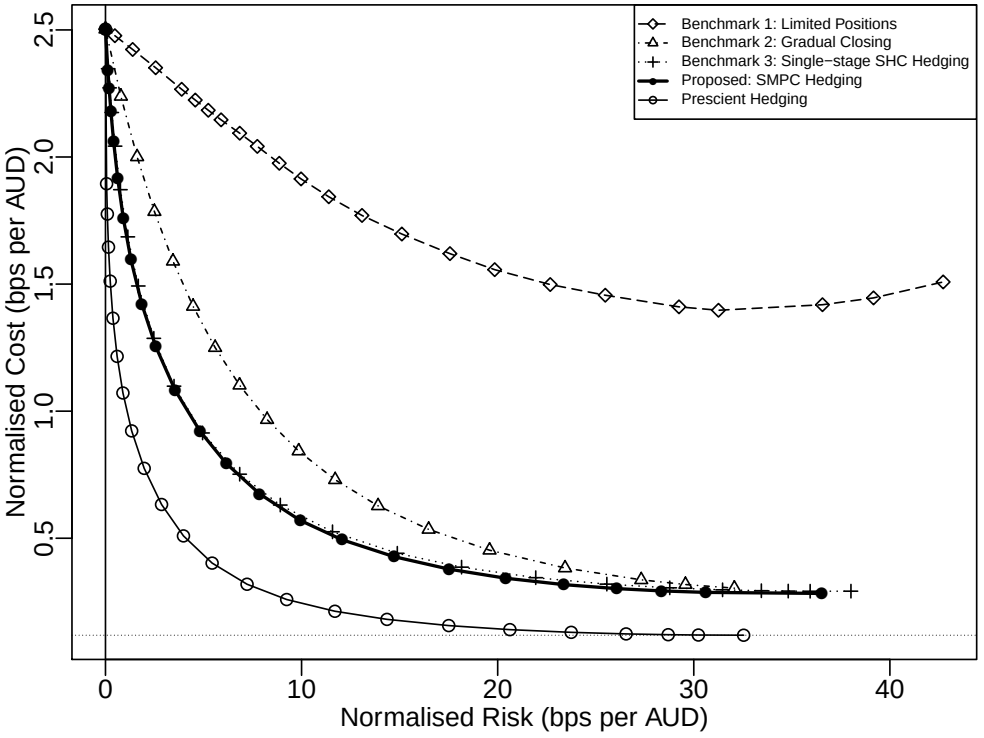


FIGURE 7.10: Risk-cost profiles for the real-world data experiment.

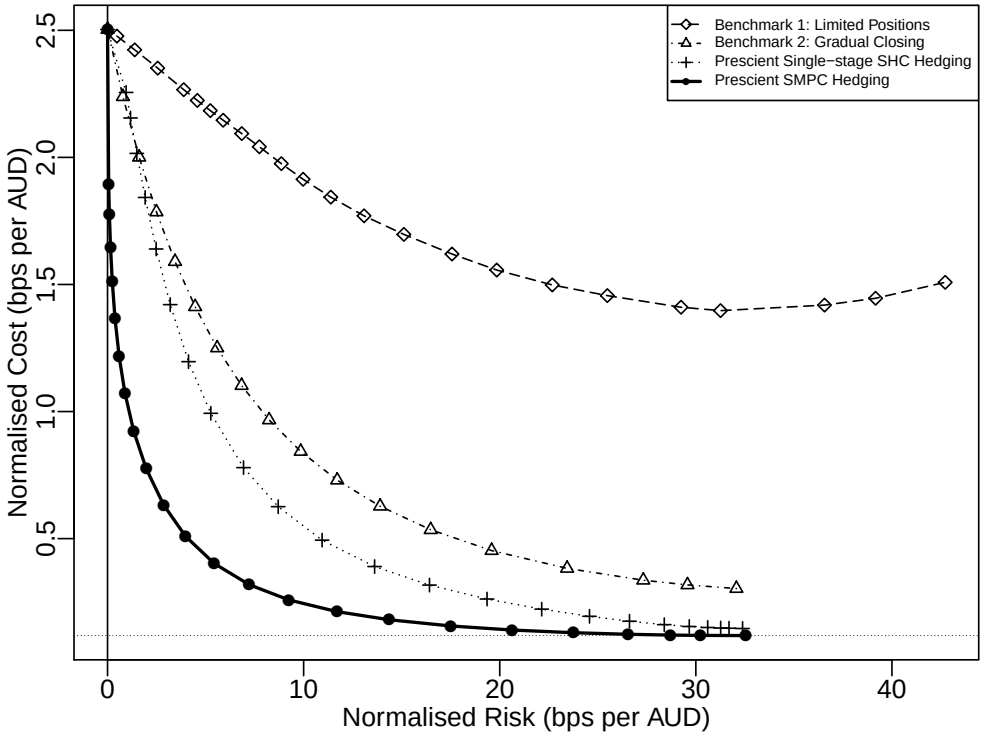


FIGURE 7.11: Risk-cost profiles for the real-world data experiment with prescient data.

These differences are the result of using more definite models for market impact and FX rate, versus a less precise model for describing client flow. Most of the actual market impact and FX rate variations are captured by the ordinary scenario generation, with not much left to be improved by the oracle. On the other hand, the simple model of Section 7.2.3 does not produce accurate client flow scenarios; hence, major improvements are observed using the oracle.

Furthermore, Figure 7.15 shows an interesting effect. Improving volatility scenarios offers a better cost versus risk compared to improving client flow for the low risk region (i.e., the left side) of the plot. Comparably, in the low cost region (i.e., the right side), this risk-cost profile converges with the ordinary scenario generation curve, while using the oracle only for the client flow gives a considerably better cost versus risk in this region, converging with the prescient hedging frontier. This is a direct effect of λ's influence on minimising the hedging cost function (3.20). That is, when $\lambda \to 0$ (only transaction cost is being minimised), the effect of client flow modelling error is more evident, and when $\lambda \to \infty$ (i.e., only minimising risk), the client flow modelling errors become negligible compared to FX rate modelling error. In conclusion, the dealers can choose to invest only on better client flow or volatility modelling depending on their risk preference.

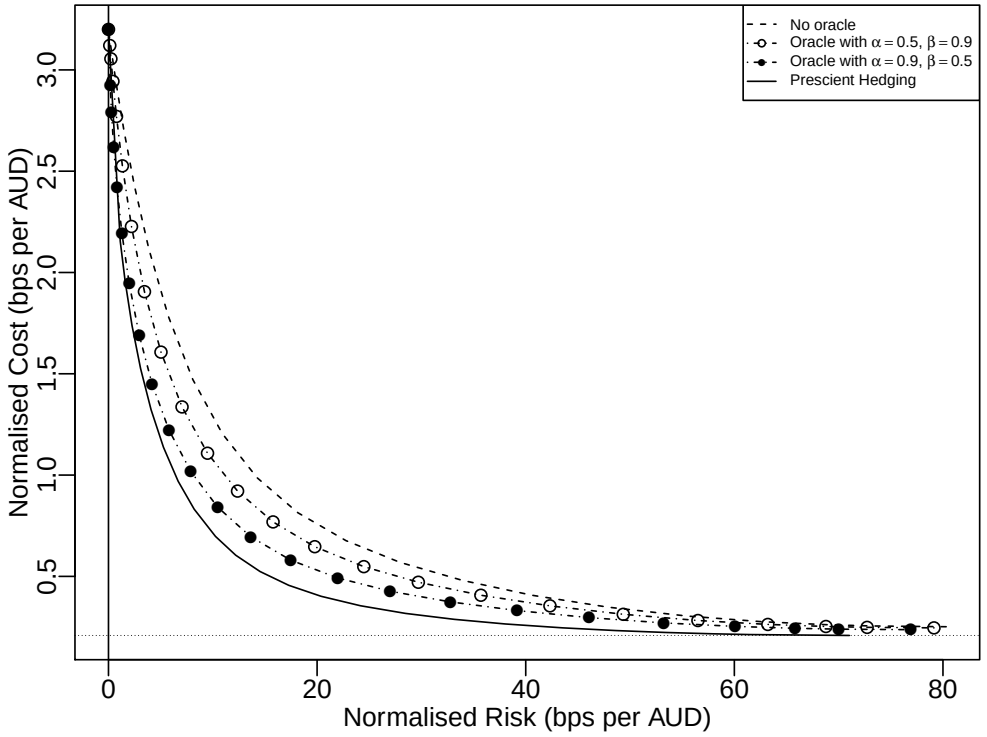


FIGURE 7.12: Effect of oracle accuracy on risk-cost profile improvement in the synthetic data experiment.

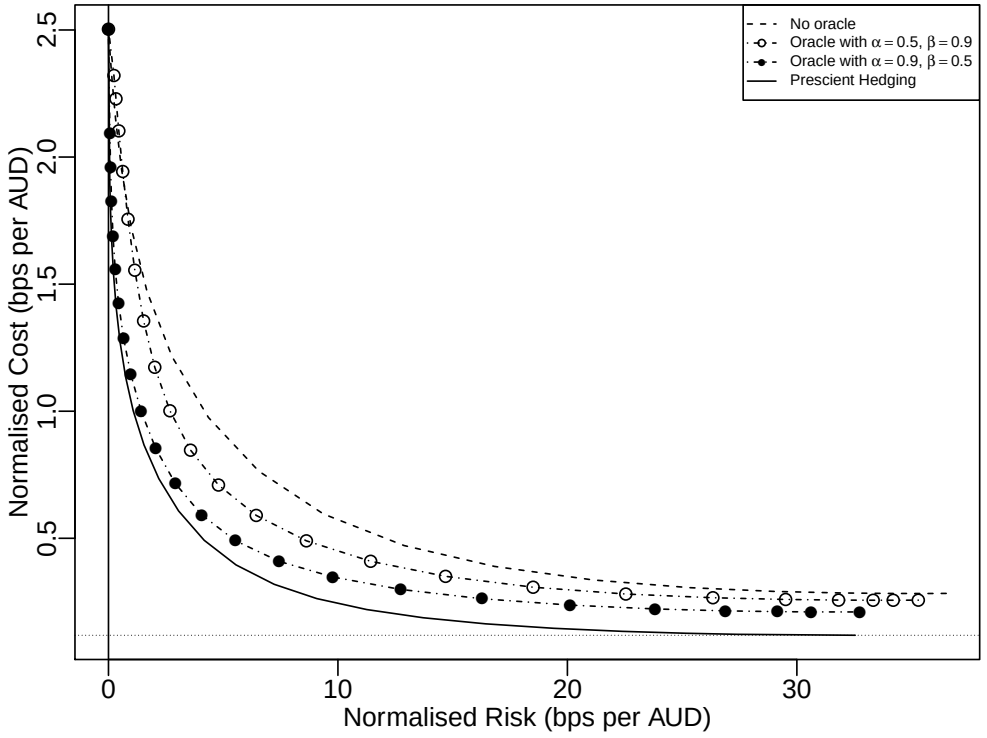


FIGURE 7.13: Effect of oracle accuracy on risk-cost profile improvement in the real-world data experiment.

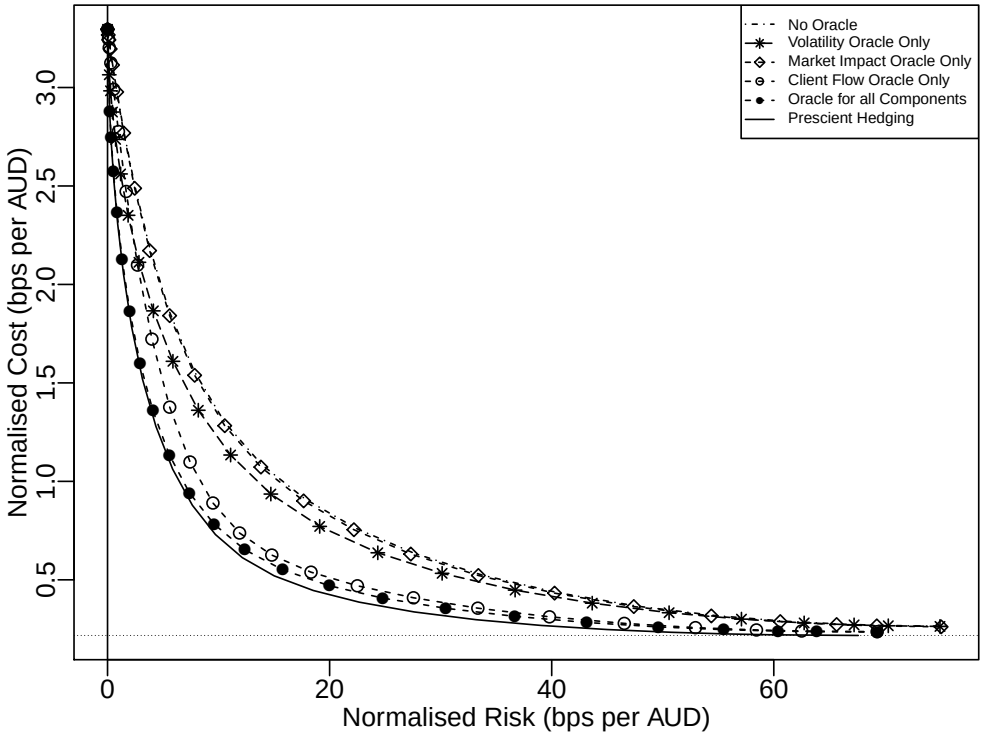


FIGURE 7.14: Effect of using the oracle for individual scenario components on synthetic data experiment's risk-cost profiles.

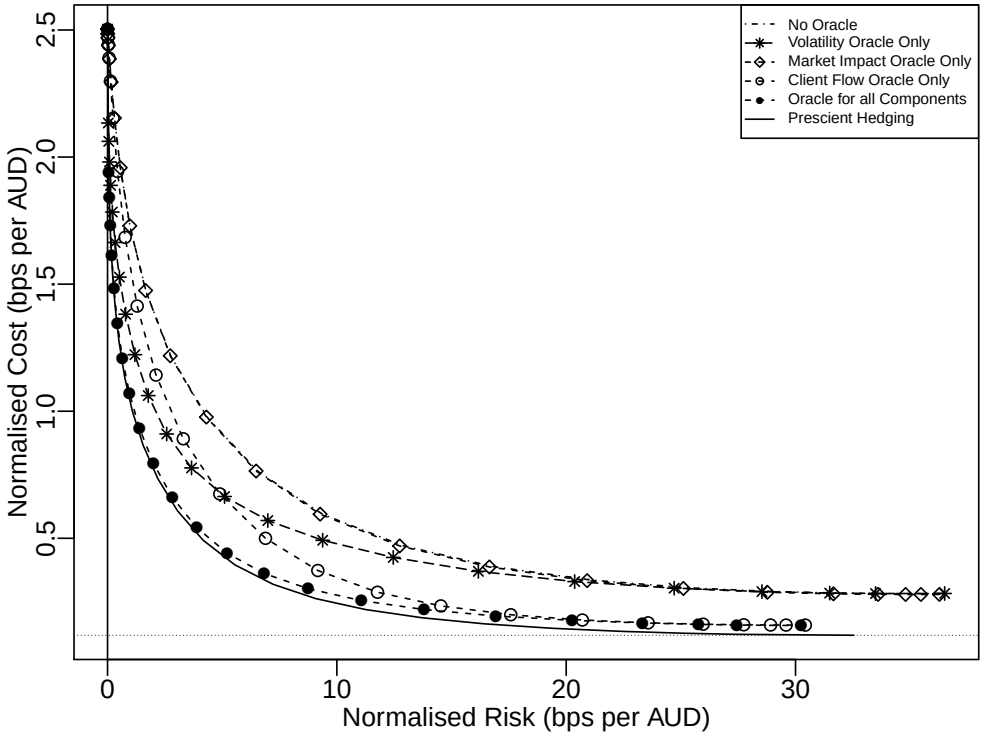


FIGURE 7.15: Effect of using the oracle for individual scenario components on real-world data experiment's risk-cost profiles.

TABLE 7.1: Percentage of cost savings for different oracle accuracies in the synthetic data experiment. The values show normalised costs for risk = 10 bps, scaled between the single-stage strategy and prescient hedging results.

Oracle	$\beta = 1$	$\beta = 0.8$	$\beta = 0.6$	$\beta = 0.4$	$\beta = 0.2$	$\beta = 0$
$\alpha = 0$	25.9					
Market impact:						
$\alpha = 0.2$	25.9	25.7	25.1	24.7	25.7	26.3
$\alpha = 0.4$	25.7	25.4	27.2	27.0	26.3	27.1
$\alpha = 0.6$	26.3	26.5	26.1	25.5	25.1	26.3
$\alpha = 0.8$	25.3	25.9	26.2	25.8	26.6	26.5
$\alpha = 1$	26.6	26.3	26.0	27.9	25.7	26.5
Volatility:						
$\alpha = 0.2$	26.1	27.2	27.1	27.1	28.3	28.6
$\alpha = 0.4$	30.8	31.2	31.6	31.8	31.8	32.6
$\alpha = 0.6$	36.2	34.8	35.2	35.6	35.9	35.7
$\alpha = 0.8$	37.0	37.6	37.5	38.9	38.1	38.4
$\alpha = 1$	37.8	38.2	37.7	37.5	38.8	38.2
Client flow:						
$\alpha = 0.2$	32.2	33.4	37.3	39.8	43.3	46.9
$\alpha = 0.4$	39.9	43.0	46.5	50.8	57.9	65.4
$\alpha = 0.6$	46.4	48.7	54.5	60.9	69.0	78.3
$\alpha = 0.8$	50.5	56.5	63.0	69.5	78.3	84.9
$\alpha = 1$	55.3	61.2	68.8	76.0	83.9	87.8
All components:						
$\alpha = 0.2$	35.1	36.7	38.6	42.8	45.9	51.3
$\alpha = 0.4$	46.0	50.3	53.3	58.4	64.6	72.2
$\alpha = 0.6$	56.1	60.2	65.4	72.6	79.9	89.0
$\alpha = 0.8$	62.1	67.7	72.8	80.6	88.7	97.3
$\alpha = 1$	65.3	72.0	78.5	86.0	94.4	100.0

TABLE 7.2: Percentage of cost savings for different oracle accuracies in the real-world data experiment. The values show normalised costs for risk = 10 bps, scaled between the single-stage strategy and prescient hedging results.

Oracle	$\beta = 1$	$\beta = 0.8$	$\beta = 0.6$	$\beta = 0.4$	$\beta = 0.2$	$\beta = 0$
$\alpha = 0$	4.6					
Market impact:						
$\alpha = 0.2$	4.8	4.8	4.9	5.0	4.8	4.8
$\alpha = 0.4$	5.1	5.1	5.2	5.1	4.9	5.1
$\alpha = 0.6$	5.3	5.3	5.2	5.3	5.3	5.3
$\alpha = 0.8$	5.5	5.4	5.4	5.5	5.5	5.5
$\alpha = 1$	5.5	5.5	5.5	5.5	5.5	5.6
Volatility:						
$\alpha = 0.2$	0.4	0.9	1.2	1.6	1.8	1.7
$\alpha = 0.4$	16.5	16.3	16.0	16.3	17.6	20.5
$\alpha = 0.6$	23.9	24.3	24.8	26.1	28.2	31.0
$\alpha = 0.8$	27.2	27.7	28.7	29.6	31.2	32.9
$\alpha = 1$	29.2	29.7	30.7	31.8	31.6	32.3
Client flow:						
$\alpha = 0.2$	12.7	14.9	16.9	20.3	24.5	30.1
$\alpha = 0.4$	19.8	23.6	27.9	33.5	40.9	49.6
$\alpha = 0.6$	26.5	31.5	37.4	44.4	53.5	63.9
$\alpha = 0.8$	32.6	38.4	45.1	53.8	63.5	73.3
$\alpha = 1$	37.9	44.6	52.1	61.0	70.3	78.4
All components:						
$\alpha = 0.2$	9.7	11.3	13.4	16.3	20.1	25.3
$\alpha = 0.4$	31.6	34.9	40.6	46.2	54.2	64.4
$\alpha = 0.6$	44.3	48.8	54.8	62.8	73.6	87.0
$\alpha = 0.8$	50.3	55.8	62.9	72.2	84.3	97.2
$\alpha = 1$	54.3	60.5	68.4	78.4	90.7	100.0

7.7 Automatic Client Flow Forecasting

In Section 7.6, specifically Table 7.2, it was shown that improving the quality of client flow scenarios has the most significant effect on the overall hedging quality.

In this section, the evolutionary time-series forecasting techniques proposed in Chapter 5 will be integrated as a part of the hedging system, in order to model and predict the client flow. The forecasting target is the expected client flow, $\mathbb{E}[\boldsymbol{f}]$, for each of the foreign currencies being traded, i.e., $\boldsymbol{f}_{USD}$, $\boldsymbol{f}_{EUR}$, $\boldsymbol{f}_{NZD}$, and $\boldsymbol{f}_{JPY}$. As the client flow is non-stationary and clients' behaviour changes with time, the model selection is repeated for every week of data. This results in a total of 40 different models (i.e., four currencies over 10 weeks), justifying automatically evolved models instead of a hand-tuned methodology.

Additionally, the error measure introduced in Chapter 6 will be employed in the evolutionary fitness function to compare different forecasters. This error, which models the reaction of the hedging system's LQ formulation to forecast errors in closed-form, improves computation speed compared to full backtesting, while keeping the same amount of realistic assumptions in evaluating errors of various prediction horizons.

7.7.1 Historical Data and Forecast Horizon

The data was prepared similar to that in Section 7.3.2. The same sixteen weeks of Westpac FX client transaction data were selected, with AUD chosen as the home currency and USD, EUR, NZD, and JPY selected as foreign assets. This data was filtered and aggregated to create 16 hourly values per day, from 7:00 to 23:00 AEDT.

As the hedging algorithm requires the client flow forecasts until the end-of-trading session, the forecast horizon was determined dynamically based on the number of hours until 23:00 for every point in time.

7.7.2 Prediction Grammar

Conforming to the methodology proposed in Chapter 5, first a grammar was designed to describe the pre-processing steps, feature generation, and the learning model. Subsequently, a cross-validation scheme was used to evolve the grammar to select the best forecaster for the data.

7.7.2.1 Pre-processing

Pre-processing grammar was designed according to the methodology described in Section 5.3.2. The most significant change was adding a new logarithmic variance stabilisation function: as the client flow can be negative to denote sells, the ordinary logarithm cannot be used without resorting to complex numbers. Hence, a *signed pseudo-logarithm* function, defined as

$$\operatorname{slog}(x) = \begin{cases} \log(x) & x \geq 1 \\ 0 & -1 \leq x \leq 1 \\ -\log(-x) & x \leq -1 \end{cases}$$

is proposed. This function is not strictly invertible, as loss of information occurs at the $-1 \leq x \leq 1$ range. In practice, considering the average flow which is in millions of dollars, one can reverse the results with only a negligible loss.

Additionally, the known daily and weekly cycle duration, respectively being 16 and 80 samples, were added to the frequency list.

7.7.2.2 Features

The feature generation grammar was also implemented as described in Section 5.3.3. Implemented features include:

- An auto-regressive component, customisable with a lag order.
- Seasonal auto-regressive components for daily and weekly cycles, customisable with lag and width orders.
- FX volatility: Windows of hourly realised volatility were used as explanatory variables. In a fair study, only ex-ante (i.e., predicted) volatility should be used; however, the ex-post (i.e., observed) volatility was included instead. This is to validate this hypothesis that using volatility improves client flow prediction. If a relationship is found which was not already covered by other variables (e.g., time-of-day, which theoretically drives both flow and volatility [87]), one can then undertake extra computational efforts to improve volatility prediction over the stochastic volatility model that was proposed and used in Section 7.2.
- Hour-of-day dummy variables: A binary vector with length 15, determining the hour of day from 7 a.m. to 11 p.m.. The 16th value was removed, as it is linearly dependent on other values.

TABLE 7.3: Candidate features for predicting FX client flow.

Feature	Window type	Maximum width	Maximum lag
Auto-regressive	Past lags	N/A	16
Daily cycle auto-regressive	Past lags, centred	4	10
Weekly cycle auto-regressive	Past lags, centred	4	2
Realised FX volatility	Centred	4	N/A
Hour-of-day	Binary vector	N/A	N/A
Day-of-week	Binary vector	N/A	N/A

- Day-of-week dummy variables: A binary vector with length 4, determining the day of week. The fifth variable was removed, as it is linearly dependent on other values.

The details of these features are presented in Table 7.3.

7.7.2.3 Learner Models

The learner model grammar was implemented as described in Section 5.3.4. Two learning techniques were used:

- A linear model, which is commonly used to describe forecasting models in finance and econometrics [233].
- A SVM with radial basis function kernel. Hyper-parameters of the support vector machine include ϵ, γ, and C [234].

7.7.2.4 Fitness Function

Fitness function implementation followed the cross-validation methodology described in Section 5.4.3. As mentioned earlier, to adapt to the variable client flow dynamics, a new forecasting model was selected for each of the ten weeks in the backtesting period. For each model, all data from past weeks were used for cross-validation, with the length of the testing period fixed to four weeks (20 days), and the rest of the data allocated to initial model fitting. Using Algorithm 5.4, the aggregated forecast error of rolling window predictions over the testing period was reported as the fitness score.

In two separate evolutionary optimisation runs, root mean square error (RMSE) as well as the ΔJ measure proposed in Chapter 6 were used to calculate the forecast errors in the fitness function.

In Section 6.7, it was shown that deviation of the hedging cost function from the optimal, where the flow forecast is $\hat{\boldsymbol{f}}$ and the observed flow is $\boldsymbol{f}$, can be determined by

$$\Delta J = (\hat{\boldsymbol{f}} - \boldsymbol{f})^{\mathrm{T}} \boldsymbol{\Theta} (\hat{\boldsymbol{f}} - \boldsymbol{f}) \tag{7.8}$$

where $\boldsymbol{\Theta}$ is obtained from (6.15).

The results of Section 7.6, in addition to the analysis of Chapter 3, reveal that the sensitivity of the hedging to client flow is at its maximum when the minimum cost is desired (i.e., the right end of risk-cost profiles). Hence, $\boldsymbol{\Theta}$ was computed accordingly with $\lambda = 0$. Notice that the ΔJ is not a replacement for backtesting, as different conditions including the variable volatility and market impact are not included in its assumptions.

Accordingly, RMSE was computed using

$$RMSE = \sqrt{\frac{1}{N} (\hat{\boldsymbol{f}} - \boldsymbol{f})^{\mathrm{T}} (\hat{\boldsymbol{f}} - \boldsymbol{f})}$$

where N is the length of vector $\boldsymbol{f}$, i.e., the total number of predictions over all horizons.

7.7.2.5 Grammatical Evolution

Grammatical evolution was implemented using the package *gramEvol* [11]. The number of generations was set to 50, and the rest of the parameters were determined automatically by the package. More details about the architecture of this package are presented in Appendix A.

7.7.3 Comparison with Other Techniques

In addition to evolving grammar based predictors, the following techniques were also employed for forecasting the client flow:

- Gaussian model: In Section 7.2.3, a Gaussian process was used for generating client flow scenarios. The same stochastic model, i.e., Eq. (7.6), was fitted to the first six weeks of historic data and used to forecast the expected future flow.
- Naïve predictor: A naïve predictor repeats the data from last observation as the future forecasts. This is effectively equal to considering a random walk model for the data [36].
- Averaging model predictor: The client flow for the last four weeks were averaged and the mean value was reported as the future flow.

- ETS and ARIMA: The R package *forecast* [200] was used to create ARIMA and ETS forecasting models. Model orders were determined automatically by the package, using the same period selected for training the grammar based prediction. To reflect data behaviour changes in the model training, for each prediction, model parameters were fit only to the most recent ten days of data.

7.7.4 Forecast Results

The prediction results, comparing the forecaster models evolved from the grammar against other techniques are presented in Table 7.4. Errors are reported separately for the cross-validation period and the out-of-sample period, i.e., the backtesting week after the model's cross-validation period. For each currency, the errors were averaged for all ten weeks, and then normalised to the error of the Gaussian model. The reported error measures are:

- ΔJ, implemented using (7.8), and averaged over every trading day.
- RMSE, averaged over every prediction window in a trading day.
- Mean absolute error (MAE), averaged over every prediction window in a trading day.

Compared to backtesting, which requires 50 ms for computing the hedging cost of a single day on a 3.4 GHz Intel Core i7-2600 machine, ΔJ, RMSE, and MAE only take 255 µs, 246 µs, and 104 µs to run respectively.

Table 7.4 shows that the out-of-sample results are dominated by the Gaussian model. Additionally, while the evolutionary model selection using the ΔJ measure was successful in finding the best in-sample models for ΔJ, it was not as successful for other measures, either in-sample or out-of-sample. A similar observation is made for the evolutionary optimisation using RMSE, where no other model was able to provide better results than the Gaussian distribution. This can be explained as GE over-fitting the models to the cross-validation data, and lack of an underlying repeatable pattern in the client flow. This phenomenon was previously observed by other researchers for FX rates [34, 36, 40], where the FX returns were best modelled using an i.i.d. Gaussian distribution. Several clues exist that hint at similar circumstances regarding the clients' behaviour [6, 44].

The results of backtesting, in the form of risk-cost profiles, are presented in Figure 7.16. The profiles include:

- Prescient hedging.
- SMPC hedging with Section 7.3.2's stochastic scenario generation (i.e., without time-series forecasting).

TABLE 7.4: Normalised client flow forecast error for different prediction techniques. Each measure is normalised as a percentage of its respective Gaussian model error.

Forecaster	In-sample			Out-of-sample		
	ΔJ	RMSE	MAE	ΔJ	RMSE	MAE
Evolved (with ΔJ)	**93.6**	106.5	124.4	117.6	112.0	131.5
Evolved (with RMSE)	101.6	**99.9**	101.3	102.0	100.6	103.2
Gaussian Model	100.0	100.0	**100.0**	**100.0**	**100.0**	**100.0**
Averaging	104.4	101.3	106.6	104.4	101.3	106.6
ARIMA	103.1	102.9	111.6	110.1	103.2	112.4
ETS	114.5	104.4	116.6	113.1	104.6	117.2
Naïve	1230.6	151.6	222.4	1230.6	151.6	222.4

(A) Normalised forecast errors for USD client flow.

Forecaster	In-sample			Out-of-sample		
	ΔJ	RMSE	MAE	ΔJ	RMSE	MAE
Evolved (with ΔJ)	**83.1**	109.7	134.9	104.3	112.8	137.1
Evolved (with RMSE)	99.9	100.2	101.1	106.4	102.4	104.4
Gaussian Model	100.0	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
Averaging	101.9	100.9	104.7	101.9	100.9	104.7
ARIMA	98.2	105.8	122.3	107.6	107.3	124.9
ETS	110.5	105.8	122.7	107.5	106.4	126.0
Naïve	1038.7	144.8	230.6	1038.7	144.8	230.6

(B) Normalised forecast errors for EUR client flow.

Forecaster	In-sample			Out-of-sample		
	ΔJ	RMSE	MAE	ΔJ	RMSE	MAE
Evolved (with ΔJ)	**84.0**	110.6	139.5	108.9	115.4	144.3
Evolved (with RMSE)	98.9	**98.1**	**98.2**	152.8	118.8	116.5
Gaussian Model	100.0	100.0	100.0	**100.0**	**100.0**	**100.0**
Averaging	100.3	101.0	105.4	100.3	101.0	105.4
ARIMA	96.3	102.6	112.4	101.3	102.5	110.4
ETS	109.6	104.1	118.1	109.0	104.6	120.1
Naïve	2083.7	154.2	257.2	2083.7	154.2	257.2

(C) Normalised forecast errors for NZD client flow.

Forecaster	In-sample			Out-of-sample		
	ΔJ	RMSE	MAE	ΔJ	RMSE	MAE
Evolved (with ΔJ)	**97.3**	114.2	148.1	103.5	117.8	147.4
Evolved (with RMSE)	99.7	**99.4**	**99.7**	100.9	102.0	102.2
Gaussian Model	100.0	100.0	100.0	**100.0**	**100.0**	**100.0**
Averaging	105.6	110.2	142.5	105.6	110.2	142.5
ARIMA	102.0	113.6	155.2	118.0	116.8	167.8
ETS	116.4	118.6	175.9	122.5	122.9	190.4
Naïve	2269.8	156.0	274.2	2269.8	156.0	274.2

(D) Normalised forecast errors for JPY client flow.

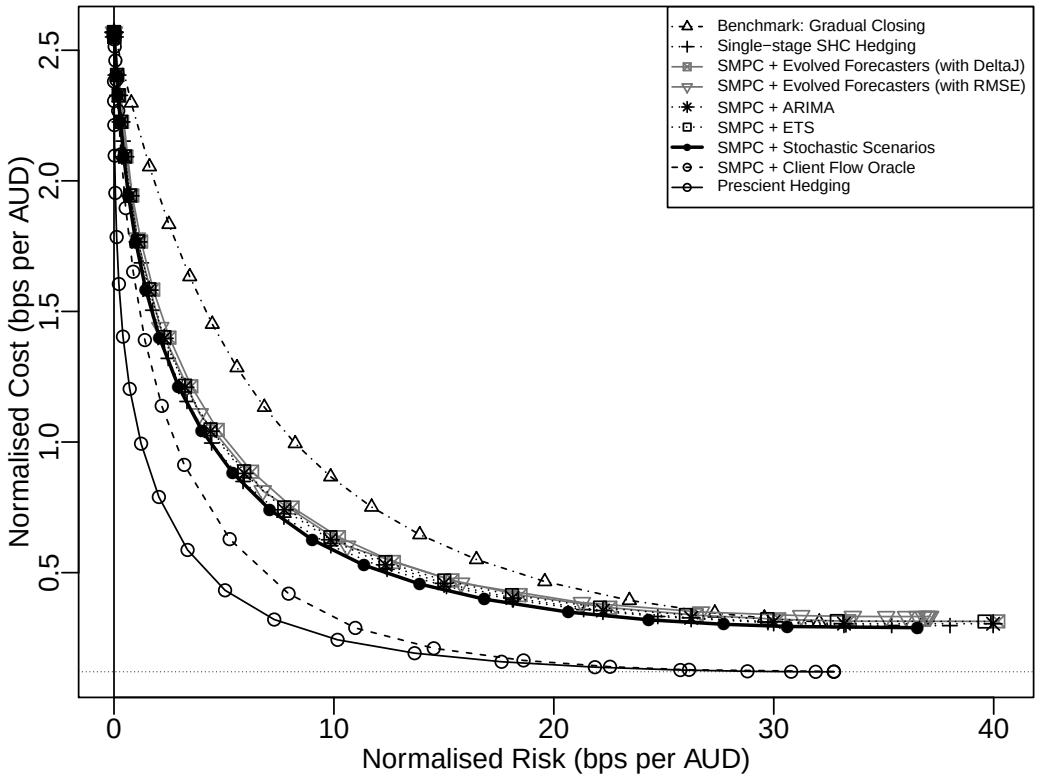


FIGURE 7.16: Effect of prediction on hedging risk-cost profiles. SMPC hedging with stochastic scenario generation is sub-optimal compared to the profiles using prescient data, but outperforms hedging models using time-series forecasting.

- SMPC hedging with client flow oracle; the volatility and market impact coefficient scenarios are generated using stochastic models of Section 7.3.2.
- SMPC hedging with client flow forecasted by ARIMA.
- SMPC hedging with client flow forecasted by ETS.
- SMPC hedging with client flow forecasted by predictors evolved using ΔJ.
- SMPC hedging with client flow forecasted by predictors evolved using RMSE.
- Single-stage shrinking horizon (SHC) hedging algorithm (proposed in Section 3.5).
- Gradual closing strategy (proposed in Section 3.4.4), used as a benchmark.

The results show that SMPC hedging with stochastic model scenario generation is still the best realistic hedging algorithm, and using any forecasts, as expected from the out-of-sample ΔJ errors, only worsens the results.

7.8 Summary

SMPC hedging, proposed and formulated in Chapter 3, was thoroughly tested and verified using both synthetic and real-world data.

TABLE 7.5: Summary of FX hedging improvement using different techniques. The cost is measured at risk = 0.0010 AUD per AUD. The improvements are measured as the percentage of possible cost savings against the prescient hedging and the gradual closing strategy benchmark.

	Cost (bps per AUD)	Improvement (%)
Prescient hedging	0.2473	100.0
SMPC + Client flow foreknowledge	0.3304	86.4
SMPC + Stochastic scenarios	**0.5851**	**44.7**
Single-stage SHC hedging	0.5959	42.9
SMPC + GE forecasts	0.6344	36.6
Benchmark: Gradual closing strategy	0.858	0

First, different stochastic models were proposed, both for generating synthetic examples, and also as benchmark models for fitting real-world data. The SMPC hedging was then compared to other hedging techniques studied in Chapter 3, and was shown to outperform them.

To analyse the weaknesses of the stochastic models and also further verify the SMPC technique, a scenario generating oracle was defined to simulate non-perfect modelling by perturbing the future data. It was shown that by improving the models and increasing their forecasting quality, SMPC hedging results approach that of a prescient hedging algorithm. It was noted that the client flow model, based on a univariate Gaussian process, was the bottleneck of hedging results, and improving client flow prediction could considerably boost the hedging's risk-cost profiles.

Finally, the proposed GE based time-series forecasting technique was applied to the client flow. It was noticed that this technique, in addition to other time-series models, e.g., ETS and ARIMA, were unable to provide any forecasting ability beyond a Gaussian model.

The summary of hedging results, comparing different techniques for real-world data, are presented in Table 7.5. It can be observed that the SMPC hedging, combined with the stochastic scenario generation, outperforms other techniques and offers a 44.7% improvement over the benchmark strategy.

Chapter 8

Renewable Energy Management using Model Predictive Control

8.1 Introduction

Recent advances in battery energy storage systems (BESS) and solar photovoltaic (PV) electricity generation have opened new opportunities for widespread adoption of grid connected renewable energy systems (RES) [235, 236].

Due to the inherent uncertainties of both generation and consumption, optimal control techniques and stochastic models have been proposed to improve RES's performance by controlling the battery's charging and discharging schedule. However, these techniques either assume existence of an accurate forecaster, or utilise simple stochastic models in their methodology. Additionally, the computational cost of these stochastic models and control techniques are often beyond the ability of current microcontroller systems. Consequently, despite intensive research, commercial solutions are still using naïve and inefficient strategies.

In this chapter, the problems of electricity demand management in a grid-connected RES with battery storage is studied. The control objectives are defined as both *peak shaving* (i.e., reducing the peak demand from the grid [237]) and electricity cost minimisation. This requires modelling and forecasting uncertain consumption behaviour and PV electricity generation, as well as considering variable electricity prices and limitations of the battery storage.

In addition to serving as a basis for testing the techniques proposed in Chapters 5 and 6, this chapter offers the following contributions:

- Proposing a customised grammar for forecasting electricity consumption and PV generation using GE.
- A stochastic model predictive control (SMPC) approach towards combined peak shaving and cost saving for grid connected renewable energy systems with battery storage.

8.2 Case Study: TransGrid iDemand

TransGrid iDemand is a renewable energy and electricity demand management system installed at TransGrid's Sydney West site. Commissioned in 2014, iDemand is designed to reduce the site's grid consumption during peak demand [238].

The iDemand system includes (Figure 8.1) [239]:

- Battery storage system: A 400 kWh lithium polymer battery, with a peak power output of 100 kW. This sub-system is able to provide both AC and DC supplies.
- Thin film solar panels: iDemand uses thin film cadmium telluride solar panels to generate renewable electricity. The installed panels are rated at 45 kW.
- Poly-crystalline solar panels: Poly-crystalline solar panels are more efficient and more costly than thin film panels (and in turn, poly-crystalline panels are cheaper but less efficient compared to mono-crystalline panels). iDemand's poly-crystalline installation can generate up to 53 kW of electricity.
- AC and DC LED lighting: In iDemand, metal halide and fluorescent lighting are replaced with the more efficient LED systems. Most of the lights allow automatic dimming in response to daylight conditions.
- Inverters, SCADA, weather station, and web-based portal: Using a supervisory control and data acquisition (SCADA) [240] system, inverters are controlled to allow custom flow of energy. The data, including the system's overall performance and also the site's weather conditions, are collected and made available[1] through the iDemand website [238].

iDemand's SCADA and inverters allow dynamic programming of the battery charge and discharge profile. As of November 2014, the system has been programmed to "set the battery to charge overnight and then discharge on weekdays between 1pm and 6pm" [239]. The rationale behind this is to reduce grid consumption (which can go "up to 400 kW in summer") during peak demand.

[1] TransGrid terms and conditions permit personal, in-house, or non-commercial use of their data without formal permission [241].

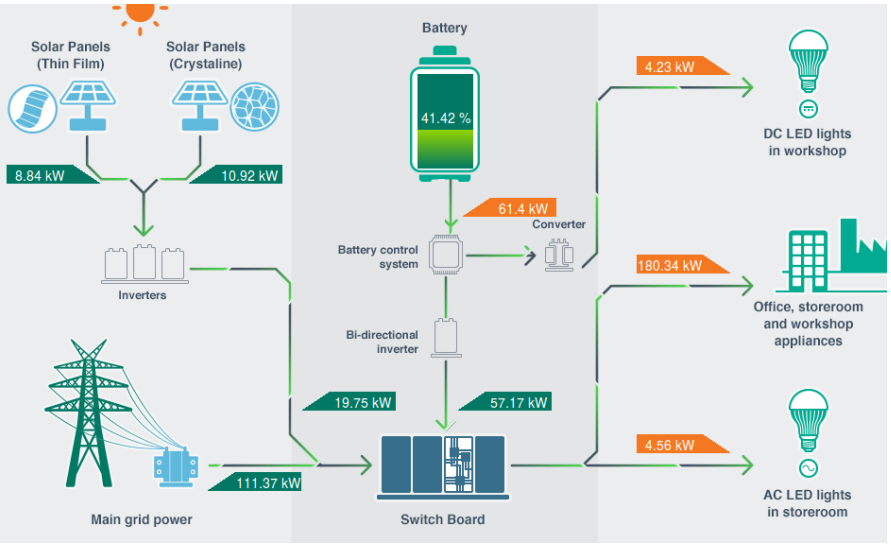


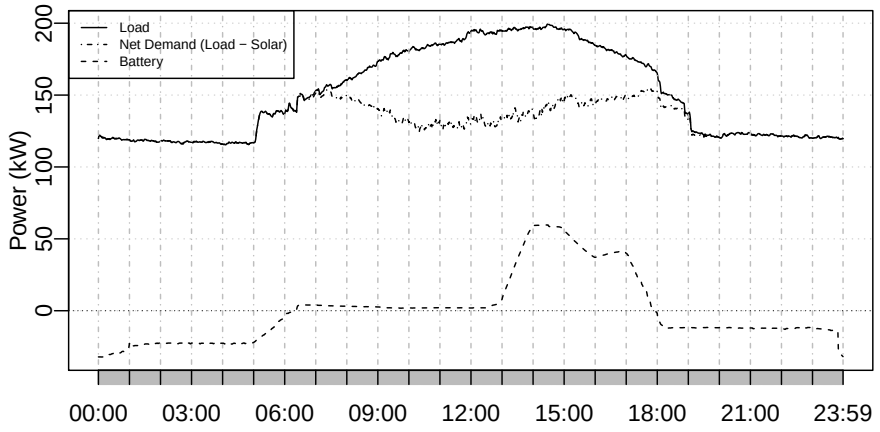
FIGURE 8.1: A screenshot of TransGrid iDemand's public web portal [238], showing real-time power flow. iDemand uses two different solar panel types (thin film PV and crystalline PV), in addition to the power from the grid, so as to supply power for offices and workshops. A 400 kWh battery stores the energy and discharges during peak demand to reduce power from the grid. Separate AC and DC LED lighting systems are also installed to increase lighting efficiency. In this instance, the total consumption of 189.13 kW is satisfied with only 111.37 kW from the grid.

8.2.1 Improving iDemand's Control Policies

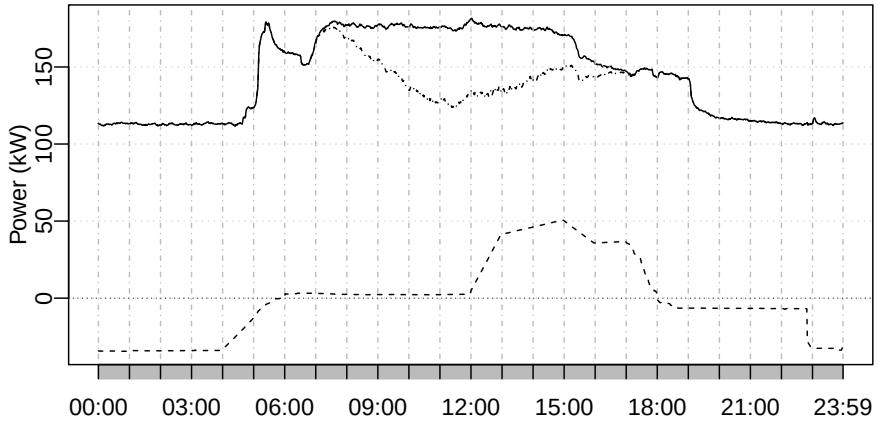
An analysis of the current iDemand load and its static battery schedule shows that it is far from reaching its goal. For example, Figure 8.2 shows that the peak demand, considering the power from solar panels, does not necessarily occur between 13:00 and 18:00. Additionally, the workshop load profile changes with time, with a new peak appearing in the 2015 second quarter's morning load.

In this chapter, stochastic receding horizon control will be used to reduce the grid consumption and shave load peaks by optimally scheduling the battery charge and discharge rates. This problem combines three different challenges:

1. Forecasting solar power generation prediction: While mostly correlated with cloud cover, many uncontrolled parameters, including shadows from nearby structures, inclination of the PV plane and its variability, and meteorological events such as haze and fog vary the PV generation.
2. Forecasting load: The site load, while following a daily and weekly cycle, is subject to high variations, influenced by many predictable (e.g., weather) and unpredictable events (e.g., new constructions or extra work shifts).
3. Control system: Many factors have to be considered in the control algorithm, including but not limited to battery capacity, battery charging maximum safe rate,



(A) First quarter (summer) 2015



(B) Second quarter (autumn) 2015

FIGURE 8.2: The mean daily load, net demand (i.e., load consumption minus solar generation), and battery power in TransGrid iDemand. Negative battery power denotes charging.

variation of grid price with time, and uncertainties of generation and consumption prediction.

Although the consumption profile changes with time, it shows a strong daily and weekly cycle (Figure 8.3). Furthermore, considering the near flat profile of net demand (site and LED load consumption minus solar generation, as in Figure 8.2), one can aim to not only reduce possible peaks, but also reduce costs by managing grid demand during periods of high electricity price.

Specifically, this work combines automatic time-series prediction techniques with a domain-expert designed grammar, to provide better forecasts as inputs to the SMPC controller.

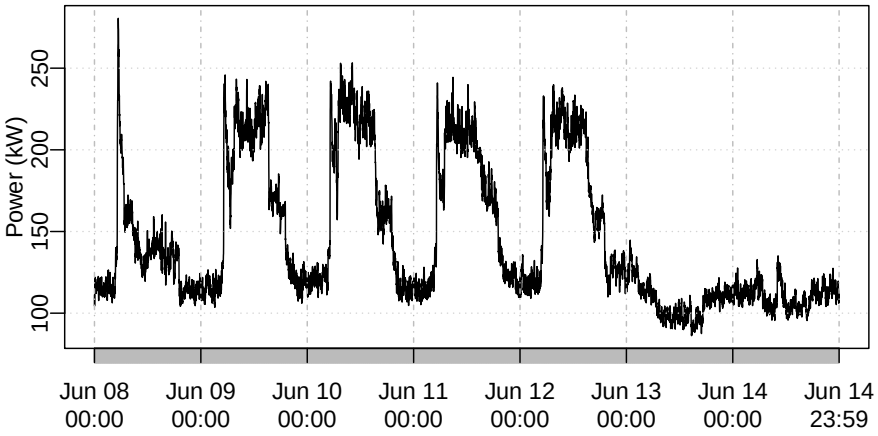


FIGURE 8.3: A week of TransGrid iDemand load. Notice that 8 June 2015 was a public holiday in New South Wales, Australia.

8.3 Optimal Scheduling for the iDemand System

In this section, SMPC is used to develop a strategy for optimally scheduling iDemand's charge/discharge cycles.

8.3.1 iDemand Dynamics

The iDemand battery's state of charge can be modelled using the following discrete state-space model:

$$b_{t+1} = b_t + g_t + s_t - l_t \tag{8.1}$$

Here,

- b_t is the battery's state of charge at time t,
- s_t is the solar PV generation power,
- l_t is the load power, and
- g_t is the power demanded from the grid.

The battery's capacity is limited to b_{max}:

$$0 \leq b_t \leq b_{max} \tag{8.2}$$

The battery charge and discharge rate is determined by

$$u_t = b_{t+1} - b_t,$$

and is limited to u_{max}:

$$|u_t| \leq u_{max}$$

The grid operator charges p_t for a unit of electricity at time t, resulting in the total cost, C, formulated as

$$C = \sum_i g_i p_i, \tag{8.3}$$

and does not buy electricity back, i.e.,

$$g_t \geq 0.$$

We define the objective of the iDemand control system as reducing costs and shaving peaks. Considering the stochastic nature of the system's variables, this is formalised as

$$\begin{aligned} &\underset{u_t;\ \forall t}{\text{argmin}} && \mathbb{E}\left[\sum g_i^2\right] + \lambda \mathbb{E}[C] \\ &\text{subject to} && |u_t| \leq u_{max} \end{aligned} \tag{8.4}$$

where the parameter $0 \leq \lambda \leq \infty$ selects between peak shaving when $\lambda = 0$, and cost saving when $\lambda \to \infty$. These goals, while not directly in opposition, may lead to different results if p_t varies with time.

8.3.2 Open-loop Optimisation

Assuming that battery storage constraint (8.2) holds, the grid consumption can be obtained from

$$g_t = u_t + l_t - s_t.$$

Notice that as $g_t \geq 0$, this may result in infeasible solutions if $s_t - l_t > u_{max}$, i.e., when local generation is more than what can be stored on battery. The alternative, which is explored in Appendix C, is to use

$$b_{t+1} = b_t + g_t + s_t - l_t + w_t,$$

where $w_t \leq 0$ is a slack variable to denote the energy wasted to satisfy the battery capacity constraint (8.2). As this never happens in the iDemand system and only adds to the complexity of the solution, this alternative was not implemented.

By replacing the grid cost with (8.3), the cost function of (8.4) can be reorganised as follows:

$$
\begin{aligned}
J &= \mathbb{E}\left[\sum g_i^2\right] + \lambda \mathbb{E}[C] \\
&= \mathbb{E}\left[\sum_i g_i^2\right] + \lambda \mathbb{E}[\sum_i g_i p_i] \\
&= \sum_i \mathbb{E}\left[g_i^2 + \lambda g_i p_i\right] \\
&= \sum_i \mathbb{E}\left[(u_i + l_i - s_i)^2 + \lambda(u_i + l_i - s_i)p_i\right] \\
&= \sum_i \mathbb{E}\left[(u_i^2 + 2u_i(l_i - s_i) + (l_i - s_i)^2) + \lambda((l_i - s_i)p_i + u_i p_i)\right] \\
&= \sum_i \mathbb{E}\left[u_i^2 + 2u_i(l_i - s_i + \frac{1}{2}\lambda p_i)\right] + \sum_i \mathbb{E}\left[(l_i - s_i)^2 + \lambda(l_i - s_i)p_i\right]
\end{aligned}
$$

As u_t is the only controllable variable and deterministic, this simplifies to

$$
J = \sum_i \left(u_i^2 + 2u_i \mathbb{E}\left[l_i - s_i + \frac{1}{2}\lambda p_i\right]\right).
$$

Similarly, the constraints on u_t and g_t can be rewritten as

$$
\begin{aligned}
-u_t &\leq l_t - s_t, \\
-u_t &\leq u_{max}, \\
u_t &\leq u_{max}.
\end{aligned}
$$

The battery energy state can be expressed recursively as

$$
b_t = b_0 + \sum_{i=0}^{t-1} u_i.
$$

Hence, the constraint on battery energy is replaced with

$$
\begin{aligned}
& 0 \leq b_t \leq b_{max} \\
\longrightarrow\ & 0 \leq b_0 + \sum_{i=0}^{t-1} u_i \leq b_{max} \\
\longrightarrow\ & -b_0 \leq \sum_{i=0}^{t-1} u_i \leq b_{max} - b_0.
\end{aligned}
$$

With the constraints and cost function transformed as a function of battery charge rate u_t, optimisation of (8.4) can be rewritten as

$$\begin{aligned}
\underset{u_t;\ \forall t}{\text{argmin}} \quad & \sum_{i=0}^{N} \left(u_i^2 + 2u_i \mathbb{E}\left[l_i - s_i + \frac{1}{2}\lambda p_i \right] \right) \\
\text{subject to} \quad & -\sum_{i=0}^{t-1} u_i \leq b_0 \\
& \sum_{i=0}^{t-1} u_i \leq b_{max} - b_0 \\
& -u_t \leq l_t - s_t \\
& -u_t \leq u_{max} \\
& u_t \leq u_{max}
\end{aligned} \quad .$$

Similar to Chapter 3, this optimisation is vectorisable:

$$\begin{aligned}
\underset{u_t;\ \forall t}{\text{argmin}} \quad & \boldsymbol{u}^{\mathrm{T}}\boldsymbol{u} + 2\boldsymbol{u}^{\mathrm{T}} \left(\mathbb{E}\left[\boldsymbol{l} - \boldsymbol{s} + \frac{1}{2}\lambda \boldsymbol{p} \right] \right) \\
\text{subject to} \quad & -\boldsymbol{\Sigma}\boldsymbol{u} \leq b_0 \vec{\mathbf{1}} \\
& \boldsymbol{\Sigma}\boldsymbol{u} \leq (b_{max} - b_0)\vec{\mathbf{1}} \\
& -\boldsymbol{u} \leq \boldsymbol{l} - \boldsymbol{s} \\
& -\boldsymbol{u} \leq u_{max}\vec{\mathbf{1}} \\
& \boldsymbol{u} \leq u_{max}\vec{\mathbf{1}}
\end{aligned} \tag{8.5}$$

where

$$\begin{aligned}
\boldsymbol{l} &= [l_0\ l_1\ \cdots\ l_N]^{\mathrm{T}}, \\
\boldsymbol{s} &= [s_0\ s_1\ \cdots\ s_N]^{\mathrm{T}}, \\
\boldsymbol{p} &= [p_0\ p_1\ \cdots\ p_N]^{\mathrm{T}},
\end{aligned}$$

$\vec{\mathbf{1}}$ is a vector of 1's, and $\boldsymbol{\Sigma}$ is a lower triangular matrix of 1's (i.e., without diagonal elements).

8.3.3 Parameter Requirements

Solving (8.5) requires the following time-varying parameters:

- $\mathbb{E}[\boldsymbol{l}]$: Expected value of the total load for the next N steps. In iDemand, the load includes the site load, the DC LED load, and the AC LED load.

- $\mathbb{E}[\boldsymbol{s}]$: Expected value of the total renewable generation for the next N steps. In iDemand, this translates to sum of solar generation for the thin film and the poly-crystalline PV panels.
- $\mathbb{E}[\boldsymbol{p}]$: Expected value of the total electricity price for the next N steps. In this chapter, we assume that the retail cost of grid electricity, while variable, is known in advance at least for 24 hours ahead.

Considering the daily cycle of charge/discharge in iDemand, we select N as the next 24 hours.

Given this information, the stochastic constraints, i.e., $-\boldsymbol{u} \leq \boldsymbol{l} - \boldsymbol{s}$, can be handled using the technique explained in Section 3.6.2. Subsequently, quadratic programming (QP) can be used to minimise (8.5). In the case of $\lambda \to \infty$ (i.e., cost saving only), the cost function is further simplified to $J = \boldsymbol{u}^{\mathrm{T}}\boldsymbol{p}$, which is a linear programming problem.

8.3.4 Open-loop Simulation Results

To test the open-loop optimisation proposed in (8.5), a synthetic test was performed. The length of each time-step was selected as 15 minutes, resulting in $N = 96$ for a 24 hour simulation. The net demand, i.e., $\boldsymbol{l} - \boldsymbol{s}$, was generated using a half sinusoidal wave, peaking at midday. Constraints were set to $u_{max} = 0.5$ and $b_{max} = 5$. The initial battery charge was set to $b_0 = 2$.

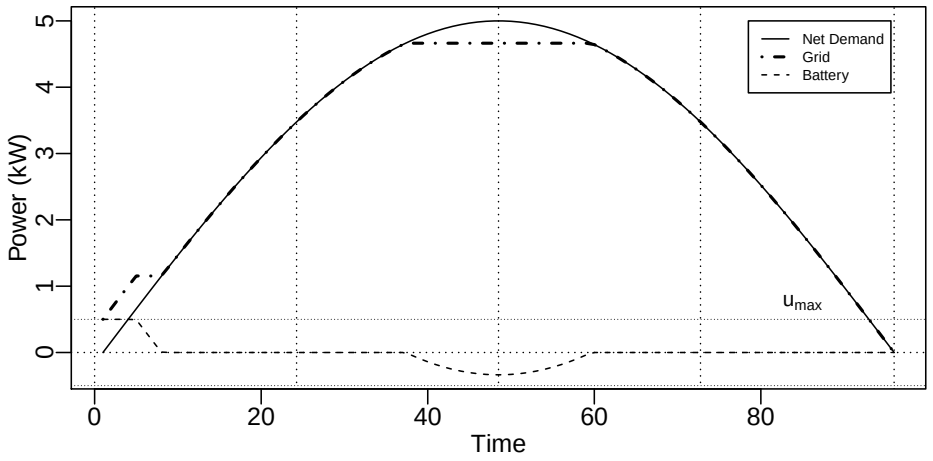
For the first test, only peak shaving was simulated by setting $\lambda = 0$. The results, presented in Figure 8.4, show that the controller charges the battery during the early low demand period, subject to the constraints, and then discharges the battery such that the peak demand from the grid is *shaved*.

For the second test, cost saving with $\lambda = 1000$ was explored. The variable grid cost per energy unit, $\boldsymbol{p}$, was chosen with a minima during high demand and two peaks at other times, as shown in Figure 8.5a. The same demand as in the first test was used. The battery's state of charge is shown in Figure 8.5b. It can be observed that the controller charges the battery during low price periods, regardless of the grid demand, while discharging at full rate during high price periods (Figure 8.5c).

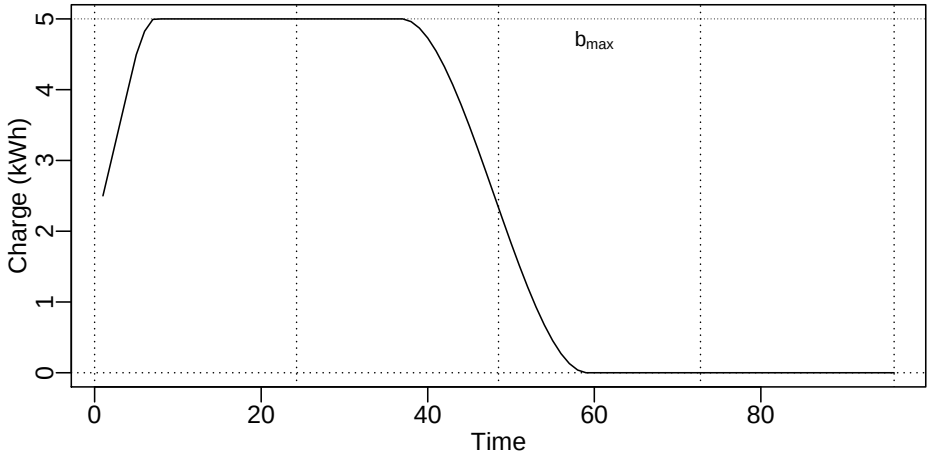
The open-loop optimisation, using the R package *quadprogpp* and with $N = 96$, takes 4 milliseconds on a 3.4 GHz Intel Core i7-2600 processor.

8.4 Automatic Load and Generation Forecasting

As determined in the previous section, the controller requires forecasting these values:



(A) Net demand, power from the grid, and battery charge/discharge rate.



(B) Battery state of charge.

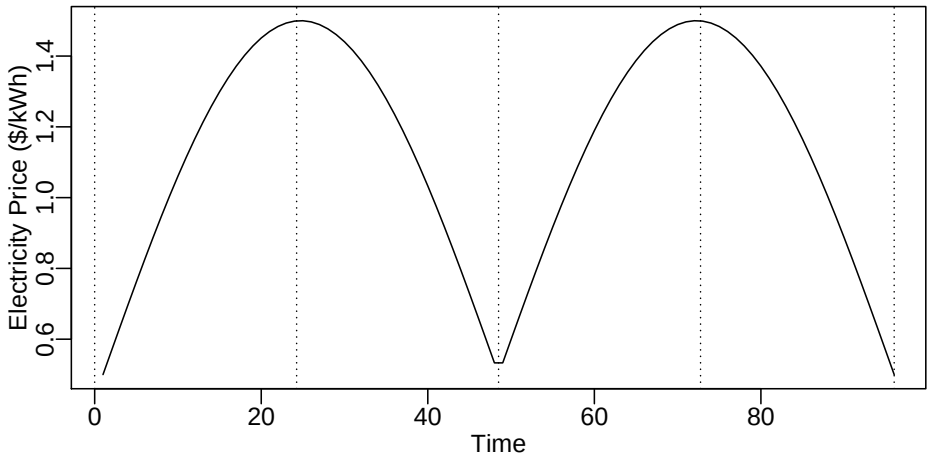
FIGURE 8.4: Open-loop battery schedule optimisation for peak shaving.

- Load, which is composed of
  - the site load, l_{site},
  - DC LED lighting load, l_{dc}, and
  - AC LED lighting load, l_{ac}.
- The renewable energy generated by
  - thin film solar panels, s_{tf}, and
  - poly-crystalline solar panels, s_{pc}.

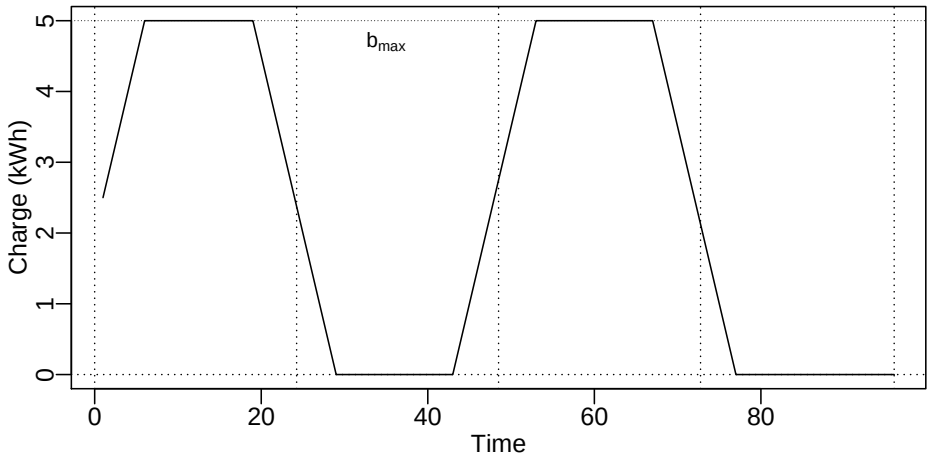
From December 2014 onwards, the AC LED lighting load profile shows a constant 2.64 kW consumption, and consequently l_{ac} was not analysed.

8.4.1 Historical and External Data

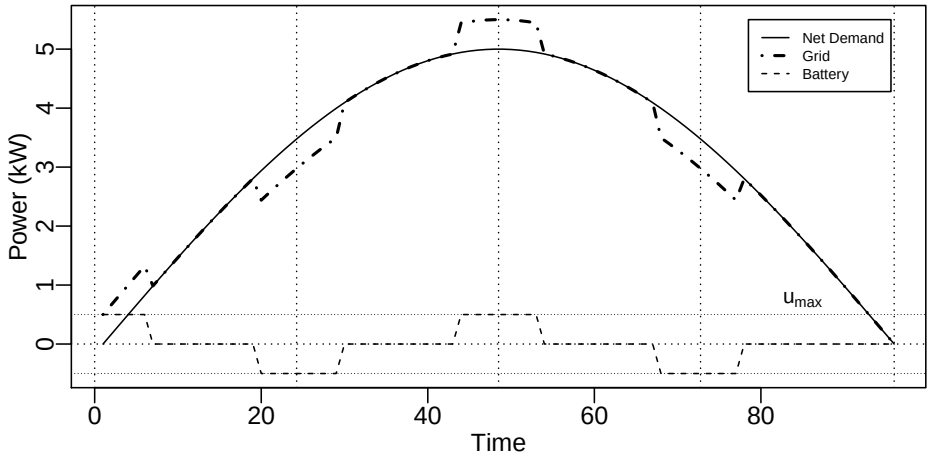
The historical load and solar data were aggregated to 15 minutes to improve training and prediction speed. For a fair comparison with the current iDemand static schedule



(A) Grid cost per energy unit.



(B) Battery state of charge.



(C) Net demand, power from the grid, and battery charge/discharge rate.

FIGURE 8.5: Open-loop battery schedule optimisation for cost saving.

TABLE 8.1: Numeric scores for descriptive sky conditions.

Condition	Numeric score
Clear	5
Scattered clouds	4
Partly cloudy	3
Mostly cloudy	2
Overcast	1
Other (e.g., rainy)	0

which is active only during weekdays, weekend data were discarded.

In addition to the historical generation and consumption, weather forecasts were also used as explanatory variables for predicting loads and PV generation. As ex-ante (i.e., historical) forecasts were not available, the ex-post (i.e., actual) observations from two nearby airports, Sydney Kingsford Smith Airport (40 km south-east of the site), and Richmond Air Force Base (30 km north of the site) were downloaded from the publicly[2] available WeatherUnderground website [243].

The temporal resolution of these observations is 30 minutes, and includes ambient temperature, humidity, and descriptive sky conditions (e.g., clear, cloudy, or partly cloudy), but doesn't report the cloud cover in percentage. To include these descriptive values in forecasting models, Table 8.1 was used to map the conditions to integer scores.

Notice that better data can be obtained from commercial services specifically designed for predicting solar power, which is expected to improve the forecast results. Still, the goal of this work is not to contribute an expert tuned model for the problem of day-ahead solar generation prediction, but to compare automatic time-series forecasting techniques using the available data.

8.4.2 Prediction Grammar

To model and predict l_{site}, l_{dc}, s_{tf}, and s_{pc}, the methodology proposed in Chapter 5 was used. Specifically, a grammar was first designed, and then using grammatical evolution the best prediction function was selected.

[2] Weather Underground, Inc., the owner of WeatherUnderground.com, permits personal, non-commercial use of their data without subscription [242].

8.4.2.1 Pre-processing

Pre-processing grammar was designed according to the methodology described in Section 5.3.2. The most significant change was adding a new logarithmic variance stabilisation function. The logarithm function approaches infinity as the data leans towards zero, which is observed in both solar generation (during night time), and LED load (at non-working hours). Hence, a scaling term was embedded within the log function to allow correct and reversible pre-processing.

Additionally, the daily and weekly cycles of the data with 15-minute intervals are known, being 96 and 480 samples, respectively. These values were added to the frequency list, and the FFT option was removed.

8.4.2.2 Features

Feature generation grammar was also implemented as described in Section 5.3.3, with both auto-regressive and exogenous features. The details of these features are presented in Table 8.2.

8.4.2.3 Learner Models

Learner model grammar was implemented as described in Section 5.3.4. Two learning techniques were used:

- A linear model, as commonly used in electricity models (e.g., the Monash electricity model [244]).

TABLE 8.2: Candidate features for predicting electricity consumption and generation in iDemand.

Feature	Window type	Maximum width	Maximum lag
Auto-regressive	Past lags	N/A	16
Daily cycle auto-regressive	Past lags, centred	4	15
Weekly cycle auto-regressive	Past lags, centred	5	5
Kingsford airport temperature	Centred	8	N/A
Kingsford airport humidity	Centred	8	N/A
Kingsford airport cloud cover score	Centred	8	N/A
Richmond base temperature	Centred	8	N/A
Richmond base humidity	Centred	8	N/A
Richmond base cloud cover score	Centred	8	N/A
Day-of-week	Binary vector	N/A	N/A

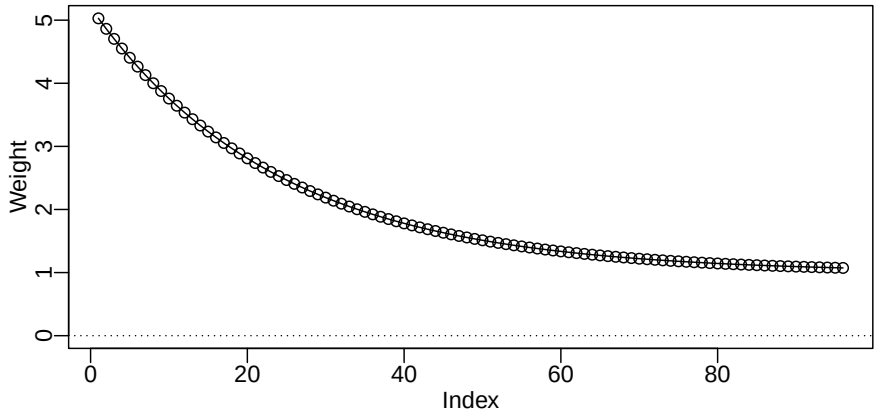


FIGURE 8.6: Fitness function error weights for 96 steps ahead predictions.

- A SVM with radial basis function kernel. Hyper-parameters of the support vector machine include ϵ, γ, and C [234].

8.4.2.4 Fitness Function

The fitness function was implemented as described in Section 5.4.3. The period from Monday 2015-01-15 to Friday 2015-05-01 (15 weeks) was selected for cross-validation, with the first ten weeks of data as the initial training window, and the rest tested using an expanding rolling window scheme. The cross-validation errors of the rolling windows were aggregated and reported as the fitness score, as proposed in Algorithm 5.4.

As the problem described in this chapter is not strictly following a classic linear-quadratic control formulation, the error measure proposed in Chapter 6 was not directly used. Inspired by its result, weighted mean square error was used instead, with weights exponentially decaying with time according to

$$w_t = \frac{e^{4(1-\frac{t}{N-1})+1}}{\sum_{j=0}^{N-1} e^{4(1-\frac{j}{N-1})+1}}.$$

The error weights versus prediction horizon are illustrated in Figure 8.6. The weights allow the cost function to be more sensitive to the inaccuracies of near future predictions, while also considering far future errors.

8.4.2.5 Grammatical Evolution

Grammatical evolution was implemented using the package *gramEvol* [11]. The number of generations was set to 50, and the rest of parameters were determined automatically

by the package. More details about the architecture of this package are available in Appendix A.

8.4.3 Comparison with Other Techniques

In addition to the evolved grammar predictors, the following techniques were also employed:

- Seasonal naïve predictor: Data from the same time of day of the last working day is repeated to create a forecast.
- Averaging model predictor: Measurements for the same time of day for the last 15 days are averaged and reported as forecasts. This technique is widely used in energy forecasting applications [245].
- ETS and ARIMA: The R package *forecast* [200] was used to create ARIMA and ETS models. As the package only supports cycles with a frequency of less than 24, the data was down-sampled to hourly intervals before being given to the package, and the forecasts were then linearly interpolated to 15 minutes. The model orders were determined automatically by the package using the same period selected for training the grammar based models. To reflect data behaviour changes in the model training, for each prediction, model parameters were fit only to the most recent ten days of data.
- ARIMAX: The ARIMAX model from the package *forecast* was used with Richmond Air Force Base weather forecasts as exogenous inputs. The methodology of model order selection and parameter fitting was similar to ARIMA.

Despite the availability of other methods for load forecasting in literature [13, 244, 246, 247], no direct comparison with these techniques were made. The reason falls under one or more of the following categories:

- The structure of these techniques was already covered in the proposed grammar (e.g., using ANNs instead of SVM as a learner).
- Exogenous variables were used which are not relevant here (e.g., wind speed, local population, or GDP).
- The proposed models were tuned to a specific dataset (e.g., Kaggle forecast competition, or Southern Australia state-wide consumption).
- The objective of the forecast was different from the expected value of day-ahead step-wise load (e.g., only predicting the daily peak).
- The iDemand system only started recording data from October 2014, which is not enough time for some of techniques that deal with variation of the load over a decade.

8.5 Results

8.5.1 Forecasts

The results of model selection using the proposed evolutionary technique are presented in Table 8.3. Some selected model features are intuitive:

- The load (both the site's and the LED's) shows strong correlation with the recent consumption data, while the thin film PV is independent of the recent events (i.e., the selected order is $p = 0$).
- Solar generation is highly correlated with weather conditions.
- The load is correlated with humidity, which can be indicative of air-conditioning, as human perception of temperature depends on humidity as well as rainy conditions.

Some selected features are counter-intuitive:

- None of the predictors use differential or logarithmic pre-processing, unlike the technique proposed in [244]. Notice that the leaner models have their own built-in scaling and centring steps.
- The week-of-day indicator binary matrix, which is expected to be only influential on the load data, is selected for the solar generation instead.
- Predicting load uses simpler linear models instead of the more powerful SVM with radial basis function kernel.

Table 8.4 compares the prediction error using different techniques for the site load, the DC LED lighting load, the thin film panels' generation, and the crystalline panels' generation. The comparison includes the in-sample period, i.e., the period used to train and validate the grammar, and the out-of-sample period, i.e., eight weeks from Monday 2015-05-04 to Friday 2015-06-24. The error measures include:

- The in-sample error obtained from the fitness function used in grammatical evolution.
- Weighted mean square error (WMSE), exponentially weighted as described for the fitness function, and averaged for every 96-step-ahead prediction of every time-step.
- Root mean square error (RMSE), averaged for every 96-step-ahead prediction of every time-step.
- Mean absolute error (MAE), averaged for every 96-step-ahead prediction of every time-step.

As expected, the *evolved* predictor outperforms other techniques for the in-sample fitness, and consequently out-of-sample WMSE error. In the case of out-of-sample RMSE

TABLE 8.3: Selected features for predicting electricity consumption and generation in iDemand. A dash denotes the feature not being selected, and p and q determine the order of lags and window widths respectively. With $q = 0$, only the value at the centre of the window is selected.

Feature	Site load	DC LED load	PV thin-film	PV crystalline
Pre-processing	-	-	-	-
Auto-regressive	p = 13	p = 3	p = 0	p = 14
Daily cycle	p = 4	p = 3	p = 14	p = 2
auto-regressive	q = 4	q = 3	q = 3	q = 2
Weekly cycle	p = 1	p = 2	-	p = 4
auto-regressive	q = 3	q = 0		q = 3
Kingsford airport				
- temperature	-	-	q = 6	q = 6
- humidity	q = 2	-	q = 0	q = 7
- cloud cover score	-	-	q = 5	q = 8
Richmond base				
- temperature	-	-	-	q = 8
- humidity	q = 3	q = 6	q = 8	q = 8
- cloud cover score	-	-	q = 3	-
Day-of-week	-	-	Selected	-
Learner	Linear	Linear	SVM	SVM

and MAE; however, the evolved predictors only produce better results for PV generation compared to other techniques, and load prediction favours naïve or averaging techniques.

This performance difference is a result of the 15-day average model being a better predictor for the far future, and the grammar-based predictor performing better in the near future forecasts. Figure 8.7 illustrates this far versus near accuracy, where the 24-hour ahead prediction of site load for the averaging model and the evolved predictor are compared using the dashed line and the dark grey line, respectively. It can be observed that the averaging model outperforms the evolved predictor; but while the averaging model's forecasts do not change with time, the evolved predictor reuses the recent observations through its auto-regressive component, and updates its prediction accordingly. Eight of these updated forecasts, performed at 2 hour intervals and predicting the next 8 steps, are shown using different symbols in Figure 8.7. The rest of the day-ahead predictions for each of these models are shown using light grey lines. It is evident that these updated forecasts outperform the averaging model, especially in near term prediction. Consequently, given that the error difference between two techniques is minimal for RMSE (16.676 versus 16.426), and considering that the controller is more influenced by near

TABLE 8.4: Day-ahead forecast error of electricity demand and solar generation in iDemand for different prediction techniques.

Prediction Technique	In-sample Fitness	WMSE	RMSE	MAE
Evolved	**13.396**	**633.634**	16.676	12.365
Naïve	16.594	895.059	18.831	13.363
Averaging	17.141	679.921	**16.426**	**11.976**
ARIMA	22.226	1474.053	27.615	22.674
ARIMAX	21.229	1475.270	27.627	22.318
ETS	15.680	872.033	19.931	14.734

(A) Day-ahead site load forecast errors.

Prediction Technique	In-sample Fitness	WMSE	RMSE	MAE
Evolved	**0.734**	**1.555**	0.841	0.665
Naïve	0.774	2.009	**0.83**	**0.552**
Averaging	0.977	1.991	0.940	0.769
ARIMA	1.150	2.878	1.179	0.979
ARIMAX	1.151	2.973	1.207	1.003
ETS	1.142	2.966	1.193	0.940

(B) Day-ahead DC LED lighting load forecast errors.

Prediction Technique	In-sample Fitness	WMSE	RMSE	MAE
Evolved	**4.628**	**22.501**	**3.175**	**1.788**
Naïve	6.833	45.133	4.163	1.871
Averaging	6.778	47.267	4.539	2.265
ARIMA	7.153	75.771	6.013	3.152
ARIMAX	6.128	61.967	5.447	4.210
ETS	137.165	182779.048	98.430	26.306

(C) Day-ahead poly-crystalline solar panel generation forecast errors.

Prediction Technique	In-sample Fitness	WMSE	RMSE	MAE
Evolved	**4.789**	**19.824**	**2.949**	**1.719**
Naïve	6.760	42.250	4.041	1.853
Averaging	6.845	44.529	4.394	2.307
ARIMA	7.502	77.851	6.257	4.549
ARIMAX	5.842	54.978	5.167	4.060
ETS	8.857	63.309	5.262	3.687

(D) Day-ahead thin film solar panel generation forecast errors.

term forecasts, as demonstrated in Chapter 6, one can expect better control using the evolved predictors.

Additionally, in the case of the poly-crystalline solar panel generation (s_{pc}), ETS technique diverges from cyclic prediction, resulting in very high errors. To allow ETS to be used, solar generation forecasts' range was bound to the capacity of the solar panels,

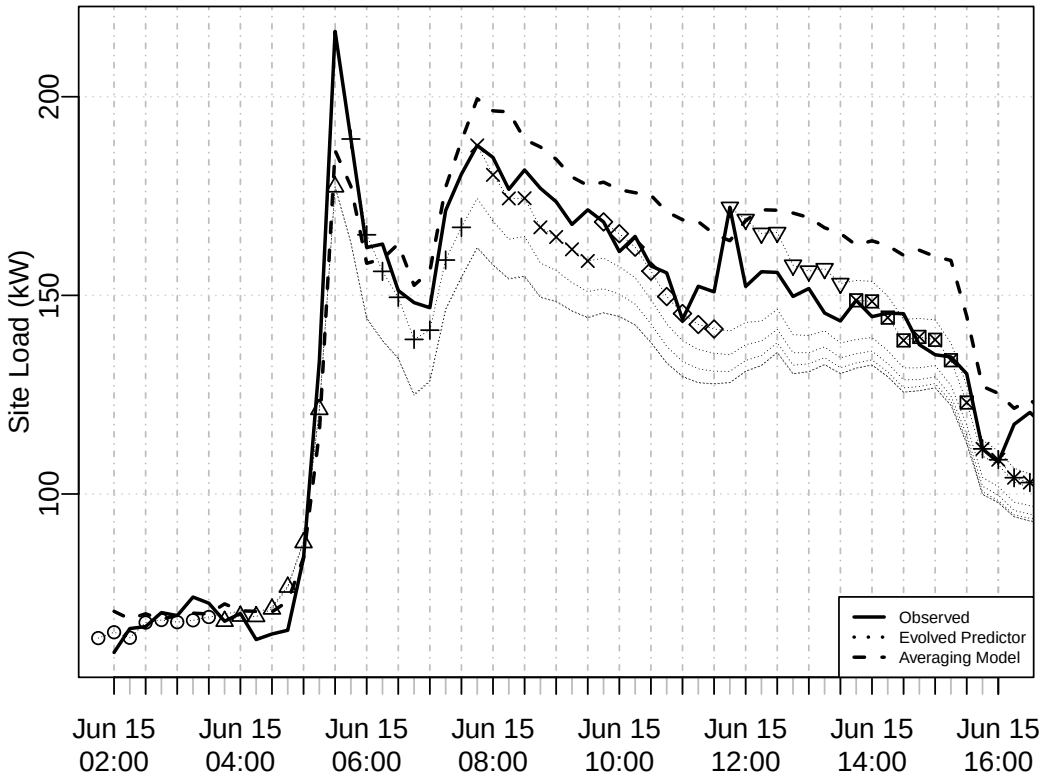


FIGURE 8.7: An example of prediction accuracy for the evolved predictor and the averaging model technique. The averaging model's forecast (dashed line) is closer to the observed site load (solid line) compared to the evolved predictor's results obtained at 02:00 (dark grey). In comparison, the eight-step-ahead predictions performed by the evolved predictor at different times of day, depicted by different symbols, are closer to the observed load. The light grey lines show the progression of each of these forecast towards the end of the day.

through

$$\min(\max(\hat{s}, s_{max}), 0)$$

where $\hat{s}$ is the forecast, s_{max} is the panels' rated power, and 0 denotes no generation (i.e., the minimum at night time).

The net demand prediction was obtained by aggregating separately predicted parameters:

$$demand = \hat{l} - \hat{s} = \hat{l}_{site} + \hat{l}_{dc} - \hat{s}_{tf} - \hat{s}_{pc}$$

The results are presented in Table 8.5. It can be observed that the proposed methodology, using evolved grammar based predictors, outperforms other techniques over all error measures.

TABLE 8.5: Day-ahead demand forecast error for different prediction techniques.

Prediction Technique	WMSE	RMSE	MAE
Evolved	**721.201**	**18.143**	**13.348**
Naïve	1082.581	21.090	14.899
Averaging	916.595	19.758	14.327
ARIMA	1444.003	26.987	21.870
ARIMAX	1488.393	27.587	22.146
ETS	1127.730	22.861	16.863

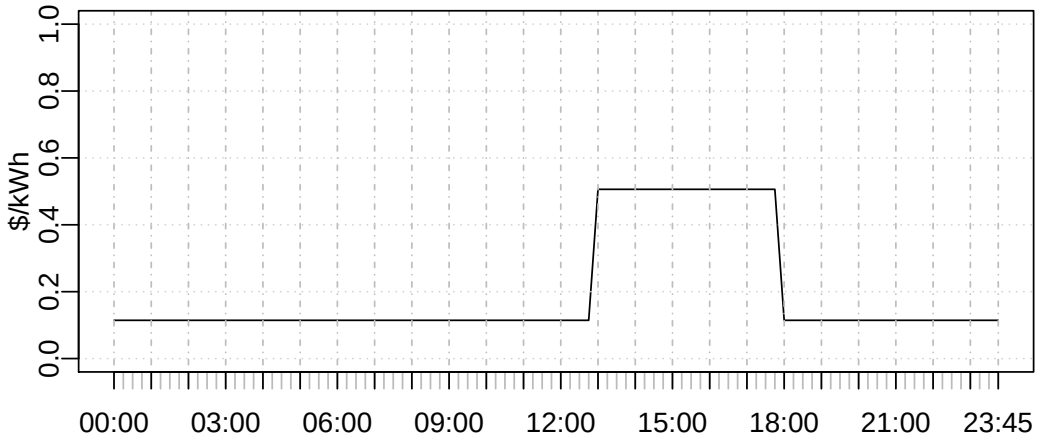


FIGURE 8.8: Synthetic price per kWh for testing iDemand peak shaving controller.

8.5.2 Controller Simulation

To allow a fair comparison with the current static schedule of iDemand, a peak versus off-peak pricing scheme was defined for the 13:00 to 18:00 time period, using the residential prices of Origin Energy Retail distributed by AusGrid [248]:

$$p_t = \begin{cases} 0.50611 \ \frac{\$}{kWh} & 13{:}00 \leq t \leq 18{:}00 \\ 0.1144 \ \frac{\$}{kWh} & \text{else} \end{cases}$$

Figure 8.8 shows the daily cycle of this pricing scheme[3].

This pricing scheme is to obtain a Pareto frontier of peak shaving against cost saving profiles by varying the λ parameter of the controller cost function (8.5); in comparison, the static schedule only produces one solution, resulting in a single profile.

Additionally, the static schedule is limited to discharge to no less than 20% of the battery capacity, and the charge rate is limited to 90.2% of its maximum rating. Hence,

[3] The Origin Powersmart domestic terms of use is based on a three tiered pricing scheme plus daily supply charges. The above assumption only uses two of these prices as a guideline.

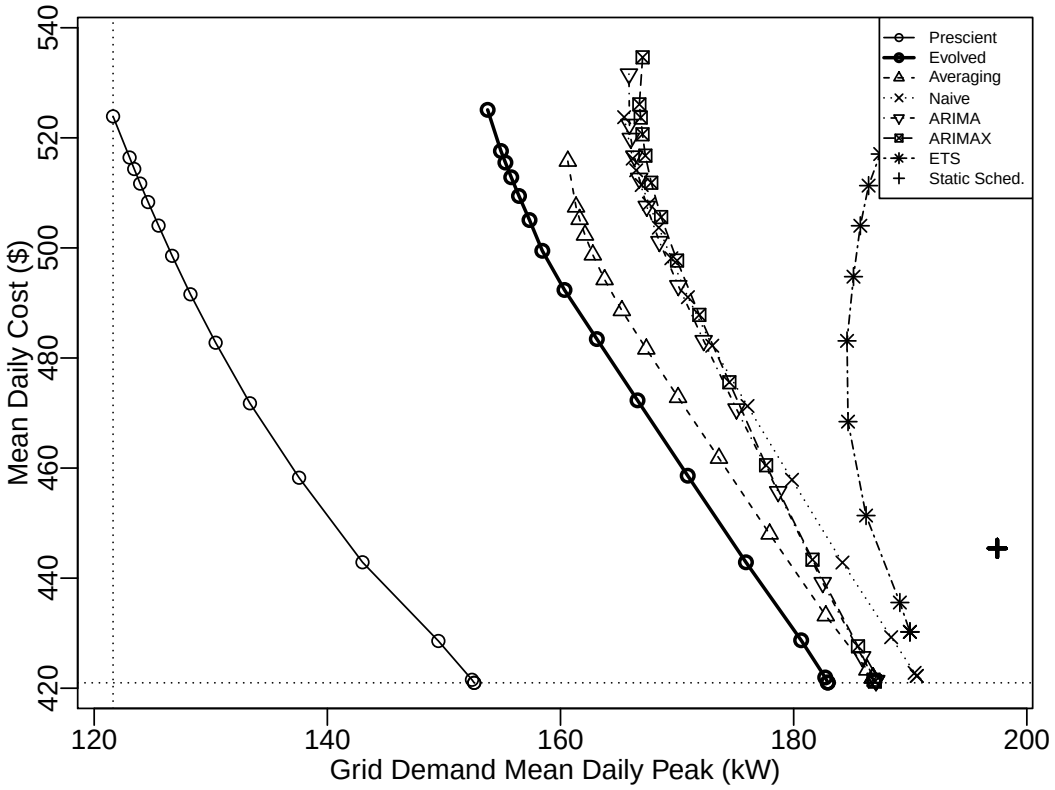


FIGURE 8.9: Mean daily peak demand versus cost using time-series forecasting and predictive control.

the following constraints were used in the simulation:

$$b_{max} = 1360\ kWh \times \frac{15\ min}{hour} \qquad u_{max} = 91\ kW$$

The initial battery charge was set to 50% of the battery capacity, i.e., $b_0 = \frac{1}{2} b_{max}$. The simulation was performed on the same period where the prediction techniques were tested, i.e., Monday 2015-05-04 to Friday 2015-06-24.

The results of simulation, in the form of daily peak grid demand against daily grid cost averaged over the testing period, are shown in Figure 8.9. It can be seen that while grammar based evolved predictor is outperforming the rest of the techniques, it is still far from the optimal frontier obtained using a prescient forecaster.

In comparison, if the peak is computed at hourly intervals, as shown in Figure 8.10, a better shaving is observed. Additionally, the second best peak shaving technique would be the ARIMA and ARIMAX predictors instead of the averaging model.

Comparable results are obtained if the standard deviation of grid demand is plotted against daily cost, using the original 15-minute intervals, as presented in Figure 8.11.

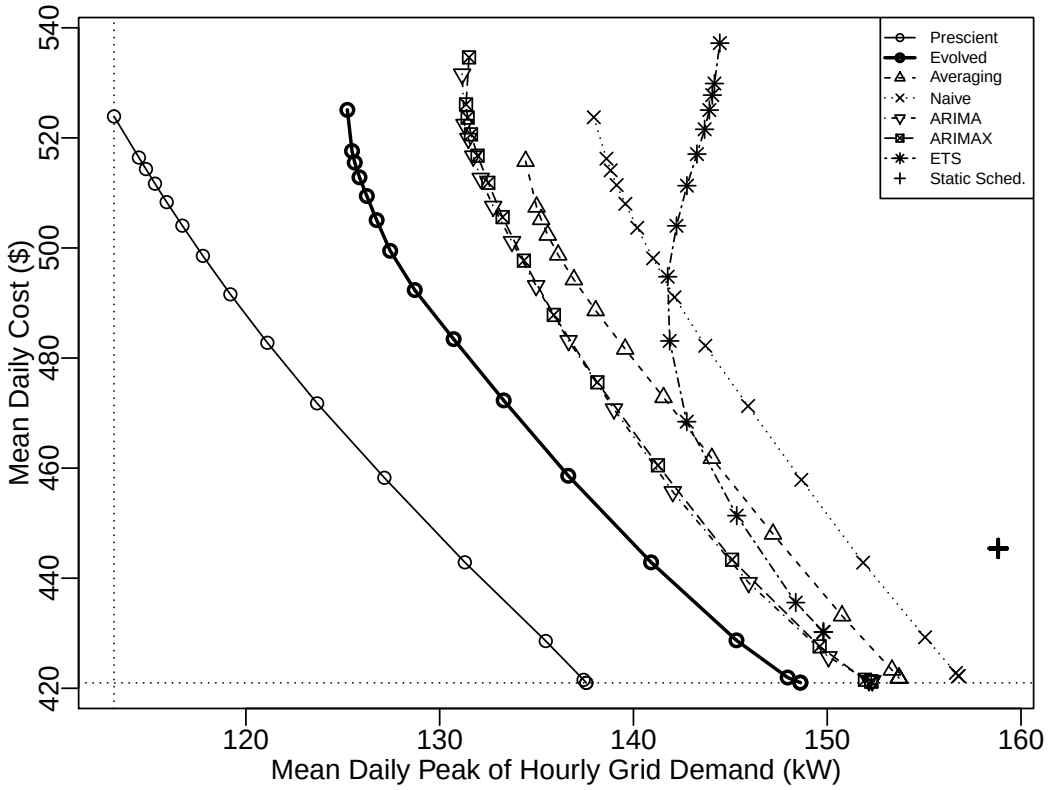


FIGURE 8.10: Mean daily peak demand versus cost, reported with hourly temporal resolution.

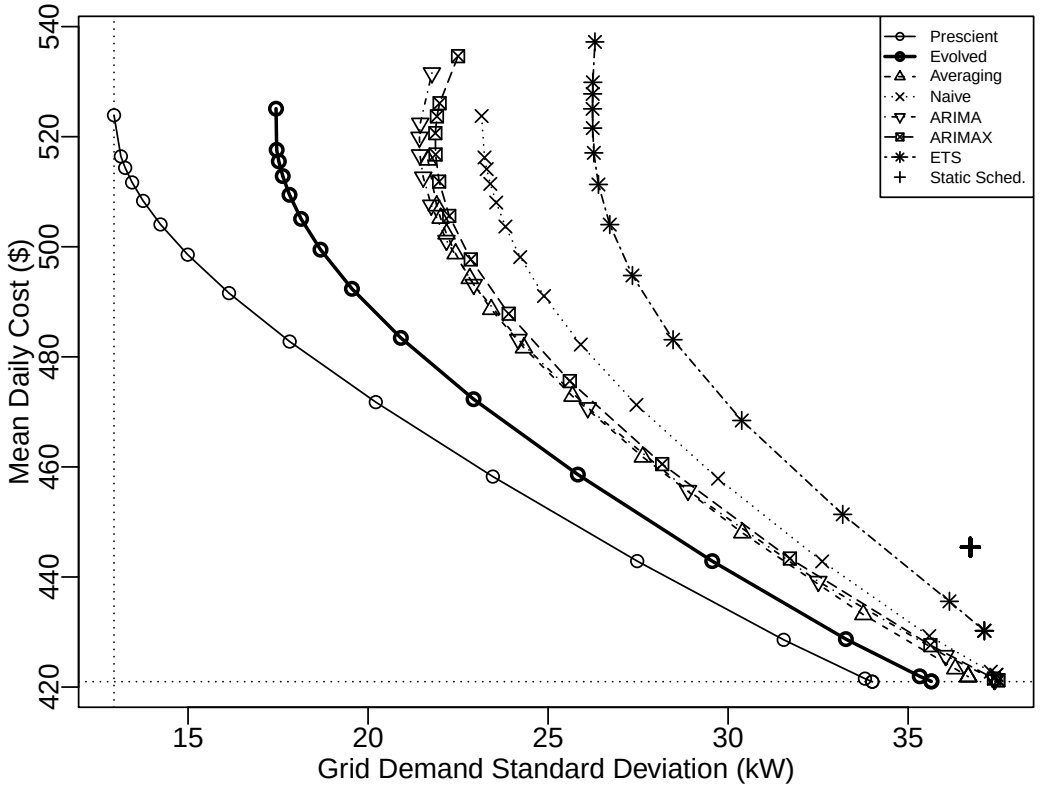


FIGURE 8.11: Standard deviation of grid demand versus mean daily cost using time-series forecasting and predictive control.

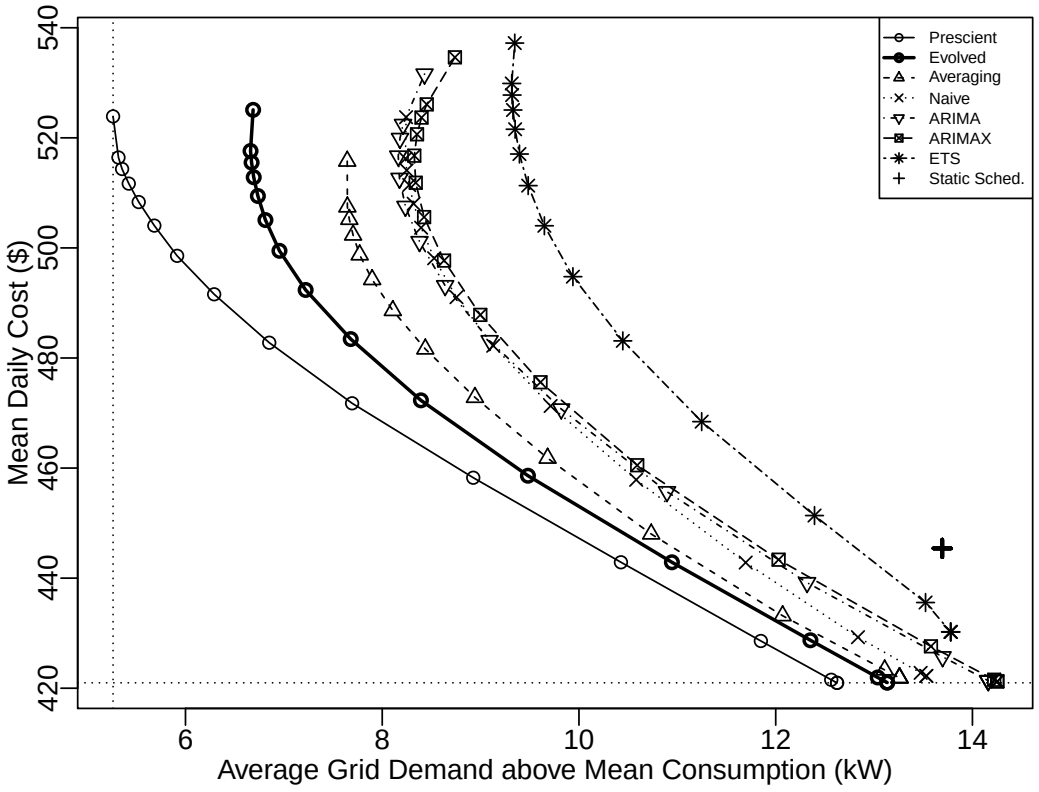


FIGURE 8.12: Average grid demand above mean consumption versus daily cost using time-series forecasting and predictive control.

In addition to the above tests, similar to the concept of value at risk in finance, the first upper partial momentum (UPM_1) of demand, defined as

$$\frac{1}{N}\sum_{i=1}^{N}\max(g_i - \bar{g}, 0)$$

where

$$\bar{g} = \frac{1}{N}\sum_{i=1}^{N} g_i,$$

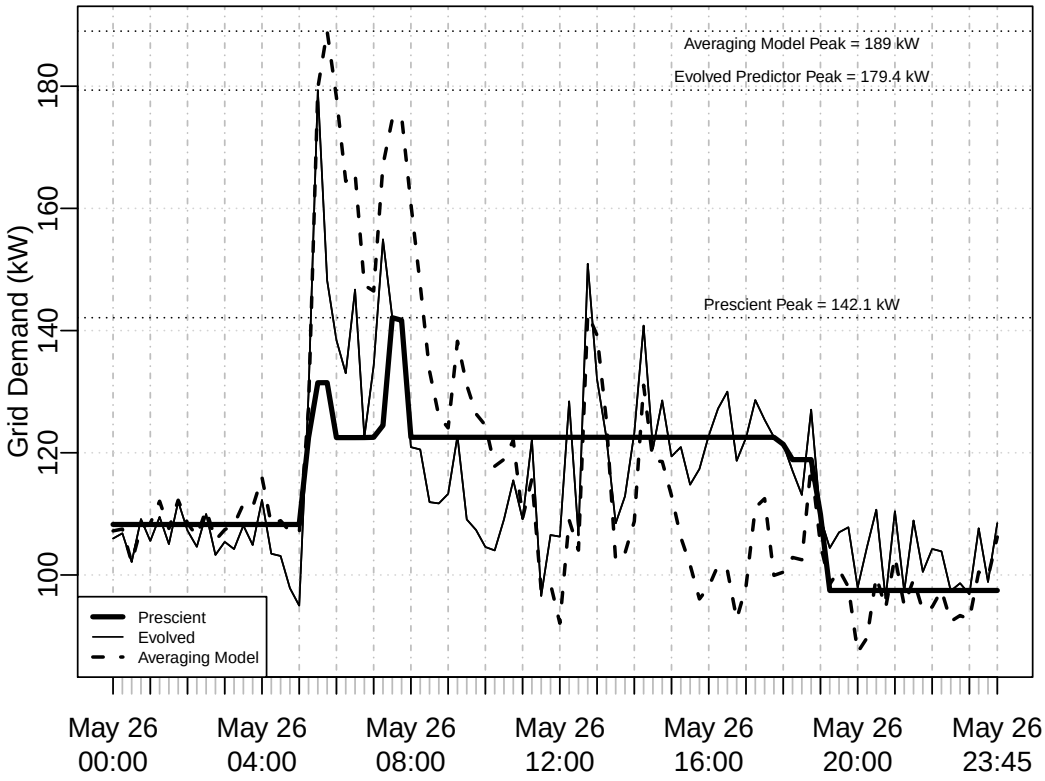
was employed to measure the grid demand above mean consumption. The results are presented in Figure 8.12 and similar observations are made.

This difference, between the daily peak computed at 15-minute intervals and other measures, is caused by how the evolved predictor and the controller respond to sudden changes in data. An example is presented in Figure 8.13, showing an early demand at 5:00 a.m. which was not predictable. Figure 8.13a shows that while the evolved predictor has managed to reduce the peak for the rest of 5 a.m. as well as the rest of the day, the *daily* peak has been set by this unexpected jump in demand. In contrast, Figure 8.13b

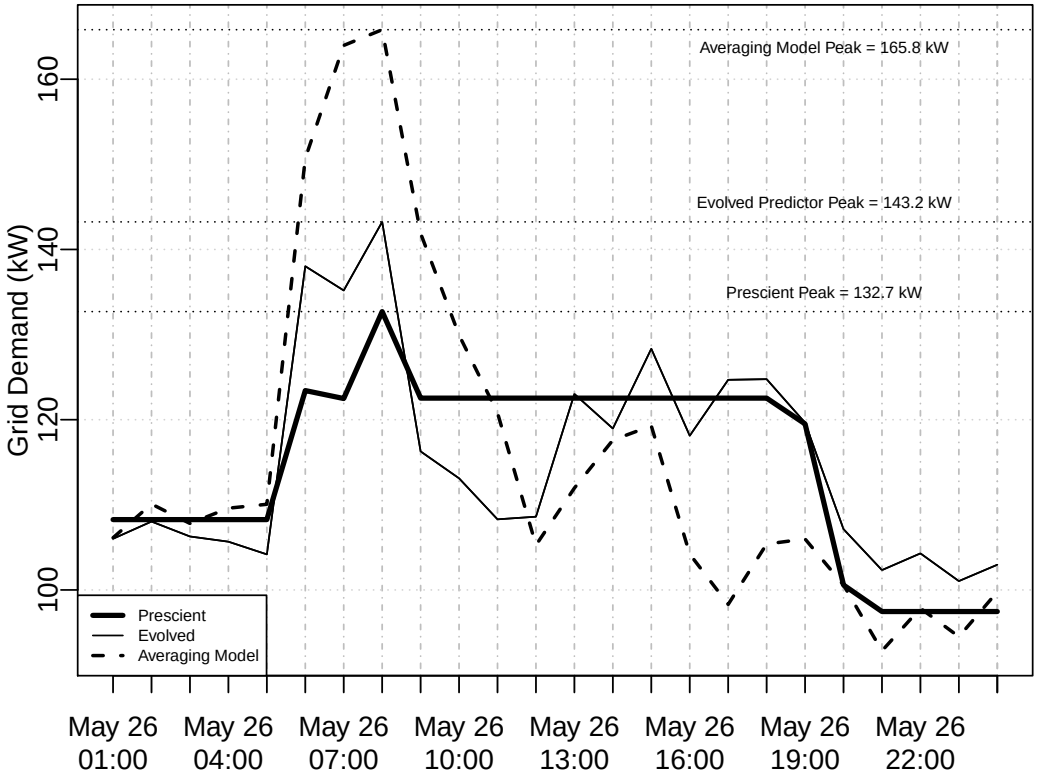
which has averaged the demand into per-hour values, reflects this peak reduction by averaging the 5:00 peak with the three other demand measurements of that hour (namely, 5:15 to 5:45). As a result, the hourly peak demand is significantly lower.

Additionally, in Figure 8.13, the averaging model shows more variance and above mean consumption compared to the grammar-based predictor. Another example, presented in Figure 8.14 demonstrates a similar behaviour for another day.

A comparison against the static scheduling is presented in Figure 8.15. The dynamic results were generated by setting $\lambda = 1000$ in (8.5). The figure shows that while the controller has managed to reduce the peak in the 13:00 to 18:00 period similar to the static schedule, peaks were also shaved during the morning period from 262 kW to 213 kW.



(A) 15-minute intervals.



(B) Hourly intervals.

FIGURE 8.13: Effect of time resolution on peak shaving results. While the 15-minute plot shows only a 10 kW peak shaving improvement for the evolved model, the hourly plot's peak is reduced by 22 kW compared to the averaging model.

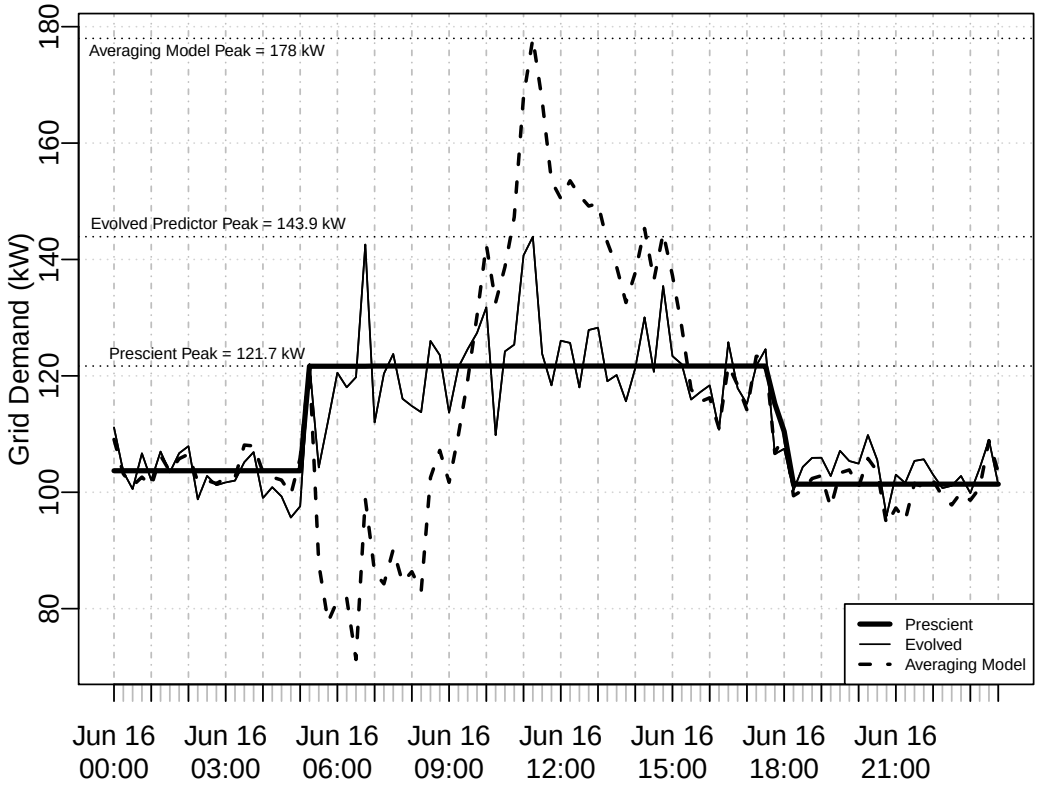


FIGURE 8.14: An example of peak shaving, comparing evolved grammar based predictor with the averaging model, with $\lambda = 0$.

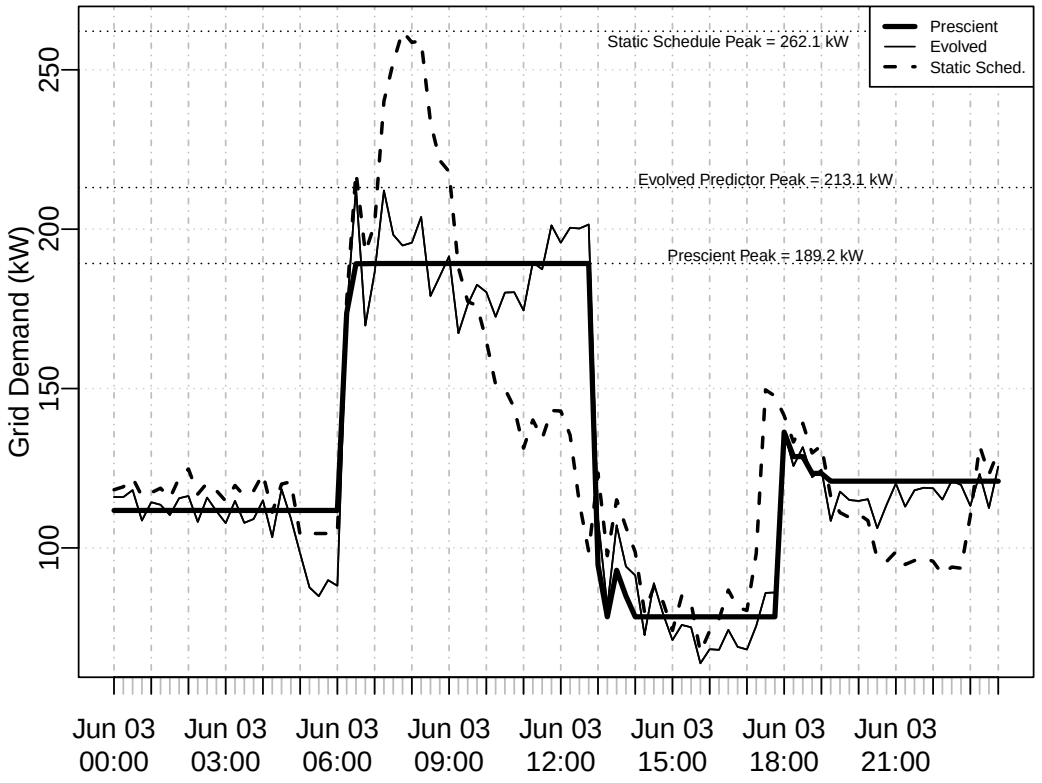


FIGURE 8.15: An example of peak shaving, with emphasis on cost saving during 13:00 to 18:00 with $\lambda = 1000$, comparing evolved grammar based predictor with the static schedule.

8.6 Summary

In this chapter, a receding horizon SMPC system, combined with automatic time-series forecasting, was used to shave peak grid demand and save electricity costs. This technique used grammatical evolution to evolve forecasters for electricity generation and consumption, and used a quadratic model to determine the optimal course of action based on the uncertainties and problem requirements. The results show that the evolved predictors outperform other commonly used forecasting algorithms. In addition, the configurability of the grammar used to create forecasters, allows experts to update the model to improve the results based on model specific knowledge or available data.

A summary of the results, comparing peak shaving outcomes using different statistics for a prescient controller, SMPC with three different forecasters, and static scheduling, is presented in Table 8.6. The table reveals that the proposed evolved grammar based forecaster offers 43.8 kW average daily peak shaving over static scheduling, a 9% improvement over the second best practical technique.

While run-time of the proposed SMPC algorithm is acceptable for implementation on microcontroller systems, the cost of model selection using GE prohibits its full deployment. This problem can be solved by offloading the computational overhead of GE optimisations to a cloud-based high performance computer (HPC) system. This methodology is further explored in Appendix D.

TABLE 8.6: Summary of demand management results using a prescient controller, SMPC with three different predictors, and static scheduling. Values in parenthesis show the percentage of improvement.

	Prescient	Evolved	Averaging	ARIMA	Static Sched.
Mean daily peak	121.6 kW (100%)	**153.7 kW (57.6%)**	160.6 kW (48.6%)	165.9 kW (41.7%)	197.5 kW (0%)
Mean daily peak (hourly)	113.2 kW (100%)	**125.2 kW (73.6%)**	134.4 kW (53.4%)	131.2 kW (60.6%)	158.8 kW (0%)
Standard deviation	12.9 kW (100%)	**17.4 kW (81.1%)**	21.7 kW (63.3%)	21.8 kW (62.9%)	36.7 kW (0%)
UPM_1	5.3 kW (100%)	**6.7 kW (83.1%)**	7.6 kW (71.8%)	8.4 kW (62.4%)	13.7 kW (0%)

Chapter 9

Conclusion

> When forecasting currency exchange rates, ⋯ there is plenty of available data. However, we have a very limited understanding of the factors that affect exchange rates. ⋯, you will be correct about 50% of the time, whatever you forecast. In situations like this, forecasters need to be aware of their own limitations, and not claim more than is possible.
>
> Chapter 1 - What can be forecast?
> *Forecasting: Principles & Practice* [36]

Success of any risk management system depends on two major factors:

1. Forecasting accuracy, or how close to reality is one's estimate of the future.
2. Ability of the risk management process to prepare for the uncertain future and respond to unanticipated events.

This thesis advocated managing risk using stochastic model predictive control (SMPC). To improve the predictive model's accuracy, a non-linear time-series forecasting model enhanced with domain-specific knowledge and tuned using evolutionary optimisation was proposed.

This methodology was applied on two real-world applications: the problem of foreign exchange (FX) risk management from an FX broker points of view, and managing electricity demand from the grid using a battery energy storage system (BESS).

SMPC showed promising results for controlling the outcomes, namely improving the FX broker's risk and transaction cost profile, and the BESS peak shaving capabilities. Extensive tests revealed that the risk management performance was highly correlated with the accuracy of future forecasts.

The proposed time-series forecasting technique proved successful in predicting future electricity demand and generation in the BESS application. In comparison, none of the studied techniques showed results better than a Gaussian distribution for predicting the FX broker's future trades. As it has been noted by other researchers, limitations of the available information and model structures must be considered when evaluating the results of time-series prediction.

9.1 Summary of Achievements

The specific aims and contributions of this thesis were listed in Section 1.2.

The theoretical contributions were presented in the chapters 3, 5, and 6:

- In Chapter 3, the dealer's dynamics were modelled in a state-space, based on realistic assumptions accommodating stochastic models for FX rates, inter-bank transaction costs, and client flow. To find the optimal portfolio of foreign currency positions, the problem was formulated as a multi-period mean-variance stochastic optimisation, which was then reduced to a quadratic program, allowing efficient solution of such system using numerical techniques.
- Chapter 5 proposed a customisable and scalable technique for predicting time-series. Context-free grammar was used to describe a configurable template forecaster structure, and the necessary requirements for optimising this structure to fit any data using grammatical evolution were discussed.
- Chapter 6 studied the suitability of common error measures (e.g., RMSE) for comparing forecast results in the context of linear-quadratic systems optimal control. It was shown that these error measures may differ from the results obtained from the controller's cost function, and consequently a closed-form solution was derived to accurately measure the cost deviation due to forecast errors. In addition to benchmark problems, application of this technique in evaluating forecast results for the FX risk management system was also discussed.

The FX risk management system was backtested extensively with real-world and synthetic data in Chapter 7. Time-varying models were proposed based on realistic market assumptions, and used to model client flow, FX rate volatility, and transaction costs. To test the capability of the risk management model, a scenario generation oracle was defined to generate scenarios with different forecasting accuracies. SMPC hedging was shown to significantly outperform benchmark strategies, and its performance with different levels of scenario generation quality was quantified. Additionally, the proposed evolutionary time-series forecasting methodology was tested on client flow. It was shown

that the client flow is best modelled using a univariate Gaussian distribution, which combined with the SMPC hedging improves the hedging performance up to 44.7%.

In Chapter 8, the proposed methodology was applied to manage a grid connected battery storage system, under the uncertainties of load prediction and renewable energy generation. A multi-period optimisation formulation was proposed to dynamically manage the charge/discharge schedule of batteries based on demand forecasts, with the configurable objectives of peak shaving and cost saving. A time-series forecasting grammar was designed based on the available information to individually model and predict components of the total demand, including the consumption by different load sources and the renewable generation from separate PV solar panels. The forecasting and scheduling methodology were then tested on real-world data from the TransGrid iDemand, and it was shown that the proposed techniques can reduce the daily peak from an average of 197.4 kW during the test period, to 153.7 kW. This is equal to a 57.6% improvement, compared to a static scheduler and a prescient controller.

9.2 Future Work

The proposed state-space model for FX risk management uses fixed length time-steps in its multi-period model. As the size of horizon increases, one may wish to use variable time-step sizes in the model, with finer steps for near future actions and coarser steps for far future. This is possible by adding an amortisation term to the transaction cost and volatility models according to the length of the time-steps. This method reduces the number of variables, and therefore the computational complexity, both in forecasting and optimisation, at the expense of accuracy. Quantisation of this trade-off is another direction for future studies.

Similarly, the proposed time-series prediction system only supports one resolution over its forecast horizon. Future work can propose models with multiple time resolutions, and then apply boosting techniques or hierarchical time-series analysis to improve the overall forecast results.

Some of the assumptions in the FX model can also be improved. For example, transaction cost models can use absolute value or logarithmic terms in addition to the quadratic cost, and instead of using variance as a measure of risk, one-sided moments can be utilised. These measures, while still convex, will not result in a quadratic programming problem. Considering that the FX market has different properties from other financial markets, one has to first justify the improvements in modelling before attempting to implement these refinements.

Additional computational intelligence techniques can replace the genetic algorithm in the evolutionary optimisation phase, including simulated annealing or particle swarms. The prediction grammar can also be redesigned to be non-recursive, or multiple chromosomes can be used for different parts of the forecasting pipeline. In order to obtain better forecasts from the available chromosomes in each generation, additional work may consider an automatic technique to boost the GE results.

The error measure introduced in Chapter 6 is limited to linear-quadratic systems without constraints. Further studies are required to include constraints in the formulation, or determine error bounds in the presence of constraints. Future work may include implementation of this error form directly in a learning algorithm, or a comprehensive study of improvements in real-world examples.

While the results of the predictors evolved by GE outperformed other techniques, one would expect to achieve better results with a more extensive search. In this thesis, the number of generations was limited to 50. Considering that time to finish the GE optimisation is directly proportional to the number of generations divided by the number of available processors, one can run the algorithm on a HPC cluster or cloud to improve search results or shorten the runtime. It must be noted that the current *gramEvol* package has basic support for multi-processing, but extending the package to consider unreliable connections in a distributed system, such as implementing an island hopping evolutionary algorithm, requires further work.

The electricity demand forecast results were obtained from *ex-post* weather observations; however, in reality, only ex-ante weather forecasts are available, and further tests are required based on prospective data. Additionally, as noted in Chapter 8, one can replace the publicly available forecasts with commercial insolation data and satellite cloud maps to improve results. The cost effectiveness of using commercial data can be the subject of a comparative study.

In addition to load and solar generation, electricity prices can also be forecast using the grammar. As a variable price will add further uncertainty to the system, one may have to include variance of the price as a risk factor in the BESS scheduler's optimisation model.

Appendix A

Package gramEvol: Grammatical Evolution in R

A.1 Introduction

Grammatical evolution (GE) uses context-free grammars (CFG) to provide a concise and versatile mechanism for expressing structure of programs. Combined with evolutionary optimisation, GE creates a powerful framework that allows integration of domain-specific knowledge, defined using a grammar, into real-world optimisation problems.

In this appendix, the R package *gramEvol* is presented, which facilitates defining, creating, evaluating, and evolving programs using GE in R [202] .

A.1.1 Rationale for an R implementation

Several open source software implementations of GE exist, including

- Java: *GEVA* [249],
- Python: *PonyGE* [250] and *PyNeurGen* [251],
- MATLAB: *GEM* [252], and
- Ruby: *GERET* [253].

gramEvol is the first package for R.

The design goal of this package is to evolve programs natively in R. The Comprehensive R Archive Network (CRAN) [254] includes many open source libraries for machine learning and data mining, making R an attractive target for machine learning program implementations. While it is possible to generate and call R code from other languages, a native implementation has the following advantages:

- R's `expression` objects can be used to define a grammar, removing an error prone text-based BNF interpretation or compilation step, allowing dynamic grammar manipulation and rapid prototyping.
- Expression are created directly in R as `expression` objects, which removes the overhead of calling R from an external program.
- Only R's base packages are used for evolutionary operations and grammar processing along with parsing and running generated programs. This eliminates the need for third-party libraries and external dependencies.

As a result, *gramEvol* allows quick and easy integration of domain-specific knowledge into machine learning problems described and programmed using R.

A disadvantage of *gramEvol* is its speed compared to compiled GE libraries, such as *libGE* [255] or *AGE* [256], which are written in C++. We assume that the computational overhead of processing the cost function is greater than the overhead of GE operators. Hence, any major speed-up will be a result of moving the cost function computational bottleneck to C, C++ or Fortran. This is already a common practice in the design and implementation of R packages. Furthermore, packages such as *Rcpp* [257] are available to facilitate porting existing R code to C++.

A.2 Using *gramEvol*

The CFG and GE's theoretical backgrounds were introduced in Section 2.6. In this section, the syntax required for defining a GE problem using *gramEvol* is described, and the internal architecture of this package is explained.

A.2.1 Defining a Grammar

In *gramEvol*, a grammar is defined by passing a list of productions rules to the function `CreateGrammar`. `CreateGrammar` automatically determines the terminal, non-terminal and start symbols based on the rules. *gramEvol* supports two type of rules: expression based rules defined using `grule`, and character string rules defined using `gsrule`.

For example, the following commands will construct the CFG of Table 2.1 using `gsrule`:

```
library("gramEvol")

ruleDef <- list(expr = gsrule("(<expr>)<op>(<expr>)", "<coef>*<var>"),
```

```
                op   = gsrule("+", "-", "*", "/"),
                coef = gsrule("c1", "c2"),
                var  = gsrule("v1", "v2"))

grammarDef <- CreateGrammar(ruleDef)

grammarDef

## <expr> ::= (<expr>)<op>(<expr>) | <coef>*<var>
## <op>   ::= + | - | * | /
## <coef> ::= c1 | c2
## <var>  ::= v1 | v2
```

Using R's native `expression` objects require a change to the grammar, as `expr op expr` is not valid in R. Instead, a functional form of `op(expr, expr)` is used with `grule`:

```
ruleDef <- list(expr = grule(op(expr, expr), coef*var),
                op   = grule(`+`, `-`, `*`, `/`),
                coef = grule(c1, c2),
                var  = grule(v1, v2))

CreateGrammar(ruleDef)

## <expr> ::= <op>(<expr>, <expr>) | <coef> * <var>
## <op>   ::= `+` | `-` | `*` | `/`
## <coef> ::= c1 | c2
## <var>  ::= v1 | v2
```

The grammar properties are reported via the `summary` function:

```
summary(grammarDef)

## Start Symbol:                <expr>
## Is Recursive:                TRUE
## Tree Depth:                  Limited to 4
## Maximum Rule Choices:        4
## Maximum Sequence Length:     18
## Maximum Sequence Variation:  2 2 2 2 4 4 2 2 2 4 2 2 2 2 4 2 2 2
## No. of Unique Expressions:   18500
```

This summary reports that:

- The non-terminal symbol of the first production rule (i.e., <*expr*>) was selected as the start symbol $\mathcal{S}$.
- The grammar is cyclic, i.e., the non-terminal symbol <*expr*> expands to more <*expr*>s. To avoid infinite recursion, the maximum recursion depth is limited to the number of production rules.
- The grammar tree depth is limited to four.
- Maximum choices in a production rule is four, given by <*op*>.
- Maximum length of a chromosome, avoiding wrapping and limiting recursions, is 18.
- The maximum variation of each integer codon in the chromosome. This value depends on the location of the codons and the grammar, and helps reduce the search space of chromosomes.
- The grammar, with recursion limited to four, can create 18500 different expressions.

`GrammarMap` maps a sequence of integers (the genotype in evolutionary algorithms) to a symbolic expression (the phenotype). The code below converts the numeric genome used in Table 2.2's example to its analytical phenotype, using the `verbose` argument to show the steps of the mapping.

```
genome <- c(2, 1, 0, 0, 3, 3, 3, 1)
expr <- GrammarMap(genome, grammarDef, verbose=TRUE)


##  Step Codon Symbol Rule                 Result
##  0                 starting:            <expr>
##  1    2     <expr> (<expr>)<op>(<expr>) (<expr>)<op>(<expr>)
##  2    1     <expr> <coef>*<var>         (<coef>*<var>)<op>(<expr>)
##  3    0     <coef> c1                   (c1*<var>)<op>(<expr>)
##  4    0     <var>  v1                   (c1*v1)<op>(<expr>)
##  5    3     <op>   /                    (c1*v1)/(<expr>)
##  6    3     <expr> <coef>*<var>         (c1*v1)/(<coef>*<var>)
##  7    3     <coef> c2                   (c1*v1)/(c2*<var>)
##  8    1     <var>  v2                   (c1*v1)/(c2*v2)
## Valid Expression Found
```

```
expr


## (c1 * v1)/(c2 * v2)
```

The returned object is of class `GEPhenotype`. It can be cast to a character string or an expression and subsequently evaluated using R's `eval` function:

```
as.character(expr)


## [1] "(c1 * v1)/(c2 * v2)"


c1 <- 1
c2 <- 2
v1 <- 3
v2 <- 4
eval(as.expression(expr))


## [1] 0.375
```

To inspect some random expressions of the grammar, `GrammarRandomExpression` can be used. For the purpose of reproducibility, the random generator seed value is first set to a fixed value:

```
set.seed(0)
GrammarRandomExpression(grammarDef, numExpr = 4)


## [[1]]
## expression((c2 * v2) + (c1 * v1))
##
## [[2]]
## expression((c1 * v1) - (c1 * v2))
##
## [[3]]
## expression(c1 * v1)
##
## [[4]]
## expression((((c1 * v2) - ((c1 * v2) - (c2 * v2))) + ((c1 * v1) +
##     (c1 * v2))) - ((c1 * v2) - (c2 * v2)))
```

From the example, it can be seen that this grammar is capable of generating both simple and complex expressions.

A.2.2 Exhaustive and Random Search in Grammar

Context-free grammars are a general way of describing program structures, not bound to evolutionary optimisation. As a result, *gramEvol* additionally supports exhaustive and random search.

The first step in any optimisation is defining a cost function. This function receives an expression generated using the grammar, and returns an appropriate score. For example, in order to find the numeric sequence that generates a certain expression, the following cost function returns the generalised Levenshtein distance of the current expression and the target:

```
evalFunc <- function(expr) {
  adist(as.character(expr), "(c1 * v1) - (c2 * v2)")
}
```

The objective is to find a suitable chromosome, and therefore the expression, that minimises the cost function, i.e., the string distance. `GrammaticalExhaustiveSearch` performs an exhaustive search to find this expression:

```
GrammaticalExhaustiveSearch(grammarDef, evalFunc)


## GE Search Results:
##   Expressions Tested: 18500
##   Best Chromosome:    0 1 0 0 1 1 1 1
##   Best Expression:    (c1 * v1) - (c2 * v2)
##   Best Cost:          0
```

`GrammaticalRandomSearch` performs a similar albeit random search. The `terminationCost` option allows the algorithm to terminate if the required minimum cost is found. In our example, the optimal cost is zero:

```
GrammaticalRandomSearch(grammarDef, evalFunc, terminationCost = 0)


## GE Search Results:
##   Expressions Tested: 1000
```

```
##   Best Chromosome:    0 1 2 0 2 3 1 1 0 3 1 1 0 0 1 1 1 2
##   Best Expression:    (c1 * v1) * (c2 * v2)
##   Best Cost:          1
```

Both of these methods have their drawbacks: testing 18500 expressions requires extensive computation, and a random search is ineffective. In such cases, considering the non-smoothness and non-convexity of the search space, evolutionary algorithms are often an efficient choice.

A.2.3 Evolving a Grammar

`GrammaticalEvolution` uses evolutionary optimisation to find the minima of `evalFunc`. Continuing the previous example, the best expression is determined using the same grammar and cost function, optimised using `GrammaticalEvolution`. Details of evolutionary optimisation, such as size of the population and number of iterations are automatically chosen by an internal heuristic:

```
result <- GrammaticalEvolution(grammarDef, evalFunc, terminationCost = 0)
print(result, show.genome = TRUE)


## Grammatical Evolution Search Results:
##   No. Generations:  3
##   Best Genome:      2 1 0 0 1 1 1 1 0 3 3 1 2 1 2 0 2 1
##   Best Expression:  (c1 * v1) - (c2 * v2)
##   Best Cost:        0
```

It is evident that the evolutionary algorithm has quickly converged to the optimisation objective.

`GrammaticalEvolution` allows monitoring the status of each generation using a callback function. This function, if provided to parameter `monitorFunc`, receives an object similar to the return value of `GrammaticalEvolution`. For example, the following function prints the information about the current generation and the best chromosome in the current generation:

```
customMonitorFunc <- function(results){
  print(results)
}
```

```
ge <- GrammaticalEvolution(grammarDef, evalFunc, terminationCost = 0,
  monitorFunc = customMonitorFunc)
```

`GrammaticalEvolution` is summarised as pseudocode in Algorithm A.1. Internally, `GrammaticalEvolution` uses `GeneticAlg.int`, which is a GA implementation with integer codons partially based on *genalg* package by Willighagen [258]:

- Using the information obtained about the grammar (e.g., number of possibles expressions and maximum sequence length), `GrammaticalEvolution` applies a heuristic algorithm based on the work of Deb and Agrawal [259] to automatically determine a suitable value for the `popSize` (i.e., the population size) and the `iterations` (i.e., the number of iterations) parameters.
- The ordinary crossover operator is considered destructive when homologous production rules are not aligned, such as for cyclic grammars [260]. Consequently, `GrammaticalEvolution` automatically changes crossover parameters depending on the grammar to improve optimisation results.
- Each integer chromosome is mapped using the grammar, and its fitness is assessed by calling `evalFunc` (i.e., the cost function).
- After reaching a termination criteria, e.g., the maximum number of `iterations` or the desired `terminationCost`, the algorithm stops and returns the best expression found so far.
- `GrammaticalEvolution` also supports multi-gene operations, generating more than one expression per chromosome using the `numExpr` parameter.

A.2.4 Parallel Processing Option

Processing expressions and computing their fitness is often computationally expensive. The *gramEvol* package can utilise parallel processing facilities in R to improve its performance. This is done through the `plapply` argument of `GrammaticalEvolution` function. By default, `lapply` function is used to evaluate all chromosomes in the population.

Multi-core systems simply benefit from using `mclapply` from package *parallel* [202], which is a drop-in replacement for `lapply` on POSIX compatible systems. The following code optimises `evalFunc` on 4 cores:

```
library("parallel")
options(mc.cores = 4)
ge <- GrammaticalEvolution(grammarDef, evalFunc,
  plapply = mclapply)
```

Algorithm A.1: GE implementation in *gramEvol*.

```
1  Function GrammaticalEvolution is
2      if missing crossover parameters then
3          determine crossover parameters based on grammar
4      end
5      if missing popSize or iterations then
6          determine the optimal popSize and iterations based on grammar
7      end
8      genotypes ← suggestions
9      for i := length(suggestions) + 1 to popSize do
10         genotypes ←(Append)— random chromosome
11     end
12     for generation := 1 to iterations do
13         for i := 1 to len(genotypes) do
14             phenotypes[i] ← GrammarMap (grammarDef, genotypes[i], wrappings)
15         end
16         fitnesses ← evalFunction(phenotypes)
17         if terminationCost is given & minimum(fitnesses) <
           terminationCost then
18             break For loop
19         end
20         genotypes ← sort genotypes by their fitness
21         new_genotypes ← genotypes[1 to elitism]
22         for i := elitism+1 to popSize do
23             parent_1 ← Select from genotypes using Roulette Wheel operator
24             parent_2 ← Select from genotypes using Roulette Wheel operator
25             new_genotypes[i] ← Crossover(parent_1, parent_2, crossover
               parameters)
26             if random number > mutationChance then
27                 Mutate new_genotypes[i]
28             end
29         end
30         genotypes ← new_genotypes
31     end
32     return the genotype and phenotype (i.e., the expression) with the best
       fitness
33 end
```

To run *gramEvol* on a cluster, `clusterapply` functions can be used instead. The *gramEvol* package must be first installed on all machines and the evaluation function and its data dependencies exported to all cluster nodes before GE is called. The following example demonstrates a four-process cluster running on the local machine:

```
library("parallel")
cl <- makeCluster(type = "PSOCK", c("127.0.0.1",
                                    "127.0.0.1",
```

```
                                    "127.0.0.1",
                                    "127.0.0.1"))
clusterEvalQ(cl, library("gramEvol"))
clusterExport(cl, c("evalFunc"))
ge <- GrammaticalEvolution(grammarDef, evalFunc,
                           plapply = function(...) parLapply(cl, ...))
stopCluster(cl)
```

A.2.5 Non-terminal Expressions

As demonstrated in Section A.2.1, a cyclic grammar allows complex expressions to be derived from a compact description. However, if the chromosome is too short, the expression may still contain non-terminal symbols even after wrapping multiple times. For example:

```
chromosome <- c(0)
expr <- GrammarMap(chromosome, grammarDef)
expr


## Non-Terminal Sequence:
##  (((<expr>)<op>(<expr>))<op>(<expr>))<op>(<expr>)
```

Non-terminal expressions are identified using `GrammarIsTerminal` function:

```
GrammarIsTerminal(expr)


## [1] FALSE
```

`GrammaticalEvolution` and other search functions automatically filter non-terminal expressions, and the user does not need to worry about them in practice.

A.3 Grammatical Evolution for Machine Learning

In this section, three applications of grammatical evolution in statistics and machine learning are explored. Other applications, such as symbolic regression and regular expression discovery using package *rex* [261] are explained in the package's vignette.

A.3.1 Model Selection and Hyper-parameter Optimisation

In machine learning, selecting the best learning model is often performed in three steps:

1. Feature selection, where different features are selected as inputs to for a learning model.
2. Model selection, where candidate learning models are compared and one of them is selected.
3. Hyper-parameter optimisation, where hyper-parameters of the model are optimised for the current objective, (e.g., the kernel type and parameters for kernel methods).

Due to their importance, dedicated packages such as *caret* [204] support feature selection and hyper-parameter optimisation for many machine learning techniques. Extending these packages to support new algorithms or combining additional steps into their operation, however, require structural changes to the package's code. In this section, we show how CFGs can offer an easily extensible framework for a simultaneous feature selection, model selection and hyper-parameter optimisation.

Here, the `ChickWeight` dataset [202] is used to demonstrate these steps. The objective is to learn the `weight` of a chicken based on the `Time` passed since its birth and its `Diet`. The `Chick` identifier is also included.

We choose a linear model, an artificial neural network (ANN), and support vector regression (SVR) from *e1071* [262] as the possible learning algorithms.

```
data("ChickWeight")
library("e1071")
library("nnet")

grammarDef <- CreateGrammar(list(
  learner = grule(function(train.data) {
    result <- NULL
    features <-  weight ~ F1 + F2 + F3
    if (length(attr(terms(features), "variables")) > 2) {
          capture.output({
        result <- model
      })
        }
    return(result)
  }),
  model = grule(lm(features, train.data),
```

```
                nnet(features, train.data, size = nn.size),
                svm(features, train.data, cost = svm.c, svm.hyperparam)),
  F1       = grule(Time, 0),
  F2       = grule(Chick, 0),
  F3       = grule(Diet, 0),
  nn.size  = grule(4, 8, 16),
  svm.hyperparam = grule(.(kernel = "linear"),
                         .(kernel = "polynomial", degree = svm.degree),
                         .(kernel = "radial", gamma = svm.gamma)),
  svm.c       = grule(0.1, 1, 10, 100, 1000),
  svm.degree  = grule(1, 2, 3, 4, 5),
  svm.gamma   = grule(0.1, 0.2, 0.5, 1.0)))
```

The start symbol, the <*learner*>, has only one production rule, which creates a function that receives the training data and returns the trained model:

- It first selects the appropriate formula of `features`, and if there is at least one regressor variable, it returns a <*model*>. The `features` formula is built by either selecting a variable (i.e., `Time`, `Chick`, and `Diet`), or 0 using <*F1*>, <*F2*>, and <*F3*> rules.
- The <*model*> can be either a `lm`, an `svm` or a `nnet` and is wrapped in `capture.output` to suppress the diagnostic but useless messages by `nnet`.
- Each learning algorithm has its own set of hyper-parameters: `nnet`'s hidden layer `size` is determined using <*nn.size*>, and `svm` uses <*sym.hyperparamm*> to select its kernel and its associated parameter in one-step. Here, `.()` is used to avoid premature interpretation of assignment operator and comma (i.e., `=` and `,`) by R.

The remaining rules, assign certain ranges of values to different hyper-parameters, similar to an ordinary *grid search*.

An example of an expression generated by this grammar is:

```
GrammarRandomExpression(grammarDef)


## expression(function(train.data) {
##     result <- NULL
##     features <- weight ~ 0 + 0 + Diet
##     if (length(attr(terms(features), "variables")) > 2) {
##         capture.output({
##             result <- nnet(features, train.data, size = 4)
```

```
##         })
##     }
##     return(result)
## })
```

This uses `Diet` as a feature, and an ANN with four neurons in its hidden layer as its model.

The grammar can generate 432 unique models:

```
summary(grammarDef)

## Start Symbol:                <learner>
## Is Recursive:                FALSE
## Tree Depth:                  4
## Maximum Rule Choices:        5
## Maximum Sequence Length:     8
## Maximum Sequence Variation:  1 2 2 2 3 5 3 5
## No. of Unique Expressions:   432
```

To assess each model, a cost function is required. In this example, we define a simple cross-validation test, returning the out-of-sample mean square error (MSE):

```
set.seed(0)

data("ChickWeight")
total.samples <- nrow(ChickWeight)
train.ind <- sample(total.samples, trunc(total.samples * .8))
train.data <- ChickWeight[train.ind,]
test.data <- ChickWeight[-train.ind,]

eval.chicken <- function(expr) {
  trainer <- eval(expr)
  model <- trainer(train.data)

  if (is.null(model)) {
    return (Inf)
  }
```

```
  test.results <- predict(model, test.data)
  cost <- mean((test.results - test.data$weight)^2)
  return (cost)
}
```

The `eval.chicken` function, first evaluates the expression to get its underlying function. This function is then applied to the training data to obtain a model. If the model is `NULL`, i.e., some error has occurred during the training, it returns a very high cost. Otherwise, the model is used on the testing data, and the MSE of the results is returned.

To find the best combination of features, model and hyper-parameters, Grammatical Evolution is applied to the appropriate grammar and cost function:

```
result <- GrammaticalEvolution(grammarDef, eval.chicken)
result

## Grammatical Evolution Search Results:
##   No. Generations:  108
##   Best Expression:  function(train.data) {
##     result <- NULL
##     features <- weight ~ Time + Chick + 0
##     if (length(attr(terms(features), "variables")) > 2) {
##         capture.output({
##             result <- svm(features, train.data, cost = 100,
##                           kernel = "radial", gamma = 0.1)
##         })
##     }
##     return(result)
## }
##   Best Cost:        68.3212558249473
```

The optimal model uses only two of the available features with a radial kernel SVR, and is identical to the result of an exhaustive search:

```
GrammaticalExhaustiveSearch(grammarDef, eval.chicken)

## GE Search Results:
##   Expressions Tested: 432
##   Best Chromosome:    0 0 0 1 2 3 2 0
```

	Exhaustive search	GE minimum	GE median	GE maximum
Error	68.32	68.32	68.32	988.45
Generations	-	1	39.50	108
Time (s)	71.72	0.61	19.60	149.57

TABLE A.1: Summary of GE's performance for 100 runs of the model selection example.

```
##   Best Expression:    function(train.data) {
##     result <- NULL
##     features <- weight ~ Time + Chick + 0
##     if (length(attr(terms(features), "variables")) > 2) {
##         capture.output({
##             result <- svm(features, train.data, cost = 100,
##                           kernel = "radial", gamma = 0.1)
##         })
##     }
##     return(result)
## }
##   Best Cost:          68.32126
```

To compare the performance of GE and exhaustive search, the GE was run 100 times, with termination condition set to reaching the global optima obtained by the exhaustive search. The error, number of generations and the duration of execution was measured. The tests were performed on a single thread on a 3.40 GHz Intel Core i7-2600 CPU. To ensure reproducibility, `set.seed(0)` was executed before running the code. The results are presented in Table A.1. Overall, the GE's average execution time is 3.6 times better than that of the exhaustive search. It must be noted that however, as the GE is a *stochastic* optimisation, on some occasions it was unable to find the global minima before reaching the maximum number of allowed iterations. In this example this was limited to 108 generations, set automatically by `GrammaticalEvolution`. As a result, the optimisation terminated prior to reaching the global optima.

The final model can be constructed from the results of GE optimisation:

```
train.func <- eval(result$best$expression)
final.model <- train.func(ChickWeight)
```

The machine learning approach used in this section was intentionally kept simple. Other learning algorithms can be added as additional rules, each with their own hyper-parameters. Different options, such as scaling or dimensionality reduction techniques can also be added to the <*learner*> function, each described using separate rules.

A.3.2 Classification

In the second example, we use GE for classification. There are many ways that GE can be adopted for classification, e.g., a model selection on classifiers similar to Section A.3.1. Here, we directly define a grammar which takes input variables and returns the classification result, with a structure similar to a decision tree.

In this example, the objective is defined as separating *Iris versicolor* from other species in the Iris flower dataset. Here, the data is evaluated from a data-frame instead of the program's environment.

```
data("iris")
iris$Species <- ifelse(iris$Species == 'versicolor',
                                        'versicolor', 'other')
ClassifyFitFunc <- function(expr) {
  sum(eval(expr, envir = iris) != iris$Species)
}
```

The grammar is defined using the following code:

```
ruleDef <- list(
  result     = grule(ifelse(expr, 'versicolor', 'other')),
  expr       = grule((expr) & (sub.expr),
                     (expr) | (sub.expr),
                     sub.expr),
  sub.expr   = grule(comparison(var, func.var)),
  comparison = grule(`>`, `<`, `==`, `>=`, `<=`),
  func.var   = grule(num, var, func(var)),
  func       = grule(mean, max, min, sd),
  var        = grule(Sepal.Length,Sepal.Width,Petal.Length,Petal.Width),
  num        = grule(1, 1.5, 2, 2.5, 3, 4, 5))

grammarDef <- CreateGrammar(ruleDef)
```

In this grammar, the start symbol, <*result*>, receives a `TRUE/FALSE` and returns either `'versicolor'` or `'other'`. The `TRUE/FALSE` value is generated by recursively applying boolean operators to <*sub.expr*>s. In turn, each <*sub.expr*> is created by a <*comparison*> of a <*var*> in Iris features and another value created using <*func.var*>.

A few examples of the grammar generated expression, formatted through the `pretty.print` function, are as follows:

```
pretty.print <- function(expr) cat(gsub("|", "|\n\t",
  gsub("&", "&\n\t", as.character(expr), fixed = TRUE), fixed = TRUE),
  "\n")

pretty.print(GrammarRandomExpression(grammarDef))

## ifelse(((Petal.Width > Petal.Length) &
##   (Sepal.Length >= sd(Petal.Length))) &
##   (Petal.Length == 5), "versicolor", "other")

pretty.print(GrammarRandomExpression(grammarDef))

## ifelse((Sepal.Width == min(Petal.Length)) |
##   (Sepal.Length <= sd(Sepal.Length)), "versicolor", "other")
```

The GE optimisation is performed by:

```
set.seed(10)
ge <- GrammaticalEvolution(grammarDef, ClassifyFitFunc)
```

```
expr <- ge$best$expression
pretty.print(expr)

## ifelse(((Sepal.Width >= max(Sepal.Length)) |
##   (Petal.Width <= sd(Petal.Length))) &
##   (Petal.Length >= Sepal.Width), "versicolor", "other")

err <- sum(eval(expr, envir=iris) != iris$Species)
err

## [1] 6
```

Value	Minimum	Median	Maximum
Error	4	8	22
Generations	1000	1000	1000
Time (s)	12.56	12.87	13.24

TABLE A.2: Summary of GE's performance for 100 runs of the classification example.

The classification results are visualised in Figure A.1.

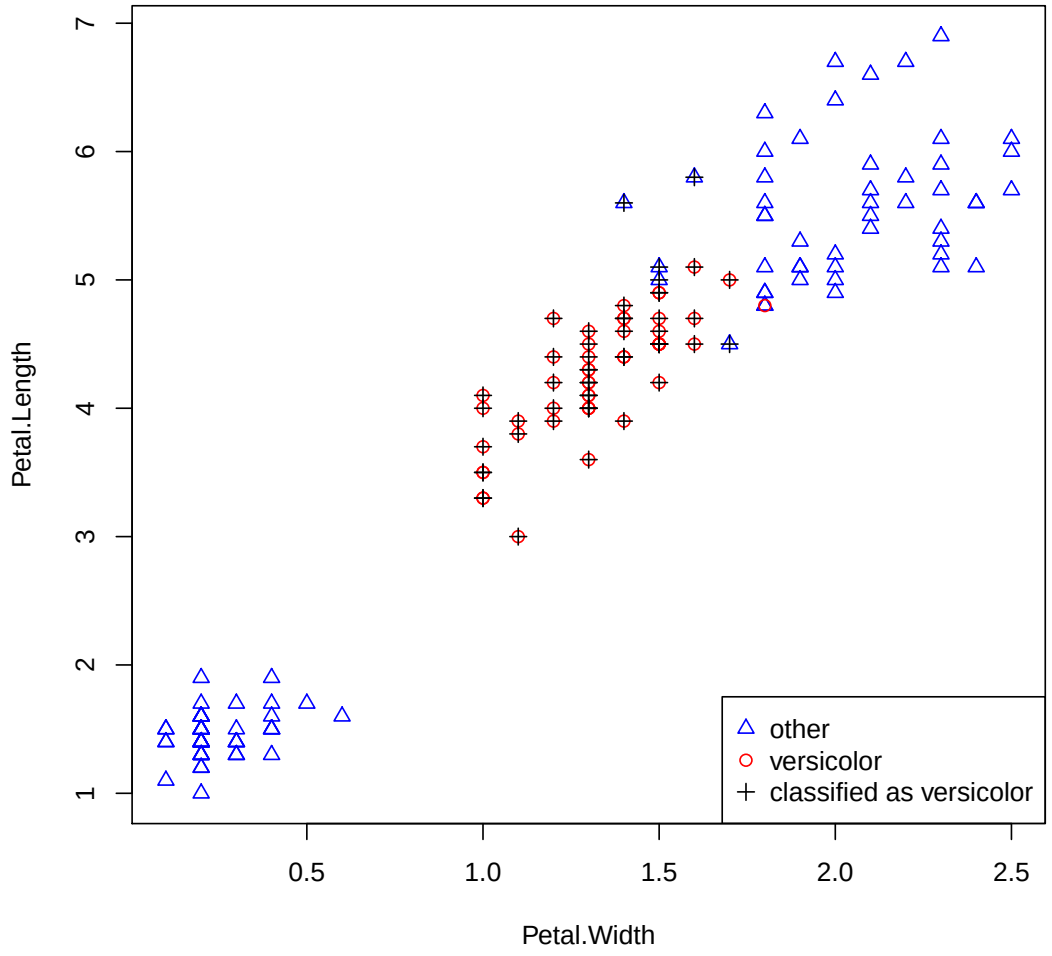


FIGURE A.1: Classification of *Iris versicolor* using GE.

Table A.2 summarises the performance of GE classifier for 100 executions. As no termination condition was given, all of the runs terminated only after reaching the maximum allowed number of generations. It is evident that on average, GE is able to find an acceptable expression with in this limit.

A.3.3 Symbolic Regression and Feature Generation

Symbolic regression is the process of discovering a function, in analytical form, which fits a given set of data. Commonly, evolutionary algorithms such as GP and GE are used for this task. Symbolic regression suffers from a possibly infinite, non-smooth and non-convex search space, and therefore is not widely used in machine learning.

Feature generation is the process of deriving new features from existing features [263]. In this technique, an evolutionary algorithm is used to generate and combine results of multiple independently discovered expression, e.g., by using a linear combination of GP results [264, 265], or by using non-linear function estimators applied to GE [13]. This can be considered a type of machine learning and symbolic regression hybrid, as the final learning model is constructed from combination of simpler *features* created through a process similar to symbolic regression.

For example, consider learning of the following sextic polynomial from numeric data:

$$f(X) = X^6 + X^5 + X^4 + X^3 + X^2 + X + 1$$

Evolving an expression that matches the observed data to this polynomial would either require a very well crafted grammar, or a successful search over a huge space, both of which suffer from extreme computationally expensive.

However, linear dependencies exist between components of this function. By designing a *multi-gene* chromosome, we can generate individual expressions independently and then combine them through a linear regression model to create the final expression. This effectively breaks the search space to several smaller ones, enabling a faster search over the whole space. Figure A.2 illustrates the difference between these two approaches.

To compare the symbolic regression and the feature generation with ordinary GE, two approaches are benchmarked using the same grammar:

```
ruleDef <- list(expr  = grule(op(expr, expr), func(expr), var),
                func  = grule(sin, cos, log, sqrt),
                op    = grule(‘+‘, ‘-‘, ‘*‘),
                var   = grule(X, X^n, n),
                n     = grule(1, 2, 3, 4))

grammarDef <- CreateGrammar(ruleDef)
```

The grammar can be used to generate different types of expressions:

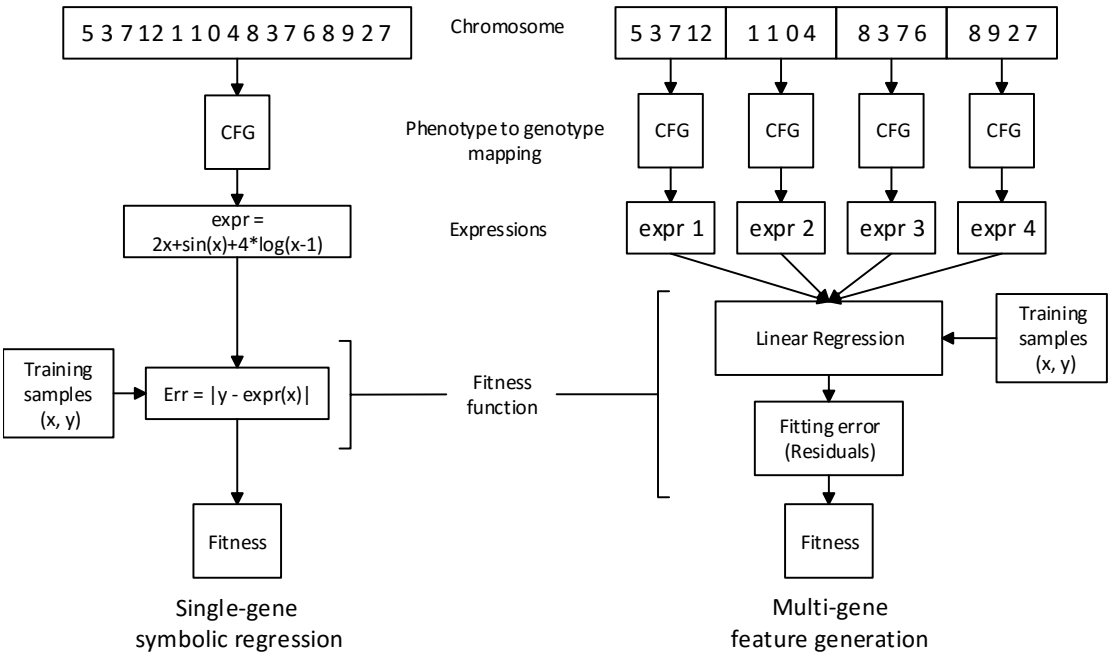


FIGURE A.2: Symbolic regression (using a single gene) vs feature generation (using multiple genes).

```
set.seed(0)
GrammarRandomExpression(grammarDef, numExpr = 3)


## [[1]]
## expression(log(2))
##
## [[2]]
## expression(sqrt(X * X) + cos(sqrt(cos(X^3))) - sqrt(log(sin(X))))
##
## [[3]]
## expression(X^3)
```

Obviously, this grammar is not tuned for the purpose of fitting high-degree polynomials.

A.3.3.1 Symbolic Regression

Firstly, symbolic regression is tested:

```
target.func <- function(X) X^6 + X^5 + X^4 + X^3 + X^2 + X + 1


X <- 1:10
```

```
Y <- target.func(X)

symRegCostFunc <- function(expr) {
  result <- suppressWarnings(eval(expr))

  if (any(is.nan(result)))
    return (Inf)

  return (mean((Y - result)^2))
}
```

The cost function handles invalid values (e.g., $\log(-1)$) by assigning a high cost to any expression with an invalid value. However, R may show warnings about `NaN`s being produced. To suppress these warnings, one can wrap the `eval` in the cost function inside `suppressWarnings`.

To allow the GE to have enough search space, the length of the chromosome is set to 60:

```
set.seed(0)
ge.single <- GrammaticalEvolution(grammarDef, symRegCostFunc,
  seqLen = 60,
  terminationCost = 1e-4)
```

This test is prone to getting stuck in a local minima and multiple restarts may be required to find the solution. Results often lack or include additional terms not in the target, e.g.,

```
ge.single


## Grammatical Evolution Search Results:
##   No. Generations:  1000
##   Best Expression:  (X^2 + X^3) * X^3
##   Best Cost:        21085073.8
```

The resulting expression, which can be simplified to $X^6 + X^5$, lacks several components and hence exhibits a high residual error.

A.3.3.2 Feature Generation

The second approach uses `GrammaticalEvolution`'s `numExpr` option to generate multiple expressions. Here, `numExpr = 5` is set, and for a fair comparison, the length allocated to each sequence `seqLen` is also reduced from 60 to 12. `GrammaticalEvolution` will still use a chromosome with length of 60, but this is divided into 5 parts (i.e., the genes), each of which are used individually to generate up to five valid expressions. A simple linear model is then applied to fit these expressions to data and the fitting residuals are reported as error.

For evaluating multiple expression, the function `EvalExpressions` offers a simpler interface compared to `eval`:

```
X <- 1:10
Y <- target.func(X)

fitLinearModel <- function(expr.list) {
  vals <- EvalExpressions(expr.list)

  if (any(is.nan(unlist(vals))) | any(is.infinite(unlist(vals))))
    return(NULL)

  mdl <- lm(Y ~ ., cbind(as.data.frame(vals), Y = Y))

  return (mdl)
}

fitnessFunction <- function(expr.list) {
  mdl <- fitLinearModel(expr.list)

  if (class(mdl) != "lm")
    return (Inf)

  return(mean(residuals(mdl)^2))
}
```

The `fitnessFunction` uses `fitLinearModel` to create a linear model of generated expressions to data. The model is then fit to the data, and the MSE of residuals are returned as its cost.

All other GE parameters (i.e., population size, mutation chance, termination condition, etc.) are kept the same:

```
set.seed(10)
ge.multi <- GrammaticalEvolution(grammarDef, fitnessFunction,
  seqLen = 12,  numExpr = 5,
  terminationCost = 1e-4)
```

This approach is clearly better at finding a close approximation to the target:

```
ge.multi
```

```
## Grammatical Evolution Search Results:
##   No. Generations:  12
##   Best Expressions: X + X^3 * X * ((X + 2) * 1)
##                   : X^4 * X^2
##                   : X^4
##                   : X^2
##                   : X^3
##   Best Cost:        6.20330039637389e-23
```

```
expr <- ge.multi$best$expression
mdl <- fitLinearModel(expr)
mdl
```

```
##
## Call:
## lm(formula = Y ~ ., data = cbind(as.data.frame(vals), Y = Y))
##
## Coefficients:
## (Intercept)        expr1        expr2
##           1            1            1
##       expr3        expr4        expr5
##          -1            1            1
```

```
X <- seq(1, 10, length.out = 40)
pred <- predict(mdl, newdata = EvalExpressions(expr))
err <- mean((target.func(X) - pred)^2)
err
```

	Symbolic regression			Feature generation		
Value	Minimum	Median	Maximum	Minimum	Median	Maximum
Error	2.14×10^5	9.12×10^8	6.15×10^9	0.00	0.00	1.48
Generations	1000	1000	1000	5	25.5	200
Time (s)	23.24	24.73	25.17	6.59	30.65	309.93

TABLE A.3: Performance of symbolic regression vs feature generation using GE, compared over 100 runs.

```
## [1] 4.64618e-21
```

In the results above, all elements of the sextic equation are found within five expressions. Three of them (X^2, X^3, X^4 and X^6) are found separately, and the other expression, `X + X^3 * X * ((X + 2) * 1)`, contains the linear combination $X+2X^4+X^5$. As X^4 is already present separately, the linear regression can extract and combine all elements with the correct y-intercept. Consequently, the regression model $\hat{f}(X)$ perfectly matches the original model:

$$\begin{aligned}\hat{f}(X) &= 1 + (X + X^3 \times X \times (X + 2)) + X^4 \times X^2 - X^4 + X^2 + X^3 \\ &= 1 + X + X^2 + X^3 + X^4 + X^5 + X^6\end{aligned}$$

A.3.3.3 Comparison

To test the stochastic performance of GE with a single and multiple genes, each method was run 100 times and their error from the target equation was noted. The results are presented in Table A.3. The results show major improvements in error, from an average 9.12×10^{10} for symbolic regression to a worst case of 1.48 for the feature generation approach. In comparison, the average time required to process both approaches was almost equal.

A.4 Summary

The *gramEvol* package allows creation of native R programs using GE. After specifying a grammar and evaluation function, users can employ GE techniques with little additional code. Parallel execution is also supported via parallel computing functions within R.

One disadvantage of GE lies in its stochastic nature, as it does not guarantee the convergence to the global optima. The *gramEvol* package includes an exhaustive search option which can ensure an optimal solution at the expense of computation time.

Appendix B

Sensitivity Analysis of Minimal Cost Hedging to Client Flow Forecast Error

B.1 Introduction

In Chapter 6, a general purpose formulation was proposed to compute the effect of forecast error on the final cost of any LQ controller. This formulation was then used as an error measure for choosing the best client flow forecaster through grammatical evolution.

In this appendix, the effects of client flow forecast error on the cost function is derived by analysing the sensitivity of the controller's cost function to changes in client flow forecasts. This was originally built up on an early observation, where improving the hit-rate or RMSE of the client flow prediction did not necessarily improve the hedging results.

For simplicity and focusing on only the effects of the client flow, the following assumptions were made to simplify derivation:

- Only the *minimum transaction cost* case (i.e., $\lambda = 0$ in Chapter 3 formulations) is considered.
- Only effects of client flow variations are studied, and other parameters are assumed to be constant.
- Only a univariate model is studied. This is possible as currently there is no dependency between the client flow of different currencies in the transaction cost model.

- No constraints are assumed for the positions and hedging actions.
- Time invariant δ (i.e, bid-ask spread coefficient) is assumed.
- Forecasts are performed once, and are not updated at each time-step.

B.2 Hedging Cost Function

For a single currency, let $h(i) = h, i \in [0, \ldots, N]$ be the hedging actions. According to the assumptions and definitions of Section 3.3.1, the cost from the start to the end of trading session (i.e., $t = 1$ to $t = N$) is

$$\begin{aligned} C &= \sum_{i=0}^{N} f_{cost}(h(i)) \\ &= \sum_{i=0}^{N} \delta h^2(i) \end{aligned} \tag{B.1}$$

where $f_{cost}(x) = \delta x^2$ is the quadratic transaction cost of a trade with size x.

As described in (3.2), at the end of trading session all positions are to be closed, or $x(N+1) = 0$, hence $h(N) = -(x(N) + f(N))$, and therefore

$$C = \sum_{i=0}^{N-1} \delta h^2(i) + \delta(x(N) + f(N))^2 \tag{B.2}$$

where $f(t)$ is the client flow, and $x(t)$, the current position's of the dealer, is obtained by expanding the dealer's state-space equation (3.1):

$$x(t) = x(0) + \sum_{i=0}^{t-1} f(i) + \sum_{i=0}^{t-1} h(i).$$

B.3 The Minimum Transaction Cost

In Chapter 3, especially Section 3.5, it was shown that the minimum transaction cost is obtained when the hedging actions are distributed equally in time, or $h(i) = h; i \in$

$[0, 1, \cdots, N]$. This can be re-evaluated by computing the minima of (B.2):

$$
\begin{aligned}
&\frac{\partial C}{\partial h} = 0 \\
\longrightarrow\ & 2\delta N h + 2\delta\Big(x(0) + \sum_{i=0}^{N} f(i) + Nh\Big)\frac{\partial\Big(x(0) + \sum_{i=0}^{N} f(i) + Nh\Big)}{\partial h} = 0 \\
\longrightarrow\ & 2\delta N h + 2\delta\Big(x(0) + \sum_{i=0}^{N} f(i) + Nh\Big)\Big(N\Big) = 0 \\
\longrightarrow\ & h = \frac{-1}{N+1}\Big(x(0) + \sum_{i=0}^{N} f(i)\Big)
\end{aligned}
$$

At any time t, individual hedging actions $h(t)$ can be rewritten based on current $x(i)$ and future $f(i), i \in [t, \ldots, N]$:

$$
\begin{aligned}
h(t) &= \left(\frac{N-(t-1)}{N-(t-1)}\right)\frac{-1}{N+1}\left(x(0) + \sum_{i=0}^{N} f(i)\right) \\
&= \frac{-1}{N-(t-1)}\left(\frac{N-(t-1)}{N+1}\left(x(0) + \sum_{i=0}^{N} f(i)\right)\right) \\
&= \frac{-1}{N-(t-1)}\left(x(0) + \sum_{i=0}^{N} f(i) + \frac{-t}{N+1}\left(x(0) + \sum_{i=0}^{N} f(i)\right)\right) \\
&= \frac{-1}{N-(t-1)}\left(x(0) + \sum_{i=0}^{t-1} f(i) + \sum_{i=t}^{N} f(i) + \sum_{i=0}^{t-1} h(i)\right) \\
&= \frac{-1}{N-(t-1)}\left(x(0) + \sum_{i=0}^{t-1} f(i) + \sum_{i=0}^{t-1} h(i) + \sum_{i=t}^{N} f(i)\right) \\
&= \frac{-1}{N-(t-1)}\left(x(t) + \sum_{i=t}^{N} f(i)\right)
\end{aligned}
\tag{B.4}
$$

With $f(i), i \in \{t, \ldots, N\}$ known (i.e., using a prescient forecaster), the minimum cost will be

$$
\begin{aligned}
C_{min} &= \delta\sum_{i=0}^{N}(h(t))^2 \\
&= \delta\sum_{i=0}^{N}\left(\frac{1}{N+1}\Big(x(0) + \sum_{i=0}^{N} f(i)\Big)\right)^2 \\
&= \frac{\delta}{N+1}\Big(x(0) + \sum_{i=0}^{N} f(i)\Big)^2.
\end{aligned}
$$

B.4 Hedging with Forecasts

To simplify derivation, it is assumed that the prediction is performed only once, and is not updated once new observations arrive. Let $\hat{f}(t)$ denote the prediction of flow at time t. The minimum transaction cost hedging action at time t can be determined similar to (B.4), with the addition of separating observed and predicted client flows:

$$h(t) = \frac{-1}{N-(t-1)}\Big(x(0) + \sum_{i=0}^{t-1} f(i) + \sum_{i=0}^{t-1} h(i) + \sum_{i=t}^{N} \hat{f}(i)\Big)$$

For $t = 0$,

$$h(0) = \frac{-1}{N+1}\Big(x(0) + \sum_{i=0}^{N} \hat{f}(i)\Big).$$

Recursively, one can expand $h(1)$ by replacing $h(0)$ with the above:

$$\begin{aligned}
h(1) &= \frac{-1}{N}\Big(x(0) + f(0) + h(0) + \sum_{i=1}^{N} \hat{f}(i)\Big) \\
&= \frac{-1}{N}\Big(x(0) + f(0) + \frac{-1}{N+1}\Big(x(0) + \sum_{i=0}^{N} \hat{f}(i)\Big) + \sum_{i=1}^{N} \hat{f}(i)\Big) \\
&= \frac{-1}{N}\Big(\frac{N}{N+1}x(0) + f(0) - \frac{1}{N+1}\hat{f}(0) + \frac{N}{N+1}\sum_{i=1}^{N} \hat{f}(i)\Big) \\
&= \frac{-1}{N+1}x(0) + \frac{-1}{N}\Big(f(0) - \frac{1}{N+1}\hat{f}(0)\Big) + \frac{-1}{N+1}\sum_{i=0}^{N} \hat{f}(i)
\end{aligned}$$

And similarly for $h(2)$:

$$
\begin{aligned}
h(2) &= \frac{-1}{N-1}\left(x(0)+f(0)+f(1)+h(0)+h(1)+\sum_{i=2}^{N}\hat{f}(i)\right) \\
&= \frac{-1}{N-1}\Bigg(x(0)+f(0)+f(1)+\sum_{i=2}^{N}\hat{f}(i)+ \\
&\quad \frac{-1}{N+1}\Big(x(0)+\sum_{i=0}^{N}\hat{f}(i)\Big)+ \\
&\quad \frac{-1}{N+1}\Big(x(0)+\sum_{i=1}^{N}\hat{f}(i)\Big)+\frac{-1}{N}\Big(f(0)-\frac{1}{N+1}\hat{f}(0)\Big)\Bigg) \\
&= \frac{-1}{N-1}\Bigg(\frac{(N+1)-2}{N+1}x(0)+\frac{N-1}{N+1}\sum_{i=2}^{N}\hat{f}(i)+\frac{N-1}{N}f(0)+f(1)+ \\
&\quad \frac{-(N-1)}{N(N+1)}\hat{f}(0)+\frac{-2}{N+1}\hat{f}(1)\Bigg) \\
&= \frac{-1}{N+1}x(0)+\frac{-1}{N+1}\sum_{i=2}^{N}\hat{f}(i)+\frac{-1}{N}(f(0)-\frac{1}{N+1}\hat{f}(0)) \\
&\quad +\frac{-1}{N-1}(f(1)-\frac{2}{N+1}\hat{f}(1))
\end{aligned}
$$

General, any flow forecast $\hat{f}(i)$ is hedged by $\frac{-(i+1)}{N+1}\hat{f}(i)$ until the actual $f(i)$ is observed. Consequently, the hedging has to be corrected by $\frac{1}{N-i}(f(i)-\frac{i+1}{N+1}\hat{f}(i))$ after observing the real value of $f(i)$:

$$
\begin{aligned}
h(t) &= \frac{-1}{N+1}x(0)+\frac{-1}{N+1}\sum_{i=t}^{N}\hat{f}(i)+\sum_{i=0}^{t-1}\frac{-1}{N-i}(f(i)-\frac{i+1}{N+1}\hat{f}(i)) \\
&= \frac{-1}{N+1}\left(x(0)+\sum_{i=t}^{N}\hat{f}(i)+\sum_{i=0}^{t-1}\frac{1}{N-i}\Big((N+1)f(i)-(i+1)\hat{f}(i)\Big)\right)
\end{aligned}
\tag{B.5}
$$

B.4.1 Minimum Transaction Cost with Forecasts

When prediction is used, the cost function can be rewritten by replacing hedging actions in (B.1) with (B.5), resulting in:

$$
C = \frac{\delta}{(N+1)^2}\sum_{t=0}^{N}\left[x(0)+\sum_{i=t}^{N}\hat{f}(i)+\sum_{i=0}^{t-1}\frac{1}{N-i}\Big((N+1)f(i)-(i+1)\hat{f}(i)\Big)\right]^2 \tag{B.6}
$$

B.5 Sensitivity to Forecasts

B.5.1 Hedging Action Sensitivity

The sensitivity of hedging actions from (B.5) to a forecast at time $k \geq 0$ (i.e., $\hat{f}(k)$) is obtained by

$$\frac{\partial h}{\partial \hat{f}(k)}(t) = \frac{-1}{N+1}\left(u(k-t) - \frac{k+1}{N-k}u((t-1)-k)\right) \tag{B.7}$$

where $u(x)$ is the step function:

$$u(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

B.5.2 Sensitivity of the Minimum Transaction Cost

Sensitivity of the minimum transaction cost to a forecast at time $k \geq 0$ (i.e., $\hat{f}(k)$) is obtained by

$$\begin{aligned}\frac{\partial C}{\partial \hat{f}(k)} &= \delta \sum_{t=0}^{N} \frac{\partial (h(t)^2)}{\partial \hat{f}(k)} \\ &= \delta \sum_{t=0}^{N} \left[2h(t)\frac{\partial h(t)}{\partial \hat{f}(k)}\right] \\ &= \frac{-2\delta}{N+1}\sum_{t=0}^{N}\left[h(t)\left(u(k-t) - \frac{k+1}{N-k}u((t-1)-k)\right)\right].\end{aligned}$$

This can be expanded by replacing $h(t)$ from (B.5):

$$\begin{aligned}\frac{\partial C}{\partial \hat{f}(k)} =& \frac{2\delta}{(N+1)^2}\sum_{t=0}^{N} \\ &\left[x(0) + \sum_{i=t}^{N}\hat{f}(i) + \sum_{i=0}^{t-1}\frac{1}{N-i}\left((N+1)f(i) - (i+1)\hat{f}(i)\right)\right] \\ &\left[u(k-t) - \frac{k+1}{N-k}u((t-1)-k)\right],\end{aligned}$$

Simplifying the summations in the first bracket, factoring out $f(t)$ and $\hat{f}(t)$, and replacing the step function in the second bracket with minimum functions, reveals the

following linear function:

$$\frac{\partial C}{\partial \hat{f}(k)} = \frac{2\delta}{N+1} \sum_{t=0}^{N} \left(\frac{\min(k,t)+1}{N - min(k,t)} \right) \left(\hat{f}(t) - f(t) \right) \tag{B.8}$$

B.5.3 Building a Sensitivity Matrix

Eq. (B.8) can be vectorised by defining $\boldsymbol{f} = [f(0)\ f(1)\ \cdots\ f(N-1)]$ as the observed client flow vector, and $\hat{\boldsymbol{f}} = [\hat{f}(0)\ \hat{f}(1)\ \cdots\ \hat{f}(N-1)]$ as the predicted flow vector. The sensitivity matrix $\boldsymbol{\Theta}_{C\ (N\times N)}$ can be defined as

$$\boldsymbol{\Theta}_C = |\theta_{k,t}| = \frac{\min(k,t)}{N - min(k,t) + 1} \quad t,k = 1,\cdots,N. \tag{B.9}$$

Notice that different time indexing of $[1,\cdots,N]$ versus $[0,\cdots,N-1]$ in (B.8) is compensated using an additional "+1".

Analytically, the sensitivity matrix can be instantiated as follows:

$$\boldsymbol{\Theta}_C = \frac{2}{N+1}\delta \begin{bmatrix} \frac{1}{N} & \frac{1}{N} & \frac{1}{N} & \frac{1}{N} & \cdots & \frac{1}{N} & \frac{1}{N} \\ \frac{1}{N} & \frac{2}{N-1} & \frac{2}{N-1} & \frac{2}{N-1} & \cdots & \frac{2}{N-1} & \frac{2}{N-1} \\ \frac{1}{N} & \frac{2}{N-1} & \frac{3}{N-2} & \frac{3}{N-2} & \cdots & \frac{3}{N-2} & \frac{3}{N-2} \\ \frac{1}{N} & \frac{2}{N-1} & \frac{3}{N-2} & \frac{4}{N-3} & \cdots & \frac{4}{N-3} & \frac{4}{N-3} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \frac{1}{N} & \frac{2}{N-1} & \frac{3}{N-2} & \frac{4}{N-3} & \cdots & \frac{N-1}{2} & \frac{N-1}{2} \\ \frac{1}{N} & \frac{2}{N-1} & \frac{3}{N-2} & \frac{4}{N-3} & \cdots & \frac{N-1}{2} & \frac{N}{1} \end{bmatrix} \tag{B.10}$$

The difference, of the hedging cost using the forecasts, from the hedging cost using a prescient forecaster, can be approximated by integrating the linear derivative obtained in (B.8), i.e.,

$$\Delta C = \frac{1}{2}(\hat{\boldsymbol{f}} - \boldsymbol{f})^{\mathrm{T}} \boldsymbol{\Theta}_C (\hat{\boldsymbol{f}} - \boldsymbol{f}) \tag{B.11}$$

which is similar to the concept of ΔJ explored in Chapter 6.

B.6 Comparison with ΔJ

ΔC assumes that the forecasts do not change between each iteration of the hedging algorithm. In comparison, the error measure in Chapter 6, ΔJ, was designed to consider the effects of forecast updates at each time-step.

Consider the following example for hedging with $N = 3$, and $t \in [0, 1, 2]$. The matrix form of ΔC requires the error vector

$$\tilde{\boldsymbol{E}} = [f(1) - \hat{f}(1),\ f(2) - \hat{f}(2),\ f(3) - \hat{f}(3)].$$

In comparison, the ΔJ in the example of Section 6.7.1 uses

$$\boldsymbol{E} = [f(1) - \hat{f}_1(1),\ f(2) - \hat{f}_1(2),\ f(3) - \hat{f}_1(3),\ f(2) - \hat{f}_2(2),\ f(3) - \hat{f}_2(3),\ f(3) - \hat{f}_3(3)]$$

where $\hat{f}_\tau(n)$ denotes the forecast of $f(n)$ at time $t = \tau$.

To convert ΔJ to ΔC, one can define and use matrix $\mathbf{M}$, such that $\boldsymbol{E} = \mathbf{M}\tilde{\boldsymbol{E}}$. For the previous example

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}.$$

Consequently, (B.9) and (6.15) are equal if $\boldsymbol{\Theta}_J = \frac{1}{2}\mathbf{M}^{\mathrm{T}}\boldsymbol{\Theta}_C\mathbf{M}$:

$$\begin{aligned} \Delta C =& \frac{1}{2}\tilde{\boldsymbol{E}}^{\mathrm{T}}\boldsymbol{\Theta}_C\tilde{\boldsymbol{E}} \\ =& \frac{1}{2}\boldsymbol{E}^{\mathrm{T}}\mathbf{M}^{\mathrm{T}}\boldsymbol{\Theta}_C\mathbf{M}\boldsymbol{E} \\ =& \boldsymbol{E}^{\mathrm{T}}(\frac{1}{2}\mathbf{M}^{\mathrm{T}}\boldsymbol{\Theta}_C\mathbf{M})\boldsymbol{E} \\ =& \boldsymbol{E}^{\mathrm{T}}\boldsymbol{\Theta}_J\boldsymbol{E} \\ =& \Delta J \end{aligned}$$

This can be verified numerically. In the example of Section 6.7.1, with $\delta = 1$,

$$\boldsymbol{\Theta}_J = \begin{bmatrix} 0.083 & 0.083 & 0.083 & 0 & 0 & 0 \\ 0.083 & 0.083 & 0.083 & 0 & 0 & 0 \\ 0.083 & 0.083 & 0.083 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.167 & 0.167 & 0 \\ 0 & 0 & 0 & 0.167 & 0.167 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.5 \end{bmatrix}.$$

Using the same δ and $N = 3$, (B.10) results in

$$\begin{aligned}\boldsymbol{\Theta}_C &= \frac{2}{3+1}\begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{2}{2} & \frac{2}{2} \\ \frac{1}{3} & \frac{2}{2} & \frac{3}{1} \end{bmatrix} \\ &= 2\begin{bmatrix} 0.083 & 0.083 & 0.083 \\ 0.083 & 0.25 & 0.25 \\ 0.083 & 0.25 & 0.75 \end{bmatrix}.\end{aligned}$$

A matrix multiplication shows that $\boldsymbol{\Theta}_J = \frac{1}{2}\mathbf{M}^{\mathrm{T}}\boldsymbol{\Theta}_C\mathbf{M}$.

Appendix C

State-space Model for Scheduling a Grid Connected Battery System

C.1 Introduction

In Chapter 8, a state-space model was proposed to model the dynamics of a grid connected battery energy storage system (BESS) with local renewable energy generation. The model, presented in (8.1), assumed that local generation is never more than local demand, and therefore overcharging the battery would never occur without the controller's explicit action. This assumption was based on the real data used for simulation.

In contrast, in many real-world systems, local generation may surpass local demand, resulting in energy that has to be either stored in the battery, or go to waste. Additionally, one can never overdraw the battery by using more energy than what is stored in the system. In the iDemand system this was enforced by allowing a 20% safety margin for the battery capacity.

Consequently, a more realistic battery state-space model is

$$\begin{aligned} b_{t+1} &= \min(\max(b_t + g_t + s_t - l_t, b_{min}), b_{max}) \\ u_t &= b_{t+1} - b_t \end{aligned} \tag{C.1}$$

where

- b_t is the battery charge state at time t,
- s_t is the local renewable generation (e.g., solar PV) power,
- l_t is the load power,
- g_t is the power demanded from the grid,

- b_{min} and b_{max} are the minimum and maximum allowed charge states of the battery, and
- u_t is the battery charge/discharge rate.

Additionally, the constraint

$$|u_t| \leq u_{max}$$

restricts the charge/discharge rate.

In this appendix, this model is analysed and an optimisation scheme without the limitations of Chapter 8 is presented.

C.2 System Dynamics

One can remove the minimum and maximum of (C.1) by using a slack variable. Let w_t be the energy wasted when the battery is fully charged (i.e., $b_t + g_t + s_t - l_t > b_{max}$). The overdraw case can be handled by g_t and the constraint $b_{t+1} \geq b_{min}$.

Consequently, (C.1) can be rewritten as

$$\begin{aligned}
b_{t+1} &= b_t + g_t + s_t - l_t - w_t, \\
u_t &= g_t + s_t - l_t - w_t, \\
b_{min} &\leq b_t + g_t + s_t - l_t - w_t \leq b_{max}, \\
0 &\leq w_t, \\
0 &\leq g_t, \\
-u_{max} &\leq u_t \leq u_{max}.
\end{aligned}$$

The cost function requires no further changes, and is described by

$$J = \mathbb{E}\left[\sum g_i^2\right] + \lambda \mathbb{E}[\sum g_i p_i]$$

where p_t is the grid usage price at time t, and the parameter $0 \leq \lambda \leq \infty$ selects between peak shaving when $\lambda = 0$, and cost-saving when $\lambda \to \infty$.

C.3 Matrix Notation

The state-space model can be vectorised as

$$\begin{aligned}
\boldsymbol{b} &= b_0\vec{\mathbf{1}} + \boldsymbol{\Sigma g} + \boldsymbol{\Sigma s} - \boldsymbol{\Sigma l} - \boldsymbol{\Sigma w} \\
\boldsymbol{u} &= \boldsymbol{g} + \boldsymbol{s} - \boldsymbol{l} - \boldsymbol{w} \\
b_{min}\vec{\mathbf{1}} &\leq \boldsymbol{b} + \boldsymbol{g} + \boldsymbol{s} - \boldsymbol{l} - \boldsymbol{w} \leq b_{max}\vec{\mathbf{1}} \\
\vec{\mathbf{0}} &\leq \boldsymbol{w} \\
\vec{\mathbf{0}} &\leq \boldsymbol{g} \\
-u_{max}\vec{\mathbf{1}} &\leq \boldsymbol{u} \leq u_{max}\vec{\mathbf{1}}
\end{aligned}$$

where b_0 is the battery's initial state, $\vec{\mathbf{0}}$ is a vector of 0's, $\vec{\mathbf{1}}$ is a vector of 1's, $\boldsymbol{\Sigma}$ is a lower triangular matrix of 1's (without diagonal elements).

Similarly, the cost function is rewritten as

$$J = \mathbb{E}\left[\boldsymbol{g}^{\mathrm{T}}\boldsymbol{g}\right] + \lambda\mathbb{E}[\boldsymbol{g}^{\mathrm{T}}\boldsymbol{p}]$$

where $\boldsymbol{p}$ is the grid prices vector.

C.4 Open-loop Optimisation

The model's controllable inputs are $\boldsymbol{w}$ and $\boldsymbol{g}$. Consequently, the open-loop optimisation can be formulated as a stochastic quadratic programming problem:

$$\begin{aligned}
\underset{\boldsymbol{g},\boldsymbol{w}}{\text{argmin}} \quad & \mathbb{E}\left[\boldsymbol{g}^{\mathrm{T}}\boldsymbol{g}\right] + \lambda\mathbb{E}[\boldsymbol{g}^{\mathrm{T}}\boldsymbol{p}] \\
\text{subject to} \quad & \boldsymbol{b} = b_0\vec{\mathbf{1}} + \boldsymbol{\Sigma g} + \boldsymbol{\Sigma s} - \boldsymbol{\Sigma l} - \boldsymbol{\Sigma w} \\
& \boldsymbol{u} = \boldsymbol{g} + \boldsymbol{s} - \boldsymbol{l} - \boldsymbol{w} \\
& b_{min}\vec{\mathbf{1}} \leq \boldsymbol{b} + \boldsymbol{g} + \boldsymbol{s} - \boldsymbol{l} - \boldsymbol{w} \\
& -b_{max}\vec{\mathbf{1}} \leq -\boldsymbol{b} - \boldsymbol{g} - \boldsymbol{s} + \boldsymbol{l} + \boldsymbol{w} \\
& \vec{\mathbf{0}} \leq \boldsymbol{w} \\
& \vec{\mathbf{0}} \leq \boldsymbol{g} \\
& -u_{max}\vec{\mathbf{1}} \leq -\boldsymbol{u} \\
& -u_{max}\vec{\mathbf{1}} \leq -\boldsymbol{u}
\end{aligned}$$

This can be solved in a manner similar to Chapter 8.

Appendix D

Cloud Computing for Battery Energy System Management

The peak shaving and cost saving system proposed in Chapter 8 is suitable for deployment on cloud computing environments. The architecture of this environment, presented in Figure D.1, is as follows:

- The control and data acquisition system (SCADA), located on the local site, collects data from different subsystems, and issues charge/discharge commands to the battery storage system. As most of the computational and storage capacity is offloaded to the cloud, this system can be implemented using a low power microcontroller system; however, a connection to the cloud is required.
- The cloud-based forecasting and control system provides its services through a web gateway. Its two major services include demand forecast model selection, and battery scheduling using MPC. Internally, both of these systems are based on a single forecasting engine, which supports the grammar based models proposed in Chapter 5.
- The forecasting engine can offload its computational load to a high performance computing (HPC) systems, and store and retrieve data from a cloud-based database. These services are available from different providers, including Amazon EC2 [266], Microsoft Azure [267], and Google Cloud [268].
- Data from third party providers, including weather forecasts, insolation predictions, and grid prices can be obtained to improve the forecasting results. The data is either downloaded from public web-based repositories (as performed in this thesis), or acquired through commercial business-to-business (B2B) services.

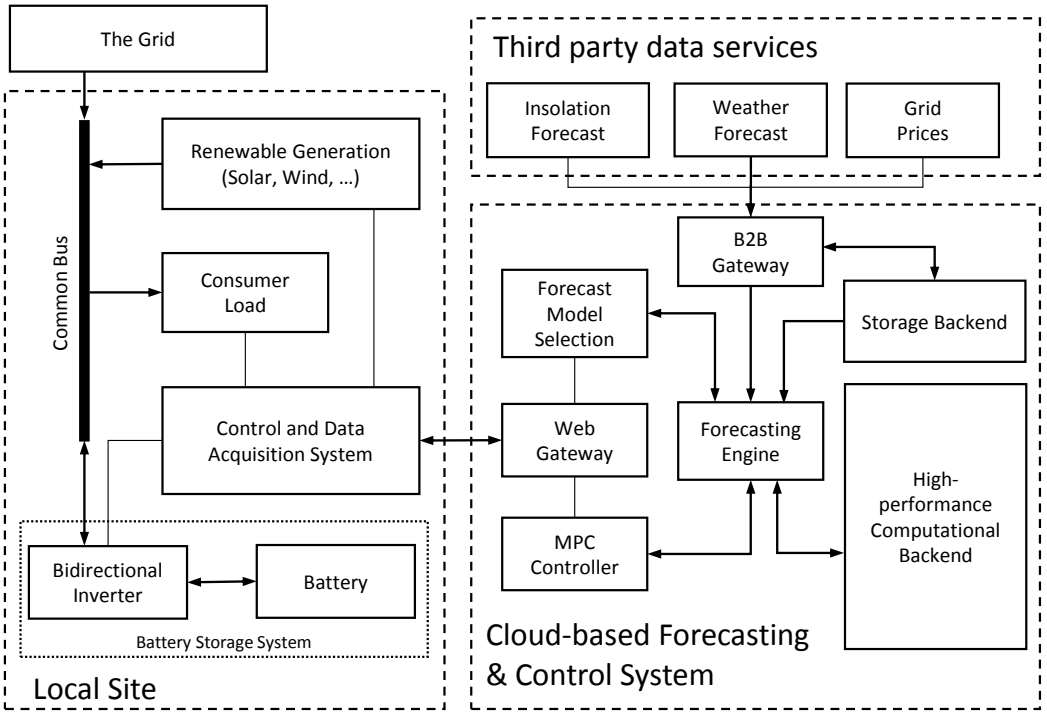


FIGURE D.1: Architecture of a cloud-based electricity peak shaving and cost saving system.

Bibliography

[1] Standards Australia/Standards New Zealand Joint Committee. *AS/NZS ISO 31000:2009 Risk Management – Principles and Guidelines*. Standards Australia, November 2009.

[2] David Hillson and Peter Simon. Options and actions (response planning). In *Practical Project Risk Management: The ATOM Methodology*, chapter 7, pages 83–92. Management Concepts, 2012.

[3] Steve Anthony. *Foreign Exchange in Practice*. Finance and Capital Markets Series. Palgrave Macmillan, Basingstoke, 2002.

[4] Barbara Rossi. Exchange rate predictability. *Journal of Economic Literature*, 51 (4):1063–1119, 2013.

[5] Irene Aldridge. *High-frequency Trading: A Practical Guide to Algorithmic Strategies and Trading Systems*, volume 459. John Wiley and Sons, 2009.

[6] Angel Pardo and Roberto Pascual. On the hidden side of liquidity. *The European Journal of Finance*, 18(10):949–967, 2012.

[7] Lawrence Harris. Order exposure and parasitic traders. Working paper, University of Southern California, 1997. URL http://www-bcf.usc.edu/~lharris/ACROBAT/Exposure.pdf.

[8] Francesco Borrelli, Alberto Bemporad, and Manfred Morari. *Predictive Control*. Cambridge University Press, 2014. URL http://www.mpc.berkeley.edu/mpc-course-material.

[9] Farzad Noorian and Philip H.W. Leong. Dynamic hedging of foreign exchange risk using stochastic model predictive control. In *IEEE Conference on Computational Intelligence for Financial Engineering & Economics (CIFEr)*, pages 441–448. IEEE, 2014.

[10] Farzad Noorian, Barry Flower, and Philip H.W. Leong. Stochastic receding horizon control for short-term risk management in foreign exchange. *Journal of Risk*, 2016. In press, Accepted 21 August 2015.

[11] Farzad Noorian, Anthony Mihirana de Silva, and Philip H.W. Leong. gramEvol: Grammatical Evolution in R. *Journal of Statistical Software*, 2016. In press, Accepted 4 April 2015.

[12] Farzad Noorian and Philip H.W. Leong. Expert assisted time-series forecasting using grammatical evolution. *Manuscript under preparation*, 2016.

[13] Anthony Mihirana de Silva, Farzad Noorian, Richard I. A. Davis, and Philip H. W. Leong. A hybrid feature selection and generation algorithm for electricity load prediction using grammatical evolution. In *IEEE 12th International Conference on Machine Learning and Applications ICMLA 2013, special session on Machine Learning in Energy Applications*, pages 211–217, 2013.

[14] Anthony Mihirana Silva, Richard I.A. Davis, Syed A. Pasha, and Philip H.W. Leong. Forecasting financial time series with grammar-guided feature generation. *Computational Intelligence*, 2016.

[15] Farzad Noorian. An evolutionary approach to GECCO 2015 industrial challenge - First prize winner's methodology. Technical report, The University of Sydney, 2015. URL http://www.ee.usyd.edu.au/cel/farzad/GECCO_IC_2015.

[16] Farzad Noorian and Philip H.W. Leong. On time series forecasting error measures for finite horizon control. *IEEE Transactions on Control System Technology*, 2016. Accepted with minor revision.

[17] Ramazan Gençay, Giuseppe Ballocchi, Michel Dacorogna, Richard Olsen, and Olivier Pictet. Real-time trading models and the statistical properties of foreign exchange rates. *International Economic Review*, 43(2):463–491, 2002.

[18] Monetary and Economic Department. *Foreign exchange turnover in April 2013: preliminary global results*. Bank For International Settlement, September 2013. URL http://www.bis.org/publ/rpfx13fx.pdf.

[19] The World Bank. Data - GDP (current US$), 2015. URL http://data.worldbank.org/indicator/NY.GDP.MKTP.CD/countries. accessed July 2015.

[20] Rachel Evans and David Goodman. *Currency Trading Jumps to Records in U.S., U.K. Amid Divergence*. Bloomberg.com, January 2015. URL http://www.bloomberg.com/news/articles/2015-01-27/currency-trading-jumps-to-records-in-u-s-u-k-amid-divergence.

[21] John N. Kallianiotis. The foreign exchange market. In *Exchange Rates and International Financial Economics: History, Theories, and Practices*, chapter 2. Palgrave Macmillan, Basingstoke, 2013.

[22] Michael Sager and Mark P. Taylor. Commercially available order flow data and exchange rate movements: Caveat emptor. *Journal of Money, Credit and Banking*, 40(4):583–625, 2008.

[23] Martin D.D. Evans and Richard K. Lyons. Forecasting exchange rate fundamentals with order flow. Working paper, National Bureau of Economic Research, July 2009.

[24] Martin D.D. Evans. Order flows and the exchange rate disconnect puzzle. *Journal of International Economics*, 80(1):58–71, 2010.

[25] Eugene F. Fama and Merton H. Miller. *The Theory of Finance*, volume 3. Dryden Press, Hinsdale, IL, 1972.

[26] Jonathan Clarke, Tomas Jandik, and Gershon Mandelker. The efficient markets hypothesis. In Robert C. Arffa, editor, *Expert Financial Planning: Advice from Industry Leaders*, chapter 9, pages 126–141. Wiley, New York, 2001.

[27] Martin Sewell. History of the efficient market hypothesis. Research note RN/11/04, University College London, 2011.

[28] Burton G. Malkiel. The efficient market hypothesis and its critics. *Journal of Economic Perspectives*, 17(1):59–82, 2003.

[29] Michael D. Godfrey, Clive W.J. Granger, and Oskar Morgenstern. The random-walk hypothesis of stock market behaviour. *Kyklos*, 17(1):1–30, 1964.

[30] Michael C. Jensen. Some anomalous evidence regarding market efficiency. *Journal of Financial Economics*, 6(2–3):95–101, 1978.

[31] Jeremy Berkowitz and Lorenzo Giorgianni. Long-horizon exchange rate predictability? *Review of Economics and Statistics*, 83(1):81–91, 2001.

[32] Jiahan Li, Ilias Tsiakas, and Wei Wang. Predicting exchange rates out of sample: Can economic fundamentals beat the random walk? *Journal of Financial Econometrics*, 13(2):293–341, 2014.

[33] Charles Goodhart. The foreign exchange market: A random walk with a dragging anchor. *Economica*, 55:437–460, 1988.

[34] Yin-Wong Cheung, Menzie D. Chinn, and Antonio Garcia Pascual. Empirical exchange rate models of the nineties: Are any fit to survive? Working Paper

9393, National Bureau of Economic Research, December 2002. URL http://www.nber.org/papers/w9393.

[35] Andrew W. Lo. The adaptive markets hypothesis. *The Journal Of Portfolio Management*, 30(5):15–29, 2004.

[36] Rob J. Hyndman and George Athanasopoulos. *Forecasting: Principles and Practice*. Otexts, 2013. URL https://www.otexts.org/fpp.

[37] Christian Ullrich, Detlef Seese, and Stephan Chalup. Investigating FX market efficiency with support vector machines. In *Quantitative Methods in Finance Confrenece*, pages 13–16, 2006.

[38] Tomáš Tokár and Denis Horváth. Market inefficiency identified by both single and multiple currency trends. *Physica A: Statistical Mechanics and its Applications*, 391(22):5620–5627, 2012.

[39] Allan Timmermann and Clive W.J. Granger. Efficient market hypothesis and forecasting. *International Journal of Forecasting*, 20(1):15–27, 2004.

[40] Armin Shmilovici, Yoav Kahiri, Irad Ben-Gal, and Shmuel Hauser. Measuring the efficiency of the intraday Forex market with a universal data compression algorithm. *Computational Economics*, 33(2):131–154, March 2009.

[41] Victor Golovtchenko. *Australian Insider Trading Duo Gets Jail Time After Trading Margin FX*. Finance Magnates Ltd, 2015. URL http://www.financemagnates.com/forex/analysis/australian-insider-trading-duo-gets-jail-time-after-trading-margin-fx/.

[42] Liam Vaughan, Gavin Finch, and Bob Ivry. *Secret Currency Traders' Club Devised Biggest Market's Rates*. Bloomberg.com, December 2013. URL http://www.bloomberg.com/news/articles/2013-12-19/how-secret-currency-traders-club-devised-biggest-market-s-rates.

[43] Rod Cross and Victor Kozyakin. Fact and fictions in FX arbitrage processes. In *Journal of Physics: Conference Series*, volume 585, pages 1–9. IOP Publishing, 2015.

[44] Markets Committee. *High-frequency trading in the foreign exchange market*. Bank For International Settlement, 2011. URL http://www.bis.org/publ/mktc05.pdf.

[45] John N. Kallianiotis. Foreign exchange risk and its management. In *Exchange Rates and International Financial Economics: History, Theories, and Practices*, chapter 6. Palgrave Macmillan, Basingstoke, 2013.

[46] Antoine Gautier, Frieda Granot, and Maurice Levi. Alternative foreign exchange management protocols: an application of sensitivity analysis. *Journal of Multinational Financial Management*, 12(1):1 – 19, 2002.

[47] Financial Management Group. *Australian Government Foreign Exchange Risk Management Guidelines*. Australian Government Department of Finance and Administration, 2006.

[48] International Trade Administration. *Trade Finance Guide: A Quick Reference for U.S. Exporters*. U.S. Department of Commerce, 2007.

[49] Steven M. Bragg. Foreign exchange risk management. In *Treasury Management: The Practitioner's Guide*, chapter 9, pages 207–238. John Wiley & Sons, Inc., 2010.

[50] John N. Kallianiotis. Foreign currency derivatives. In *Exchange Rates and International Financial Economics: History, Theories, and Practices*, chapter 5. Palgrave Macmillan, Basingstoke, 2013.

[51] Philippe Jorion. *Financial Risk Manager Handbook*. John Wiley & Sons, 2007.

[52] Abdulnasser Hatemi-J and Eduardo Roca. Calculating the optimal hedge ratio: Constant, time varying and the Kalman filter approach. *Applied Economics Letters*, 13(5):293–299, 2006.

[53] Chris D'Souza. How do Canadian banks that deal in foreign exchange hedge their exposure to risk? Working paper, Bank of Canada, 2002.

[54] Mark B. Garman and Steven W. Kohlhagen. Foreign currency option values. *Journal of international Money and Finance*, 2(3):231–237, 1983.

[55] Konstantin Volosov, Gautam Mitra, Fabio Spagnolo, and Cormac Lucas. Treasury management model with foreign exchange exposure. *Computational Optimization and Applications*, 32(1-2):179–207, 2005.

[56] Christian Ullrich. *Forecasting and Hedging in the Foreign Exchange Markets*, volume 623 of *Lecture Notes in Economics and Mathematical Systems*. Springer, 2009.

[57] James A. Primbs. Dynamic hedging of basket options under proportional transaction costs using receding horizon control. *International Journal of Control*, 82 (10):1841–1855, 2009.

[58] Alberto Bemporad, Leonardo Bellucci, and Tommaso Gabbriellini. Dynamic option hedging via stochastic model predictive control based on scenario simulation. *Quantitative Finance*, 14(10):1739–1751, 2014.

[59] Norman Josephy, Lucia Kimball, and Victoria Steblovskaya. Alternative hedging in a discrete-time incomplete market. *Journal of Risk*, 16(1):85–117, Fall 2013.

[60] Mark Rubinstein. Markowitz's "portfolio selection": A fifty-year retrospective. *Journal of Finance*, 52(3):1041–1045, June 2002.

[61] Harry Markowitz. Portfolio selection. *The Journal of Finance*, 7(1):77–91, March 1952.

[62] Petter N. Kolm, Reha Tütüncü, and Frank J. Fabozzi. 60 years of portfolio optimization: Practical challenges and current trends. *European Journal of Operational Research*, 234(2):356–371, 2014.

[63] Miguel Sousa Lobo, Maryam Fazel, and Stephen Boyd. Portfolio optimization with linear and fixed transaction costs. *Annals of Operations Research*, 152(1): 341–365, 2007.

[64] David B. Brown and James E. Smith. Dynamic portfolio optimization with transaction costs: Heuristics and dual bounds. *Management Science*, 57(10):1752–1770, 2011.

[65] Evan Hurwitz and Tshilidzi Marwala. State of the art review for applying computational intelligence and machine learning techniques to portfolio optimisation. *ACM Computing Research Repository (CoRR)*, arXiv:0910.2276, 2009.

[66] Stanislav Uryasev. Conditional value-at-risk: Optimization algorithms and applications. In *Proceedings of the IEEE/IAFE/INFORMS 2000 Conference on Computational Intelligence for Financial Engineering (CIFEr)*, pages 49–57. IEEE, 2000.

[67] Giuseppe Carlo Calafiore and Fatemeh Kharaman. Multi-period asset allocation with lower partial moments criteria and affine policies. In *IEEE Conference on Computational Intelligence for Financial Engineering & Economics (CIFEr)*, pages 100–106, 2014.

[68] Dan diBartolomeo. Smarter rebalancing: Using single period optimization in a multi-period world. Research publications, Northfield Information Services, 2012. URL http://www.northinfo.com/Documents/500.pdf.

[69] John M. Mulvey, William R. Pauling, and Ronald E. Madey. Advantages of Multiperiod Portfolio Models. *The Journal of Portfolio Management*, 29(2):35–45, 2003.

[70] Gerd Infanger. Dynamic asset allocation strategies using a stochastic dynamic programming approach. *Handbook of asset and liability management*, 1:199–251, 2006.

[71] Joëlle Skaf and Stephen Boyd. Multi-period portfolio optimization with constraint and transaction costs. Technical report, Electrical Engineering Department, Stanford University, April 2009. URL http://www.stanford.edu/~boyd/papers/pdf/dyn_port_opt.pdf.

[72] K.P. Anagnostopoulos and G. Mamanis. The mean-variance cardinality constrained portfolio optimization problem: An experimental evaluation of five multiobjective evolutionary algorithms. *Expert Systems with Applications*, 38(11):14208 – 14217, 2011.

[73] K. Metaxiotis and K. Liagkouras. Multiobjective evolutionary algorithms for portfolio management: A comprehensive literature review. *Expert Systems with Applications*, 39(14):11685–11698, 2012.

[74] Hanhong Hu, Yi Wang, Kesheng Wang, and Yun Chen. Particle swarm optimization (PSO) for the constrained portfolio optimization problem. *Expert Systems with Applications*, 38(8):10161–10169, August 2011.

[75] Zeinodin Alizadeh and Hossein Panahian. Portfolio optimization problems in different input data using particle swarm optimization. *International Journal of Computer Applications*, 50(2):23–26, July 2012.

[76] Raquel J. Fonseca, Steve Zymler, Wolfram Wiesemann, and Berc Rustem. Robust optimization of currency portfolios. *Journal of Computational Finance*, 15(1): 3–30, September 2011.

[77] Giuseppe Carlo Calafiore. Multi-period portfolio optimization with linear control policies. *Automatica*, 44(10):2463–2473, 2008.

[78] Ananth Madhavan and Seymour Smidt. An analysis of changes in specialist inventories and quotations. *The Journal of Finance*, 48(5):1595–1628, 1993.

[79] Richard K. Lyons. Profits and position control: A week of FX dealing. *Journal of International Money and Finance*, 17(1):97–115, 1998.

[80] David Tien. Hedging demand and foreign exchange risk premia. Technical report, Haas School of Business, UC Berkeley, 2002. URL https://www.bis.org/cgfs/conf/mar02k.pdf.

[81] John N. Kallianiotis. Foreign exchange rate determination. In *Exchange Rates and International Financial Economics: History, Theories, and Practices*, chapter 3. Palgrave Macmillan, Basingstoke, 2013.

[82] Yu-Lun Chen and Yin-Feng Gau. News announcements and price discovery in foreign exchange spot and futures markets. *Journal of Banking & Finance*, 34(7): 1628–1636, 2010.

[83] Gordon Johnson and Thomas Schneeweis. Jump-diffusion processes in the foreign exchange markets and the release of macroeconomic news. *Computational Economics*, 7(4):309–329, 1994.

[84] Kevin P. Evans. Intraday jumps and US macroeconomic news announcements. *Journal of Banking & Finance*, 35(10):2511–2527, 2011.

[85] Martin D.D. Evans and Richard K. Lyons. How is macro news transmitted to exchange rates? *Journal of Financial Economics*, 88(1):26–50, 2008.

[86] Tomáš Bunčák. Jump processes in exchange rates modeling. Working paper, Masaryk Institute of Advanced Studies. Czech Technical University in Prague, 2013. URL http://mpra.ub.uni-muenchen.de/51350/1/MPRA_paper_51350.pdf.

[87] Frank McGroartya, Owain ap Gwilymb, and Stephen Thomasc. The role of private information in return volatility, bid–ask spreads and price levels in the foreign exchange market. *Journal of International Financial Markets, Institutions and Money*, 19(2):387–401, 2009.

[88] Martin Scholtus, Dick van Dijk, and Bart Frijns. Speed, algorithmic trading, and market quality around macroeconomic news announcements. *Journal of Banking & Finance*, 38:89–105, 2014.

[89] Roger D. Huang and Hans R. Stoll. The components of the bid-ask spread: A general approach. *Review of Financial Studies*, 10(4):995–1034, 1997.

[90] Dan diBartolomeo and Howard Hoffman. *A Market Impact Model that Works*. Northfield Information Services, Inc., 2007. URL http://www.northinfo.com/documents/279.pdf.

[91] Andrew Ferraris. *Equity Market Impact Models*. Deutsche Bank AG, 2008. URL http://www.dbquant.com/Presentations/Berlin200812.pdf.

[92] Somayeh Moazeni, Thomas F. Coleman, and Yuying Li. Optimal portfolio execution strategies and sensitivity to price impact parameters. *SIAM Journal on Optimization*, 20(3):1620–1654, 2010.

[93] Tarun Ramadorai. What determines transaction costs in foreign exchange markets? *International Journal of Finance & Economics*, 13(1):14–25, 2008.

[94] W. Wang, X. Wei, D. Choi, X. Lu, G. Yang, and C. Sun. Electrochemical cells for medium- and large-scale energy storage: Fundamentals. In Chris Lim, Maria Menictas, and Mariana Skyllas-KazacosTuti, editors, *Advances in Batteries for Medium and Large-Scale Energy Storage*, chapter 1, pages 3–28. Woodhead Publishing, 2015.

[95] Haisheng Chen, Thang Ngoc Cong, Wei Yang, Chunqing Tan, Yongliang Li, and Yulong Ding. Progress in electrical energy storage system: A critical review. *Progress in Natural Science*, 19(3):291–312, 2009.

[96] K.C. Divya and Jacob Østergaard. Battery energy storage technology for power systems – an overview. *Electric Power Systems Research*, 79(4):511–520, 2009.

[97] John P. Barton and David G. Infield. Energy storage and its use with intermittent renewable energy. *IEEE Transactions on Energy Conversion*, 19(2):441–448, 2004.

[98] Craig Froome and Paul Meredith. Review of storage options for grid connected PV within Australia. In *In proceedings of the 48th AuSES Annual Conference (Solar 2010)*, Canberra, ACT, Australia, December 2010. Australian Solar Energy Society.

[99] Rajab Khalilpour and Anthony Vassallo. Leaving the grid: An ambition or a real choice? *Energy Policy*, 82:207–221, 2015.

[100] Hao Liang, Amit Kumar Tamang, Weihua Zhuang, and Xuemin Sherman Shen. Stochastic information management in smart grid. *IEEE Communications Surveys Tutorials*, 16(3):1746–1770, 2014.

[101] P. Arun, Rangan Banerjee, and Santanu Bandyopadhyay. Optimum sizing of photovoltaic battery systems incorporating uncertainty through design space approach. *Solar Energy*, 83(7):1013–1025, 2009.

[102] Yu Ru, Jan Kleissl, and Sonia Martinez. Battery sizing for grid connected PV systems with fixed minimum charging/discharging time. In *American Control Conference (ACC)*, pages 270–275. IEEE, 2012.

[103] Ru Yu, Jan Kleissl, and Sonia Martinez. Storage size determination for grid-connected photovoltaic systems. *IEEE Transactions on Sustainable Energy*, 4(1): 68–81, January 2013.

[104] Zhenpo Wang and Shuo Wang. Grid power peak shaving and valley filling using vehicle-to-grid systems. *IEEE Transactions on Power Delivery*, 28(3):1822–1829, 2013.

[105] Amir-Hamed Mohsenian-Rad and Alberto Leon-Garcia. Optimal residential load control with price prediction in real-time electricity pricing environments. *IEEE Transactions on Smart Grid*, 1(2):120–133, 2010.

[106] Pengwei Du and Ning Lu. Appliance commitment for household load scheduling. *IEEE Transactions on Smart Grid*, 2(2):411–419, 2011.

[107] Bo Lu and Mohammad Shahidehpour. Short-term scheduling of battery in a grid-connected PV/battery system. *IEEE Transactions on Power Systems*, 20(2): 1053–1061, 2005.

[108] Nikolaos Gatsis and Georgios B. Giannakis. Residential load control: Distributed scheduling and convergence with lost AMI messages. *IEEE Transactions on Smart Grid*, 3(2):770–786, 2012.

[109] Manisa Pipattanasomporn, Murat Kuzlu, and Saifur Rahman. An algorithm for intelligent home energy management and demand response analysis. *IEEE Transactions on Smart Grid*, 3(4):2166–2173, 2012.

[110] H.R. Teymour, D. Sutanto, K.M. Muttaqi, and P. Ciufo. Solar PV and battery storage integration using a new configuration of a three-level NPC inverter with advanced control strategy. *IEEE Transactions on Energy Conversion*, 29(2):354–365, June 2014.

[111] Yoash Levron and Doron Shmilovitz. Power systems' optimal peak-shaving applying secondary storage. *Electric Power Systems Research*, 89:80–84, 2012.

[112] A. Hooshmand, J. Mohammadpour, H. Malki, and H. Daneshi. Power system dynamic scheduling with high penetration of renewable sources. In *American Control Conference (ACC)*, pages 5827–5832, June 2013.

[113] M.K.C. Marwali, M.A. Haili, S.M. Shahidehpour, and K.H. Abdul-Rahman. Short term generation scheduling in photovoltaic-utility grid with battery storage. *IEEE Transactions on Power Systems*, 13(3):1057–1062, 1998.

[114] H.M.I. Pousinho, J. Contreras, P. Pinson, and V.M.F. Mendes. Robust optimisation for self-scheduling and bidding strategies of hybrid CSP–fossil power plants. *International Journal of Electrical Power & Energy Systems*, 67:639 – 650, 2015.

[115] Azadeh Kamjoo, Alireza Maheri, and Ghanim A. Putrus. Chance constrained programming using non-Gaussian joint distribution function in design of standalone hybrid renewable energy systems. *Energy*, 66:677–688, 2014.

[116] Michael Urbina and Zuyi Li. A fuzzy optimization approach to PV/battery scheduling with uncertainty in PV generation. In *Proceedings of 38th North American Power Symposium (NAPS)*, pages 561–566, 2006.

[117] Ruey Hsun Liang and Jian Hao Liao. A fuzzy-optimization approach for generation scheduling with wind and solar energy systems. *IEEE Transactions on Power Systems*, 22(4):1665–1674, November 2007.

[118] Hide Nishihara, Ittetsu Taniguchi, Shinya Kato, and Masahiro Fukui. A real-time power distribution based on load/generation forecasting for peak-shaving. In *IEEE 11th International New Circuits and Systems Conference (NEWCAS)*, pages 1–4, 2013.

[119] A. Nottrott, J. Kleissl, and B. Washom. Energy dispatch schedule optimization and cost benefit analysis for grid-connected, photovoltaic-battery storage systems. *Renewable Energy*, 55:230–240, 2013.

[120] Vojtech Veselỳ and Danica Rosinová. Robust model predictive control design. In Tao Zheng, editor, *Model Predictive Control*, chapter 1, pages 1–24. InTech, 2010. URL http://www.intechopen.com/books/model-predictive-control/robust-model-predictive-control-design.

[121] Joëlle Skaf, Stephen Boyd, and Assaf Zeevi. Shrinking-horizon dynamic programming. *International Journal of Robust and Nonlinear Control*, 20(17):1993–2002, 2010.

[122] Graham C. Goodwin, María M. Seron, and José A. De Doná. *Constrained Control and Estimation: An Optimisation Approach*. Springer, 2006.

[123] Frank L. Lewis, Draguna L. Vrabie, and Vassilis L. Syrmos. *Optimal Control*. John Wiley & Sons, Inc., 2012.

[124] S. Joe Qin and Thomas A. Badgwell. A survey of industrial model predictive control technology. *Control Engineering Practice*, 11(7):733–764, 2003.

[125] Przemyslaw Ignaciuk and Andrzej Bartoszewicz. Linear-quadratic optimal control of periodic-review perishable inventory systems. *IEEE Transactions on Control Systems Technology*, 20(5):1400–1407, 2012.

[126] V. Suresh and Dipak Chaudhuri. Dynamic scheduling - a survey of research. *International Journal of Production Economics*, 32(1):53–63, 1993.

[127] Giuseppe Carlo Calafiore and Lorenzo Fagiano. Stochastic model predictive control of LPV systems via scenario optimization. *Automatica*, 49(6):1861–1866, 2013.

[128] Giuseppe Carlo Calafiore. Random convex programs. *SIAM Journal on Optimization*, 20(6):3427–3464, 2010.

[129] Georg Schildbach, Giuseppe Carlo Calafiore, Lorenzo Fagiano, and Manfred Morari. Randomized model predictive control for stochastic linear systems. In *American Control Conference (ACC)*, pages 417–422, 2012.

[130] Georg Schildbach, Lorenzo Fagiano, Christoph Frei, and Manfred Morari. The scenario approach for stochastic model predictive control with bounds on closed-loop constraint violations. *Automatica*, 50(12):3009–3018, 2014.

[131] Daniele Bernardini and Alberto Bemporad. Stabilizing model predictive control of stochastic constrained linear systems. *IEEE Transactions on Automatic Control*, 57(6):1468–1480, 2012.

[132] Marco C. Campi and Simone Garatti. The exact feasibility of randomized solutions of uncertain convex programs. *SIAM Journal on Optimization*, 19(3):1211–1230, 2008.

[133] Georg Schildbach, Lorenzo Fagiano, and Manfred Morari. Randomized solutions to convex programs with multiple chance constraints. *SIAM Journal on Optimization*, 23(4):2479–2501, 2013.

[134] Giuseppe C. Calafiore and Lorenzo Fagiano. Robust model predictive control via scenario optimization. *IEEE Transactions on Automatic Control*, 58(1):219–224, 2013.

[135] Werner Römisch. Scenario generation. In *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, Inc., 2011.

[136] Paul Glasserman. *Monte Carlo Methods in Financial Engineering*, volume 53 of *Stochastic Modelling and Applied Probability*. Springer, 2003.

[137] Carlos Jabbour, J. Peña, J. Vera, and L. Zuluaga. An estimation-free, robust CVaR portfolio allocation model. *Journal of Risk*, 11(1):1–22, 2007.

[138] Manfred Gilli, Evis Këllezi, and Hilda Hysi. A data-driven optimization heuristic for downside risk minimization. *Journal of Risk*, 8(3):1–19, 2006.

[139] Florian Herzog, Simon Keel, Gabriel Dondi, L.M. Schumann, and Hans P. Geering. Model predictive control for portfolio selection. In *American Control Conference (ACC)*, pages 1252–1259, 2006.

[140] Nikolas Topaloglou, Hercules Vladimirou, and Stavros A. Zenios. A dynamic stochastic programming model for international portfolio management. *European Journal of Operational Research*, 185(3):1501–1524, 2008.

[141] James A. Primbs. Portfolio optimization applications of stochastic receding horizon control. In *American Control Conference (ACC)*, pages 1811–1816, 2007.

[142] Giuseppe Carlo Calafiore. An affine control method for optimal dynamic asset allocation with transaction costs. *SIAM Journal on Control and Optimization*, 48 (4):2254–2274, 2009.

[143] Jacek Gondzio, Roy Kouwenberg, and Ton Vorst. Hedging options under transaction costs and stochastic volatility. *Journal of Economic Dynamics and Control*, 27(6):1045–1068, 2003.

[144] James A. Primbs. LQR and receding horizon approaches to multi-dimensional option hedging under transaction costs. In *American Control Conference (ACC)*, pages 6891–6896, 2010.

[145] Peter J. Meindl and James A. Primbs. Dynamic hedging of single and multi-dimensional options with transaction costs: A generalized utility maximization approach. *Quantitative Finance*, 8(3):299–312, 2008.

[146] Alberto Bemporad, Tommaso Gabbriellini, Laura Puglia, and Leonardo Bellucci. Scenario-based stochastic model predictive control for dynamic option hedging. In *IEEE Conference on Decision and Control (CDC)*, pages 6089–6094, 2010.

[147] Alberto Bemporad, Laura Puglia, and Tommaso Gabbriellini. A stochastic model predictive control approach to dynamic option hedging with transaction costs. In *American Control Conference (ACC)*, pages 3862–3867, 2011.

[148] Wencong Su, Jianhui Wang, and Jaehyung Roh. Stochastic energy scheduling in microgrids with intermittent renewable energy resources. *IEEE Transactions on Smart Grid*, 5(4):1876–1883, July 2014.

[149] Emilio Perez, Hector Beltran, Néstor Aparicio, and Pedro Rodriguez. Predictive power control for PV plants with energy storage. *IEEE Transactions on Sustainable Energy*, 4(2):482–490, 2013.

[150] Keyu Long and Zaiyue Yang. Model predictive control for household energy management based on individual habit. In *Proceedings of 25th Chinese Control and Decision Conference (CCDC)*, pages 3676–3681, May 2013.

[151] Matthew P. Johnson, Amotz Bar-Noy, Ou Liu, and Yi Feng. Energy peak shaving with local storage. *Sustainable Computing: Informatics and Systems*, 1(3):177–188, 2011.

[152] Cara R. Touretzky and Michael Baldea. Integrating scheduling and control for economic MPC of buildings with energy storage. *Journal of Process Control*, 24 (8):1292–1300, 2014.

[153] Miles Lubin, Cosmin G. Petra, Mihai Anitescu, and Victor Zavala. Scalable stochastic optimization of complex energy systems. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–10, 2011.

[154] Dinghuan Zhu and Gabriela Hug. Decomposed stochastic model predictive control for optimal dispatch of storage and generation. *IEEE Transactions on Smart Grid*, 5(4):2044–2053, July 2014.

[155] Michael O'Neill and Conor Ryan. Grammatical evolution. *IEEE Transactions on Evolutionary Computation*, 5(4):349–358, 2001.

[156] Donald E. Knuth. Backus Normal Form vs. Backus–Naur Form. *Communications of the ACM*, 7(12):735–736, December 1964.

[157] John H. Holland. *Adaptation in Natural and Artificial Systems.* MIT Press, Cambridge, MA, USA, 1992.

[158] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, volume 1. MIT press, 1992.

[159] Robert I. McKay, Nguyen Xuan Hoai, Peter Alexander Whigham, Yin Shan, and Michael O'Neill. Grammar-based genetic programming: A survey. *Genetic Programming and Evolvable Machines*, 11:365–396, 2010.

[160] Melanie Mitchell. *An Introduction to Genetic Algorithms.* MIT Press, Cambridge, MA, USA, 1996.

[161] John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis With Applications to Biology, Control, and Artificial Intelligence.* The University of Michigan Press, Ann Arbor, 1975.

[162] Mandavilli Srinivas and Lalit M. Patnaik. Genetic algorithms: A survey. *Computer*, 27(6):17–26, June 1994.

[163] Michael Sipser. Context-free grammars. In *Introduction to the Theory of Computation*, chapter 2, pages 91–122. PWS Publishing Company, 1997.

[164] Milan Borkovec, Ian Domowitz, and Christopher Escobar. How big is "Big" - some evidence from aggregate trading costs in the FX market. Report, Investment Technology Group, Inc., 2013. URL http://www.itg.com/marketing/ITG_WP_FX_BorkovecDomowitzEscobar_20130912.pdf.

[165] Angelo Ranaldo. Segmentation and time-of-day patterns in foreign exchange markets. *Journal of Banking & Finance*, 33(12):2199–2206, 2009.

[166] Peter J. Brockwell and Richard A. Davis. *Introduction to Time Series and Forecasting*. Springer Texts in Statistics. Springer, New York, Berlin, Heidelberg, 2002.

[167] Anthony Mihirana de Silva and Philip H.W. Leong. *Grammar-Based Feature Generation for Time-Series Prediction*. Springer, 2015.

[168] Lei Li, Farzad Noorian, Duncan J.M. Moss, and Philip H.W. Leong. Rolling window time series prediction using MapReduce. In *IEEE International Conference on Information Reuse and Integration (IRI)*, pages 757–764. IEEE, 2014.

[169] Harris Drucker, Chris J.C. Burges, Linda Kaufman, Alex Smola, Vladimir Vapnik, et al. Support vector regression machines. *Advances in Neural Information Processing Systems*, 9:155–161, 1997.

[170] Yaakov Engel, Shie Mannor, and Ron Meir. The kernel recursive least-squares algorithm. *Signal Processing, IEEE Transactions on*, 52(8):2275–2285, 2004.

[171] George E.P. Box and David R. Cox. An analysis of transformations. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 211–252, 1964.

[172] Robert B. Cleveland, William S. Cleveland, Jean E. McRae, and Irma Terpenning. Stl: A seasonal-trend decomposition procedure based on loess. *Journal of Official Statistics*, 6(1):3–73, 1990.

[173] Denis Kwiatkowski, Peter C.B. Phillips, Peter Schmidt, and Yongcheol Shin. Testing the null hypothesis of stationarity against the alternative of a unit root: How sure are we that economic time series have a unit root? *Journal of econometrics*, 54(1):159–178, 1992.

[174] 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(366a):427–431, 1979.

[175] Peter C.B. Philips and Pierre Perron. Testing for a unit root in time series regression. *Biometrika*, 75(2):335–346, 1988.

[176] Fabio Canova and Bruce E. Hansen. Are seasonal patterns constant over time? a test for seasonal stability. *Journal of Business & Economic Statistics*, 13(3): 237–252, 1995.

[177] Denise R. Osborn, Alice P.L. Chui, Jeremy P. Smith, and Chris R. Birchenhall. Seasonality and the order of integration for consumption. *Oxford Bulletin of Economics and Statistics*, 50(4):361–377, 1988.

[178] Fang-Mei Tseng, Hsiao-Cheng Yu, and Gwo-Hsiung Tzeng. Combining neural network model with seasonal time series ARIMA model. *Technological Forecasting and Social Change*, 69(1):71–87, 2002.

[179] Tucker McElroy and Scott Holan. Using spectral peaks to detect seasonality. In *Proceedings of the Federal Conference on Statistical Methodology*, 2009.

[180] Robert W. Colby and Thomas A. Meyers. *The Encyclopedia of Technical Market Indicators*. Irwin New York, 1988.

[181] Martin Pring. *Study Guide for Technical Analysis Explained: The Successful Investor's Guide to Spotting Investment Trends and Turning Points*. McGraw-Hill, 2002.

[182] Michael J. Artis, José G. Clavel, Mathias Hoffmann, and Dilip Madhukar Nachane. Harmonic regression models: A comparative review with applications. Working paper no. 333, Institute for Empirical Research in Economics, University of Zurich, 2007. URL http://dx.doi.org/10.2139/ssrn.1017519.

[183] Richard Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton University Press Princeton, New Jersey, 1961.

[184] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[185] Emre Soyer and Robin M. Hogarth. The illusion of predictability: How regression statistics mislead experts. *International Journal of Forecasting*, 28(3):695–711, 2012.

[186] J. Scott Armstrong. Illusions in regression analysis. *International Journal of Forecasting*, 28:689–694, 2012.

[187] Huan Liu and Lei Yu. Toward integrating feature selection algorithms for classification and clustering. *Knowledge and Data Engineering, IEEE Transactions on*, 17(4):491–502, 2005.

[188] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[189] Ray J. Frank, Neil Davey, and Stephen P. Hunt. Time series prediction and neural networks. *Journal of Intelligent and Robotic Systems*, 31(1-3):91–103, 2001.

[190] Yoshua Bengio, Ian J. Goodfellow, and Aaron Courville. Deep learning. Book in preparation for MIT Press, 2015. URL http://www.iro.umontreal.ca/~bengioy/dlbook.

[191] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Extreme learning machine: Theory and applications. *Neurocomputing*, 70(1):489–501, 2006.

[192] Thomas Hofmann, Bernhard Schölkopf, and Alexander J. Smola. Kernel methods in machine learning. *The Annals of Statistics*, 36(3):1171–1220, 2008.

[193] Douglas Reynolds. Gaussian mixture models. In *Encyclopedia of Biometrics*, pages 659–663. Springer, 2009.

[194] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[195] Souhaib Ben Taieb, Gianluca Bontempi, Amir F. Atiya, and Antti Sorjamaa. A review and comparison of strategies for multi-step ahead time series forecasting based on the NN5 forecasting competition. *Expert Systems with Applications*, 39 (8):7067–7083, 2012.

[196] Massimiliano Marcellino, James H. Stock, and Mark W. Watson. A comparison of direct and iterated multistep AR methods for forecasting macroeconomic time series. *Journal of Econometrics*, 135(1):499–526, 2006.

[197] Souhaib Ben Taieb and Rob J. Hyndman. Recursive and direct multi-step forecasting: The best of both worlds. Working paper, Monash University, Department of Econometrics and Business Statistics, 2012. URL http://robjhyndman.com/working-papers/rectify/.

[198] Souhaib Ben Taieb and Rob J. Hyndman. Boosting multi-step autoregressive forecasts. In *Proceedings of the 31st International Conference on Machine Learning*, pages 109–117, 2014.

[199] Rob J. Hyndman and Anne B. Koehler. Another look at measures of forecast accuracy. *International Journal of Forecasting*, 22(4):679–688, 2006.

[200] Rob J. Hyndman and Yeasmin Khandakar. Automatic time series forecasting: The forecast package for R. *Journal of Statistical Software*, 27:1–22, 2008.

[201] Rob J Hyndman. *forecast: Forecasting Functions for Time Series and Linear Models*, 2015. URL http://github.com/robjhyndman/forecast. R package version 6.1.

[202] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2015. URL http://www.R-project.org/.

[203] Antonio, Fabio Di Narzo, Jose Luis Aznarte, and Matthieu Stigler. *tsDyn: Time Series Analysis Based on Dynamical Systems Theory*, 2015. URL http://stat.ethz.ch/CRAN/web/packages/tsDyn/vignettes/tsDyn.pdf. R package version 0.9-43.

[204] Max Kuhn. Contributions from Jed Wing, Steve Weston, Andre Williams, Chris Keefer, Allan Engelhardt, Tony Cooper, Zachary Mayer, Brenton Kenkel, the R Core Team, Michael Benesty, Reynald Lescarbeau, Andrew Ziem, and Luca Scrucca. *caret: Classification and Regression Training*, 2015. URL http://CRAN.R-project.org/package=caret. R package version 6.0-47.

[205] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: An update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

[206] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[207] Rob J. Hyndman, Anne B. Koehler, Ralph D. Snyder, and Simone Grose. A state space framework for automatic forecasting using exponential smoothing methods. *International Journal of Forecasting*, 18(3):439–454, 2002.

[208] Rob J. Hyndman, Muhammad Akram, and Blyth C. Archibald. The admissible parameter space for exponential smoothing models. *Annals of the Institute of Statistical Mathematics*, 60(2):407–426, 2008.

[209] Jan G. De Gooijer and Rob J. Hyndman. 25 years of time series forecasting. *International Journal of Forecasting*, 22(3):443–473, 2006.

[210] Victor M. Guerrero. Time-series analysis supported by power transformations. *Journal of Forecasting*, 12(1):37–48, 1993.

[211] Osman Dag, Ozgur Asar, and Ozlem Ilk. *AID: Estimation of Box-Cox Power Transformation Parameter*, 2015. URL http://CRAN.R-project.org/package=AID. R package version 1.5.

[212] A. Senthil Kumar and Zainal Ahmad. Model predictive control (MPC) and its current issues in chemical engineering. *Chemical Engineering Communications*, 199(4):472–511, 2012.

[213] Hee Sang Ko, Juri Jatskevich, Guy Dumont, and Gi Gap Yoon. An advanced LMI-based-LQR design for voltage control of grid-connected wind farm. *Electric Power Systems Research*, 78(4):539–546, 2008.

[214] C. Bordons and J.R. Cueli. Predictive controller with estimation of measurable disturbances. Application to an olive oil mill. *Journal of Process Control*, 14(3): 305–315, 2004.

[215] Philip Doganis, Eleni Aggelogiannaki, and Haralambos Sarimveis. A combined model predictive control and time series forecasting framework for production-inventory systems. *International Journal of Production Research*, 46(24):6841–6853, 2008.

[216] Jacob Mattingley, Yang Wang, and Stephen Boyd. Receding horizon control: Automatic generation of high-speed solvers. *IEEE Control Systems*, 31(3):52–65, 2011.

[217] Krzysztof Patan. Neural network-based model predictive control: Fault tolerance and stability. *IEEE Transactions on Control Systems Technology*, 23(3):1147–1155, 2015.

[218] Minggang Liu, Yang Shi, and Fang Fang. Load forecasting and operation strategy design for CCHP systems using forecasted loads. *IEEE Transactions on Control Systems Technology*, 23(5):1672–1684, Sept 2015. ISSN 1063-6536.

[219] G. Peter Zhang and Min Qi. Neural network forecasting for seasonal and trend time series. *European Journal of Operational Research*, 160(2):501–514, 2005.

[220] Lijuan Cao. Support vector machines experts for time series forecasting. *Neurocomputing*, 51:321–339, 2003.

[221] Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936.

[222] Tan Miller. *Hierarchical Operations and Supply Chain Planning*. Springer Verlag, 2002.

[223] Jay D. Schwartz, Wenlin Wang, and Daniel E. Rivera. Simulation-based optimization of process control policies for inventory management in supply chains. *Automatica*, 42(8):1311–1320, 2006.

[224] Spyros Makridakis and Michele Hibon. The M3-competition: Results, conclusions and implications. *International journal of forecasting*, 16(4):451–476, 2000.

[225] Torben G. Andersen, Tim Bollerslev, Francis X. Diebold, and Paul Labys. The distribution of realized exchange rate volatility. *Journal of the American Statistical Association*, 96(453):42–55, 2001.

[226] Torben G. Andersen, Tim Bollerslev, and Francis X. Diebold. Roughing it up: Including jump components in the measurement, modeling, and forecasting of return volatility. *The Review of Economics and Statistics*, 89(4):701–720, 2007.

[227] Mario Cerrato, Nicholas Sarantis, and Alex Saunders. An investigation of customer order flow in the foreign exchange market. *Journal of Banking & Finance*, 35(8): 1892–1906, 2011.

[228] Shikuan Chen, Chih-Chung Chien, and Ming-Jen Chang. Order flow, bid-ask spread and trading density in foreign exchange markets. *Journal of Banking & Finance*, 36(2):597–612, 2012.

[229] Luca Di Gaspero and Eric Moyer. QuadProg++: A C++ porting of the Goldfarb–Idnani dual method for solving quadratic programming problems, 2010. URL http://www.diegm.uniud.it/digaspero/index.php/software.

[230] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3, 2010. URL http://eigen.tuxfamily.org.

[231] DailyFX Market News and Analysis. Terms and conditions, 2015. URL http://www.dailyfx.com/page/terms_and_conditions.html.

[232] DailyFX Market News and Analysis. Forex economic calendar, 2014. URL http://www.dailyfx.com/calendar.

[233] Ruey S. Tsay. *Analysis of Financial Time Series*. John Wiley & Sons, Hoboken, New Jersey, 2005.

[234] Ingo Steinwart and Andreas Christmann. *Support Vector Machines*. Springer Science & Business Media, 2008.

[235] Randolf F.M. Weterings. Battery storage systems. Master of science in management of technology thesis, Delft University of Technology, 2010.

[236] Feng Cheng, Steve Willard, Jonathan Hawkins, Brian Arellano, Olga Lavrova, and Andrea Mammoli. Applying battery energy storage to enhance the benefits of photovoltaics. In *IEEE Energytech*, pages 1–5, 2012.

[237] Alexandre Oudalov, Rachid Cherkaoui, and Antoine Beguin. Sizing and optimal operation of battery energy storage system for peak shaving application. In *IEEE Power Tech*, pages 621–625, Lausanne, 2007.

[238] TransGrid. iDemand, 2015. URL http://idemand.transgrid.com.au/.

[239] TransGrid. *iDemand: A technical guide to TransGrid's Western Sydney electricity demand management system*, 2014.

[240] Brendan Galloway and Gerhard P. Hancke. Introduction to industrial control networks. *IEEE Communications Surveys & Tutorials*, 15(2):860–880, 2013.

[241] *Terms & Conditions*. TransGrid, 2015. URL https://www.transgrid.com.au/Pages/Terms.aspx.

[242] *Term of Conditions and Use*. Weather Underground, LLC, 2015. URL http://www.wunderground.com/members/tos.asp.

[243] Weather Underground. Weather forecast & reports - Long range & local, 2015. URL http://www.wunderground.com.

[244] Rob J. Hyndman and Shu Fan. *Monash Electricity Forecasting Model*, 2015. URL http://robjhyndman.com/papers/MEFMR1.pdf.

[245] Martina Friese, Oliver Flasch, Katya Vladislavleva, Thomas Bartz-Beielstein, Olaf Mersmann, Boris Naujoks, Jörg Stork, and Martin Zaefferer. Ensemble-based model selection for smart metering data. *Proceedings of 22 Workshop on Computational Intelligence*, 22:215, 2014.

[246] Patrick E. McSharry, Sonja Bouwman, and Gabriël Bloemhof. Probabilistic forecasts of the magnitude and timing of peak electricity demand. *IEEE Transactions on Power Systems*, 20(2):1166–1172, 2005.

[247] Souhaib Ben Taieb and Rob J. Hyndman. A gradient boosting approach to the Kaggle load forecasting competition. *International Journal of Forecasting*, 30(2): 382–394, 2014.

[248] Origin Energy Retail Limited. NSW residential energy price fact sheet, 2015. URL http://www.originenergy.com.au/content/dam/origin/residential/docs/energy-price-fact-sheets/nsw/NSW_Electricity_Residential_AusGrid_eSaver.PDF.

[249] Michael O'Neill, Erik Hemberg, Conor Gilligan, Eliott Bartley, James McDermott, and Anthony Brabazon. *GEVA*: Grammatical evolution in Java. *ACM SIGEVOlution*, 3(2):17–22, 2008. URL http://ncra.ucd.ie/Site/GEVA.html.

[250] Erik Hemberg and James McDermott. PonyGE*: A Pony-sized Implementation of Grammatical Evolution in Python*, 2012. URL http://ncra.ucd.ie/Site/GEVA.html. Version 0.1.5.

[251] Don Smiley. PyNeurGen*: Python Neural Genetic Algorithm Hybrids*, 2012. URL http://pyneurgen.sourceforge.net/. Release 0.3.

[252] Erik Hemberg. *Grammatical Evolution in MATLAB (*GEM*)*, 2011. URL http://ncra.ucd.ie/GEM/. Version 0.2.

[253] Pavel Suchmann. *Grammatical Evolution Ruby Exploratory Toolkit (*GERET*)*, 2013. URL http://geret.org/.

[254] R Team. *The Comprehensive R Archive Network*, 2015. URL http://cran.r-project.org/.

[255] Miguel Nicolau. libGE *C++ Library*, 2006. URL http://bds.ul.ie/libGE/. Stable Release 0.26.

[256] Adam Nohejl. AGE*: Algorithms for Grammar-based Evolution*, 2011. URL http://nohejl.name/age/. Version 1.1.1.

[257] Dirk Eddelbuettel and Romain Francois. *Rcpp*: Seamless R and C++ integration. *Journal of Statistical Software*, 40(8):1–18, 4 2011.

[258] Egon Willighagen. genalg*: R Based Genetic Algorithm*, 2014. URL http://CRAN.R-project.org/package=genalg. R package version 0.1.1.1.

[259] Kalyanmoy Deb and Samir Agrawal. Understanding interactions among genetic algorithm parameters. *Foundations of Genetic Algorithms*, pages 265–286, 1999.

[260] Michael O'Neill, Conor Ryan, Maarten Keijzer, and Mike Cattolico. Crossover in grammatical evolution. *Genetic Programming and Evolvable Machines*, 4(1): 67–93, 2003.

[261] Kevin Ushey and Jim Hester. rex*: Friendly Regular Expressions*, 2014. URL https://github.com/kevinushey/rex. R package version 0.2.0.99.

[262] David Meyer, Evgenia Dimitriadou, Kurt Hornik, Andreas Weingessel, and Friedrich Leisch. e1071*: Misc Functions of the Department of Statistics (e1071), TU Wien*, 2014. URL http://CRAN.R-project.org/package=e1071. R package version 1.6-4.

[263] Hong Guo, Lindsay B. Jack, and Asoke K. Nandi. Feature generation using genetic programming with application to fault classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 35(1):89–99, 2005.

[264] Maarten Keijzer. Scaled symbolic regression. *Genetic Programming and Evolvable Machines*, 5(3):259–269, 2004.

[265] Dan Costelloe and Conor Ryan. On improving generalisation in genetic programming. In *Genetic Programming*, volume 5481 of *Lecture Notes in Computer Science*, pages 61–72. Springer-Verlag, Berlin, Heidelberg, 2009.

[266] Amazon Web Services, Inc. AWS: Amazon Elastic Compute Cloud EC2, 2015. URL http://aws.amazon.com/ec2/.

[267] Microsoft. Microsoft Azure: Cloud Computing Platform and Services, 2015. URL https://azure.microsoft.com/en-us/.

[268] Google. Google Cloud Platform, 2015. URL https://cloud.google.com/.