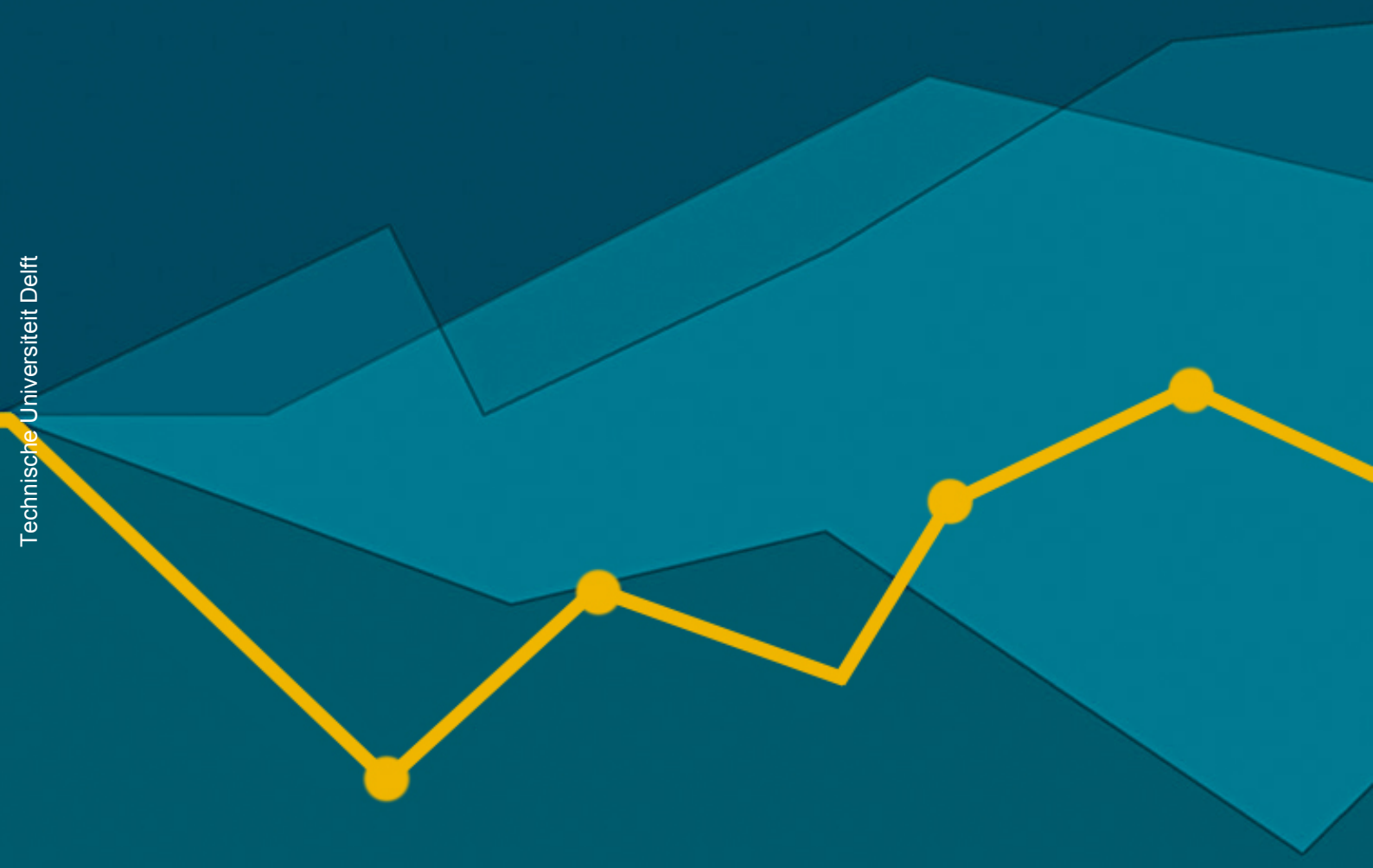
Gaussian Process Regression for Long-Term Time Series Forecasting

Bagas Abisena Swastanto



Technische Universiteit Delft

TU Delft

Gaussian Process Regression for Long-Term Time Series Forecasting

by

Bagas Abisena Swastanto

in partial fulfillment of the requirements for the degree of

Master of Science
in Computer Science

at the Faculty of Electrical Engineering, Mathematics, and Computer Science,
Delft University of Technology,

to be defended publicly on Thursday October 20, 2016 at 14:00.

Student number:	4415345	
Project duration:	January 11, 2016 – October 20, 2016	
Supervisor:	Dr. D. M. J. Tax	TU Delft
Thesis committee:	Prof. dr. ir. M. J. T. Reinders,	TU Delft
	Dr. J. C. van Gemert,	TU Delft
	Dr. ir. Huijuan Wang	TU Delft

An electronic version of this thesis is available at http://repository.tudelft.nl/.

Illustration on the cover is a modified image from http://professor-excel.com.

TU Delft

Preface

This is a master thesis project that is conducted in the Pattern Recognition Lab, Faculty of Electrical Engineering, Mathematics, and Computer Science, Delft University of Technology. This topic is chosen due to my keen interest in time series forecasting problem. Many people have supported me throughout this thesis journey. So here I would like to express my gratitude to all of them.

First of all, I would like to thank my supervisor Dr. David Tax. I really appreciate your guidance, not only about the thesis but also how to make a good scientific work. More importantly, thanks for the constant 'slap in the face' when I become sloppy in writing. It has been an honour to work with you. Also I would like to thank the committee members Prof. M.J.T. Reinders, and Dr. J.C. van Gemert, and Dr. Huijian Wang for evaluating my thesis work.

To my lovely wife Adiska, thank you for, well, everything. Without your support, I wouldn't have made it. You are truly my source of inspiration! For family and friends, whom I can't name one by one, your support means a lot. Thank you, dank je well, and terima kasih. Lastly, I want to thank the Indonesian government for the scholarship. I hope my study will benefit the country soon.

Bagas Abisena Swastanto
Delft, October 2016

Abstract

Long-term time series forecasting has found many utilities in various domains. Nevertheless, it remains difficult to perform by many existing methods. One of the most well-known forecasting techniques, the ARIMA, does not suffice the long-term forecasting task due to the mean convergence problem. Therefore, this research empirically assesses the alternative solution based on the Gaussian process (GP) regression. This study presents two approaches of Gaussian process regression for our problem: the structure modelling and the autoregressive approach. These techniques are evaluated on two synthetic datasets and two real-world datasets, which are the wind speed and electricity consumption dataset. From the experiment, it can be concluded that the GP-based forecasting techniques show more favourable long-term forecasting performance than the ARIMA model, particularly in cases where the data contain apparent trend and season. The experiment also demonstrates that the structure modelling method slightly outperforms the autoregressive approach for long-term forecast and offers a benefit that the autoregressive model does not have, which is the interpretability of the model.

Contents

1 Introduction

1.1. Background

Time series is a collection of data points that is ordered in time. There are many examples of time series data, ranging from naturally generated data such as weather or amount of greenhouse gas in the atmosphere to human-made data such as stock market and sensors recording. Due to this ubiquity, studies on time series have become an important field of research over the last few decades. These studies have found their place in many facets of science, such as social science, engineering, medicine, economy, and finance.

Most of the study of the time series come in the form of time series modelling, which is the process of understanding and defining (mathematically) the inherent structure of the time series [1, 18]. Time series modelling is very useful in several ways, from describing the feature that makes up the time series, to forecasting the future [6]. The latter part is particularly interesting, because once we have comprehended the model of a time series data, we can simulate the generation of data to an unforeseen future. The ability to forecast the future which often referred as *time series forecasting* is one of the most studied applications in time series modelling [17]. This fact is expected since time series forecasting has found many utilities across many applications.

A typical time series forecasting setting is shown in Figure 1.1. In the unshaded part of the graph, we have our time series observations. The shaded region signifies the forecasting *horizons* which are points in the future that we want to forecast. We often define the forecasting horizon in the h-step-ahead term. For example, if we want to forecast a point that is a one-time step ahead in the future, we denote it as *one*-step-ahead forecast. Similarly, a three-time step forecast is called the *three*-step-ahead forecast. Another practice that is common in time series forecasting research is to classify the forecasting horizons into distinct categories: short-term, medium-term, and long-term. Short-term denotes horizons that are closed to the observations whereas long-term is horizons that are far ahead of the last observed data. What is not clear, however, is how the boundary between each category is determined, since it is data and application specific.

The ability to predict short-term and long-term forecasts are as equally important in many applications. Hence, it is vital for a time series forecasting technique to be able to excel in all horizons. Long-term forecasting is a more challenging problem to solve than the short-term forecast due to error accumulation, the lack of information, and the growing uncertainties [6]. The effect of these challenges is more apparent as the forecasting horizon grows. Thus, one of the primary criteria of a good time series forecasting technique the capability of forecasting the future accurately while keeping the error minimum as the forecasting horizon lengthens.

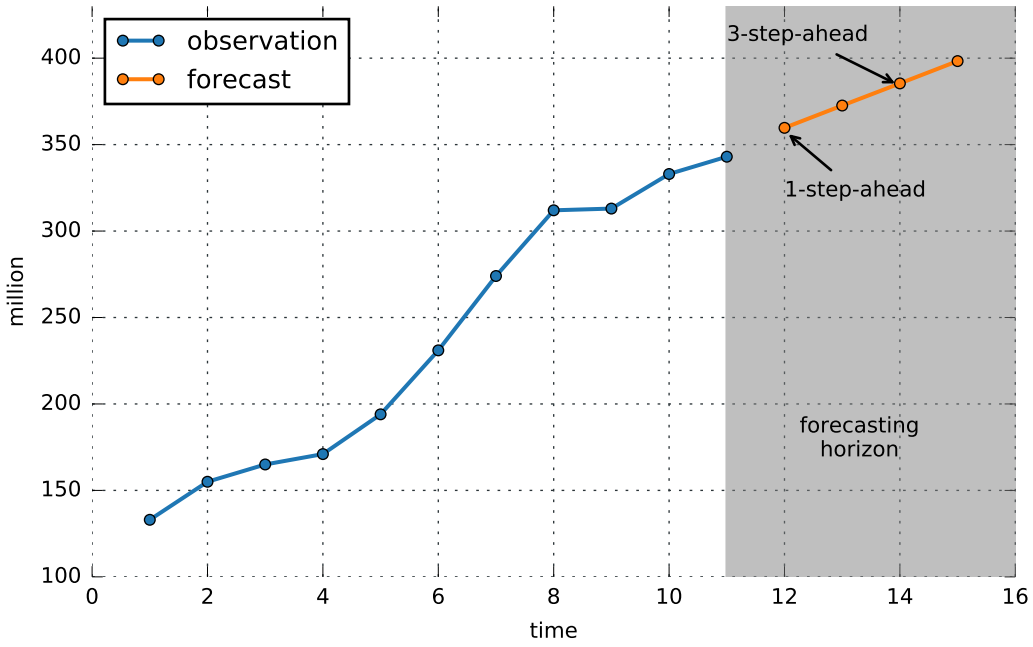


Figure 1.1: A diagram that shows a typical time series forecasting setting.

1.2. Time Series Forecasting Use Case: Wind Energy Forecasting

One of the prominent use cases of the time series forecasting is in the wind energy forecast. Wind power sees an increasing utilisation throughout the world. Amidst its fast growth, wind energy generation always suffers from the intermittent availability of the wind itself. Wind energy strongly depends on the weather conditions which vary over time. Therefore, the amount of power that can be generated by the wind turbine always fluctuates [50]. To be able to integrate the wind energy reliably to the power grid, the wind energy forecasting becomes a vital tool to have. If we can accurately predict the availability of the wind energy in the distant future, the decision to increase or decrease the load of conventional energy source to power grid can be planned accordingly. An accurate forecast for a long-term horizon is as important as a short-term horizon, and each forecast has distinct benefits. Table 1.1 by Soman et al. highlights the classification of horizons as well as its use case in the wind energy domain [50].

Table 1.1: Classification of time horizon for wind power forecasting, according to Soman et al. [50]

Time Horizon	Range	Applications
Immediate	Few seconds to 30 minutes ahead	Electricity market clearing, regulation actions
Short-term	30 minutes to 6 hours ahead	Economic load dispatch planning, load increment/decrement decision
Medium-term	6 hours to 1 day ahead	Generator online/offline decisions, operational security in day-ahead electricity market
Long-term	1 day to 1 week or more ahead	Unit commitment decisions, reserve requirement decisions, maintenance scheduling to obtain optimal operating costs

Due to the importance of wind energy forecasting, many research initiatives have been pushed toward making a better prediction model of the wind power. There are two approaches to wind power forecast-

ing: the physical method and the statistical method. The former models the physics of the atmosphere while the latter utilises the statistical time series forecasting technique. Modelling the physics is a highly expensive computing process. As a result, the statistical method is more attractive due to its relatively low cost of computing yet still capable of generating a relatively decent forecast using only the historical data. The problem with the statistical method is that it loses its accuracy as the forecasting horizon grows. Zjavka reports that time series forecasting based technique will become ineffective for horizons above 6 hours due to the growing uncertainty beyond 6 hours forecast [58]. We could see a drastic change from calm, windy period into violent gust in six hours, which causes the purely statistical method to lose its efficiency because of this uncertain dynamic. His judgement is approved by the fact that most of the current approach for long-term forecasting are dominated by the physical method or its hybrid, rather than using a purely statistical method [13, 30, 50].

1.3. Time Series

Time series is a sequence of data points ordered in time and is given by

$$\mathbf{y} = \{y_1, y_2, \dots, y_t\} \tag{1.1}$$

y_1 is the value of time series taken at the first time point, y_2 is the value taken at the second time point, and so forth. In term of statistics, the series denoted by Equation 1.1 is a realisation of an underlying stochastic process. While the stochastic process might be continuous in time, in practice observations of the process are sampled discretely, with time indices $t = 0, \pm 1, \pm 2, \dots$. In this research, only the case of *discrete-time* and *uniformly-sampled* time series is studied. Such time series, for example, is the hourly temperature of the city of Delft. The temperature itself is changing continuously over time, but here the temperature is sampled uniformly every one hour.

The study of time series involves in modelling the underlying stochastic process that generates time series. A complete description of this stochastic process is the joint probability distribution between observations

$$P(Y_1 \leq y_1, Y_2 \leq y_2, \dots, Y_t \leq y_t) \tag{1.2}$$

where Y is a random variable with y as its realisation. Although the joint distribution is the most accurate model of a time series, calculating the joint probability distribution might be too difficult for many applications. Another way to provide a meaningful description of the series is to calculate the mean

$$\mu_t = E[y_t] \tag{1.3}$$

and the autocovariance of the series.

Autocovariance measures the statistical dependence between two observations in a time series. The term auto is added because autocovariance calculates the covariance between itself, only at different times. The autocovariance between observation at time s, y_s, and time t, y_t is given by

$$\gamma(s,t) = \text{cov}(y_s, y_t) = E[(y_s - \mu_s)(y_t - \mu_t)] \tag{1.4}$$

To correctly forecast a time series, we must be able to find and exploit some regularities that exist in our time series of interest. *Stationarity* is one example of regularity that is often assumed by many time series analysis techniques because of several nice properties it has. A *stationary* time series fulfils a condition where the joint probability distribution between a set of observations $\{y_{t_1}, y_{t_2}, \dots, y_{t_k}\}$ and a time-shifted set of observations $\{y_{t_{1+l}}, y_{t_{2+l}}, \dots, y_{t_{k+l}}\}$ is equal, that is

$$P(Y_{t_1} \leq y_{t_1}, \dots, Y_{t_k} \leq y_{t_k}) = P(Y_{t_{1+l}} \leq y_{t_{1+l}}, \dots, Y_{t_{k+l}} \leq y_{t_{k+l}}) \tag{1.5}$$

for all k, all time points t, and all time shifts $l = 0, \pm 1, \pm 2, \dots$.

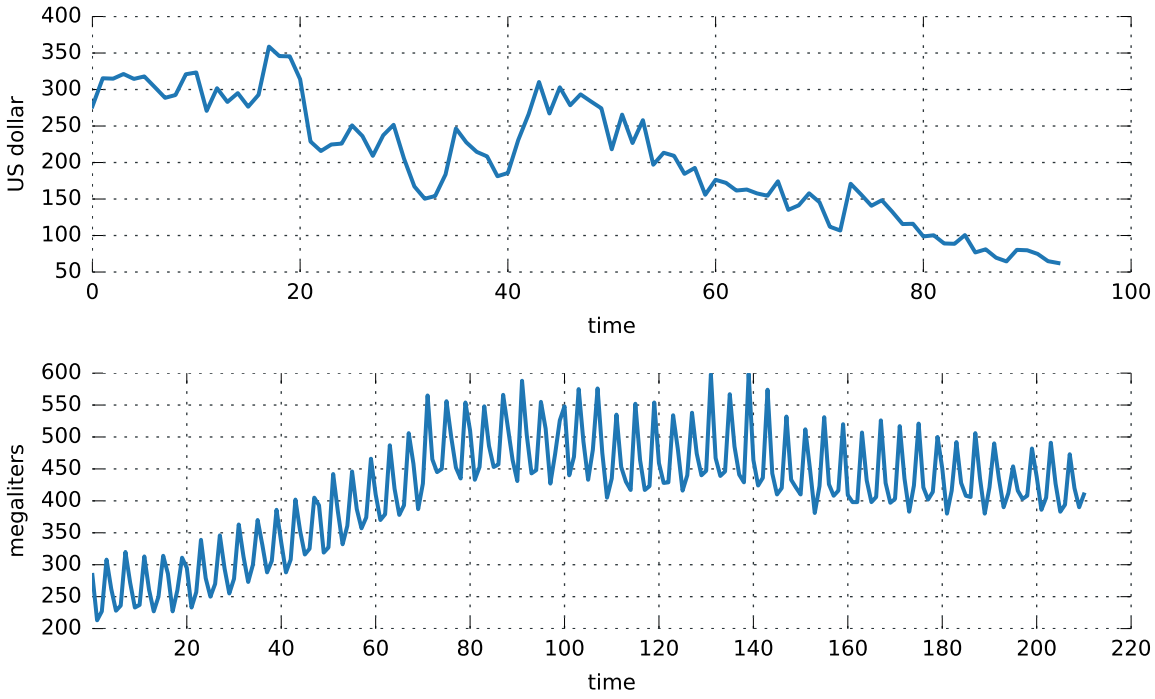


Figure 1.2: Examples of time series with a trend (first row) and season (second row). First row: Price of dozen eggs in US, 1900–1993, in constant dollars [55]. Second row: Total quarterly beer production in Australia (in megalitres) from 1956: Q1 to 2008: Q3 [28]

This stationary property is too *strict* for most time series since computing every joint distribution between all shifts is intractable for many applications. A more loose or weaker definition of stationary is to define the stationary property based on the mean and the autocovariance of the time series. In many kinds of literature, this is often named the *weak* stationary or *wide-sense* stationary. A *weak* stationary time series has the following properties

(i) the mean is constant and independent of time t

(ii) the autocovariance function $\gamma(s,t)$ depends on s and t only through the difference $|s-t|$

Given that $s = t + l$ where l expresses the time shift or lag, the autocovariance function of a weak stationary time series is equal to

$$\gamma(t+l,t) = \text{cov}(y_{t+l}, y_t) = \text{cov}(y_l, y_0) = \gamma(l)$$

From the above expression, it can be seen that the autocovariance function only depends on the difference between observations l and does not depend on time t. For the rest of this document, the term stationary refers to the weak stationary unless it is specified.

A stationary time series is very desirable in time series forecasting because its statistical (up to the second order moment) properties do not depend on time. That means if we can estimate those properties correctly, then those properties should hold for all time points and consequently, we can forecast for every time point in the future. Unfortunately, for many time series the stationary property does not hold. There are two structures which are commonly found in time series that render the stationary property invalid. These structures are *trend* and *season*. These structures clearly violate the constant mean property of a stationary time series. Figure 1.2 shows examples of time series with trend and season. The time series shown in the first row exhibits downward trend while the second row shows a time series with a yearly season.

1.4. Modelling Approach

The previous section discusses how the joint distribution between observations in Equation 1.5 is impossible to compute for most of time series applications. The joint distribution can be approximated by assuming stationarity but still the assumption is limited, therefore an alternative modelling approach is needed.

1.4.1. General Model

Consider the following general setting. We have a training set $\mathcal{D}$, where the elements are T pairs of d-dimensional input vectors and output scalar, $\mathcal{D} = \{(\mathbf{x}_t, y_t)|t = 1, \dots, T\}$. y_t is a time series observation at time t. $\mathbf{x}_t$ is a general input vector, $\mathbf{x}_t = [x_1, \dots, x_d]^T$, where the value depends on the forecasting model. All input vectors are often aggregated into a design matrix $\mathbf{X}$, and the output values are aggregated into output vector $\mathbf{y}$

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_T^T \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_T \end{bmatrix} \tag{1.6}$$

A general time series model is given by

$$y_t = f(\mathbf{x}_t; \boldsymbol{\theta}) + \epsilon_t \tag{1.7}$$

where f is a function that maps the input $\mathbf{x}$ at t to the time series observation at time t and the function is parameterised by a parameter vector $\boldsymbol{\theta}$. ϵ_t is an I.I.D noise and assumed to be Gaussian distributed

$$\epsilon_t \sim \mathcal{N}(0, \sigma_N^2) \tag{1.8}$$

if not mentioned otherwise. The h-step-ahead forecast is done by plugging in the test input $\mathbf{x}_{T+h}$ into the function f

$$y_{T+h} = f(\mathbf{x}_{T+h}; \hat{\boldsymbol{\theta}}) \tag{1.9}$$

where the parameter $\hat{\boldsymbol{\theta}}$ has already been learned from the training data. With this general model, this section will discuss several specific models for time series forecasting, starting with a very simple regression model.

1.4.2. Persistence

The persistence or also commonly referred as the naïve method directly forecasts all horizons with the last observed value in the series

$$y_{T+h} = y_T \qquad \text{for all } h \tag{1.10}$$

Although it is surprisingly simple, the persistence is shown to perform better than more complicated models in wind power forecasting, especially when forecasting a very-short horizon [13, 50]. So, any forecasting model should be able to at least outperform the persistence method, otherwise the existence of the forecasting model is questionable.

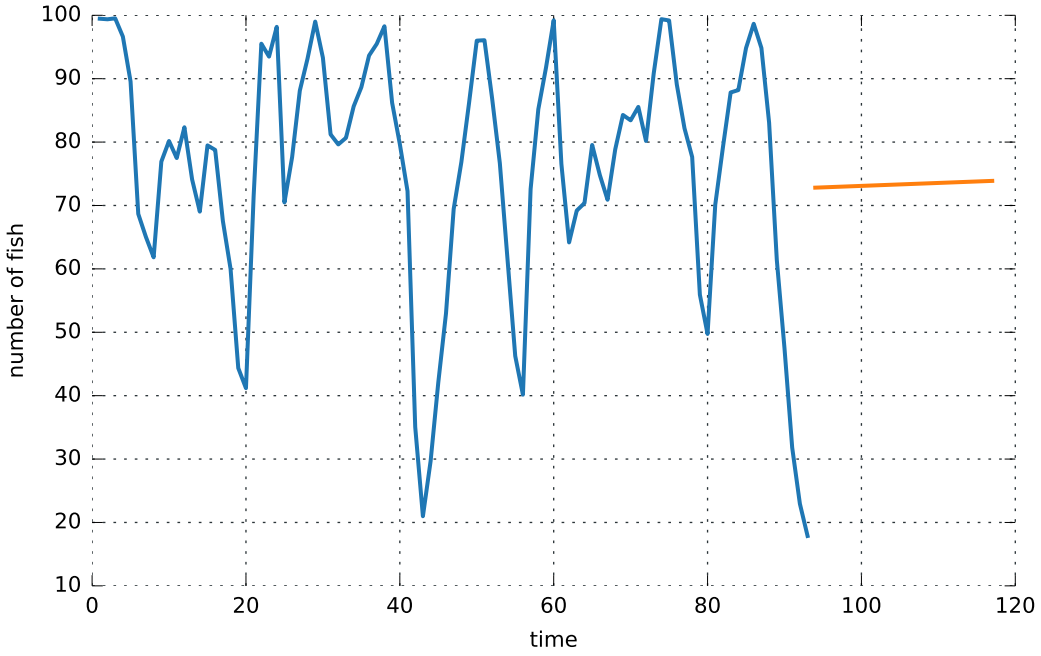


Figure 1.3: Forecast using a linear regression-in-time. The result is unsatisfactory as the forecasts is equal to the mean of the time series. Data: Recruitment (number of new fish) for a period of 453 months ranging over the years 1950-1987 (the plot starts from 1980) [48].

1.4.3. Simple Regression

For the first model, a simple linear regression in time model is proposed, where the input is the time index t

$$\mathbf{x}_t = t \tag{1.11}$$

and the forecasting model is given by

$$y_t = \beta_0 + \beta_1 t + \epsilon_t \tag{1.12}$$

Figure 1.3 shows an example of the forecast with the simple regression model. The regression can only capture the mean of the time series. It can be seen clearly that this linear regression in time is inadequate to model a complex time series.

1.4.4. Classical Approach: ARIMA

One of the most prominent time series forecasting models is the ARIMA (autoregressive integrated moving average). This model has a more complex modelling approach than the previous linear regression model. The ARIMA model originates from the ARMA (autoregressive moving average) model while the ARMA model itself is a combination of an AR (autoregressive) model with an MA (moving average) model. The concept of both AR and MA is crucial in many time series model, not only limited to the ARIMA model.

The autoregressive model is based on the idea that the current value of a time series is a *linear combination* of its past values. The number of previous values is often named as the *order* of the autoregressive and symbolised by p. An autoregressive model of order p or $AR(p)$ is given by

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + ... + \phi_p y_{t-p} + \epsilon_t \tag{1.13}$$

where $\phi_1, \dots, \phi_p$ are parameters of the model, c is a constant, and ϵ is a white noise, $\epsilon_t \sim \mathcal{N}(0, \sigma_N^2)$.

Moving average (MA) provides an alternative model to the autoregressive representation where it models the current value of a time series as a linear combination of several past errors. The moving average model of order q, or MA(q), is equal to

$$y_t = c + \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \dots + \theta_q \epsilon_{t-q} \tag{1.14}$$

where $\theta_1, \dots, \theta_q$ are parameters of the moving average.

The combination of an AR(p) and MA(q) model is the ARMA(p, q) model, and is given by

$$y_t = c + \epsilon_t + \sum_{i=1}^{p} \phi_i y_{t-i} + \sum_{j=1}^{q} \theta_j \epsilon_{t-j} \tag{1.15}$$

To make it consistent with the general model that has been outlined in the Section 1.4.1, for ARMA model we have

$$\mathbf{x}_t = [y_{t-1}, y_{t-2}, \dots, y_{t-p}, \epsilon_{t-1}, \epsilon_{t-2}, \dots, \epsilon_{t-q}]^T \tag{1.16}$$

$$f(\mathbf{x}_t) = c + \sum_{i=1}^{p} \phi_i y_{t-i} + \sum_{j=1}^{q} \theta_j \epsilon_{t-j} \tag{1.17}$$

$$\boldsymbol{\theta} = [\phi_1, \dots, \phi_p, \theta_1, \dots, \theta_q]^T \tag{1.18}$$

The ARMA model is a powerful time series model. First reason is because the ARMA model is an actualisation of the Wold's Decomposition Theorem. The theorem states that any covariance stationary process can be decomposed into two mutually uncorrelated component processes, a linear combination of lags of a white noise process and future values of which can be predicted exactly by some linear function of past observations [7]. In addition to that, the ARMA model requires only two hyperparameters, the p and q. These reasons mean that an ARMA model is capable of modelling any stationary time series with a minimum set of user-defined parameters.

The ARMA model works under the assumption that the time series is stationary. This assumption is limiting for many time series applications due to the presence of trend and season that is common in many real world data. To handle this limitation, the non-stationary time series must be made stationary first before modelling the time series with the ARMA model. The process of removing trend and season is called de-trending and de-seasonalising respectively.

One way to perform de-trending is by *differencing*. Differencing compute the difference between consecutive observations

$$y_t' = y_t - y_{t-1} \tag{1.19}$$

The above equation calculates the first-order difference. It is sometimes necessary to compute a higher order differencing like a second-order differencing if a lower order differencing still cannot remove the trend. The second order differencing is equal to

$$y_t'' = y_t' - y_{t-1}' \tag{1.20}$$

$$= (y_t - y_{t-1}) - (y_{t-1} - y_{t-2}) \tag{1.21}$$

ARIMA model is simply an ARMA model with a dth order of differencing. The 'integrated' term comes from this differencing procedure, where integration is the opposite of differencing. An ARIMA model is

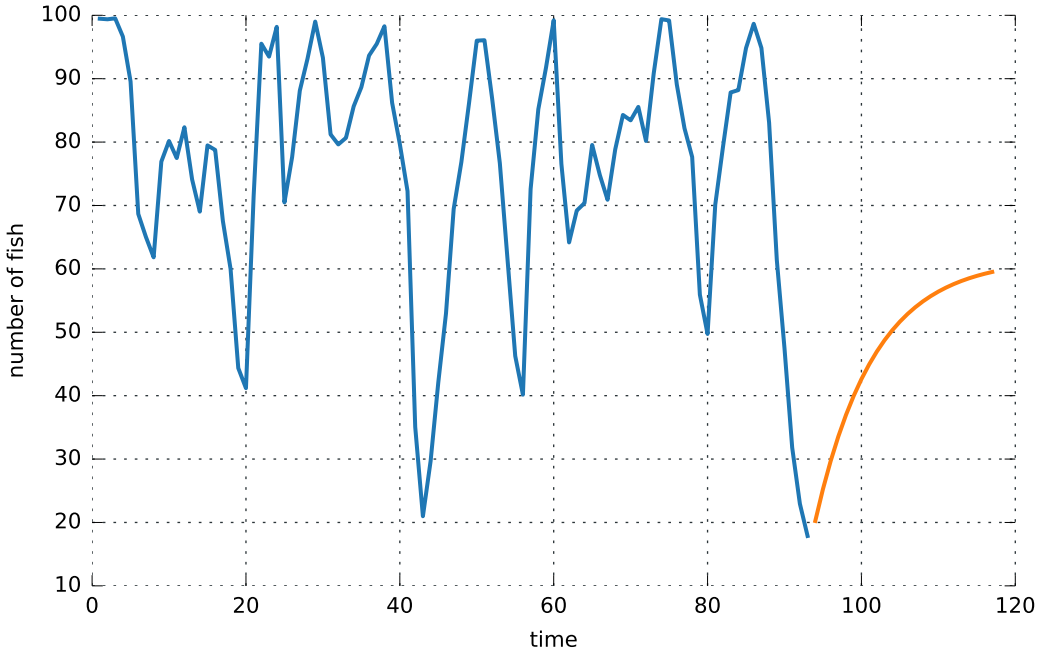


Figure 1.4: A 24-step-ahead forecast using ARIMA(1, 0, 1). It is shown that the forecast converges to the mean of the function after several horizons. Data: Recruitment (number of new fish) for a period of 453 months ranging from the years 1950-1987 (the plot starts from 1980) [48].

often notated as ARIMA(p, d, q) where p is the order of the AR, q is the order of the MA, and d is the order of differencing. An ARIMA$(1, 0, 2)$ is equal to an ARMA$(1, 2)$.

Similar to the de-trending, de-seasonalising can be performed through what is called the *seasonal differencing*. Seasonal differencing computes the difference between an observation and the observation at the same period from the previous season. For example, if we know that a time series has a season that is repeating every 12 time-unit, then the seasonal difference is equal to $y_t - y_{t-12}$. Seasonal ARIMA or SARIMA is a more general ARIMA implementation that incorporates both non-seasonal differencing to remove the trend and seasonal differencing to remove the season.

Even though the ARIMA has been around for quite some time, this technique is still being used in a significant number of application domains that require forecasts, such as tourism [20, 52], transportation [3, 32], medicine [38, 46], engineering [14, 53] and many more. The popularity of the ARIMA model is contributed from the fact that ARIMA is well-studied, theoretically mature, and backed up by ubiquitous software implementations. The last aspect is crucial to the widespread adoption of the ARIMA model outside the statistics community. Social scientists, for example, could easily incorporate forecasting using ARIMA from their statistical software of choice.

Behind its prevalence, we must be aware that ARIMA model is only effective for short-term forecasting. The forecasting power of an ARIMA model decreases as the forecasting horizon increases because the ARIMA forecast *converges to the mean* of the observations as the forecast horizon grows [48]. Figure 1.4 displays the example of a mean convergence. In this example, 24-step-ahead forecasts on a time series are done using ARIMA$(1, 0, 1)$. After several horizons, it is evident that the forecast converges to mean of the time series. Appendix A shows the mean convergence of an ARIMA$(1, 0, 1)$ model analytically.

1.4.5. Gaussian Process Regression

Previously we have discussed two forecasting models, the simple regression in time and ARIMA model. The first model is too simple, unable to model complex pattern in time series, while the second model

is better but suffers from long-term forecasting issue. The shortcomings of these two techniques are caused by their limited assumptions over the underlying function f. The simple regression assumes that y_t linear in time whereas the ARIMA model assumes that y_t is linear in the p number of past data and q number of past errors. Figure 1.3 and Figure 1.4 prove that these linear assumptions are inadequate to model a long-term forecasting problem and thus a more powerful model is needed.

The Gaussian process (GP) regression provides a solution to the problem as mentioned earlier. This model has a much less restrictive assumption over the function f. A GP regression only assumes that the function f is sufficiently *smooth*. This smoothness of the function can be defined by tuning the parameters of a Gaussian process regression. Technically, a GP regression technique states that the function f is random and drawn from a multivariate Gaussian distribution

$$\begin{bmatrix} f(\mathbf{x}_i) \\ f(\mathbf{x}_j) \\ f(\mathbf{x}_k) \\ \vdots \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} m(\mathbf{x}_i) \\ m(\mathbf{x}_j) \\ m(\mathbf{x}_k) \\ \vdots \end{bmatrix}, \begin{bmatrix} k(\mathbf{x}_i,\mathbf{x}_i) & k(\mathbf{x}_i,\mathbf{x}_j) & k(\mathbf{x}_i,\mathbf{x}_k) & \cdots \\ k(\mathbf{x}_j,\mathbf{x}_i) & k(\mathbf{x}_j,\mathbf{x}_j) & k(\mathbf{x}_j,\mathbf{x}_k) & \cdots \\ k(\mathbf{x}_k,\mathbf{x}_i) & k(\mathbf{x}_k,\mathbf{x}_j) & k(\mathbf{x}_k,\mathbf{x}_k) & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \right)$$

where m and k denote the mean and covariance function, respectively. Since the function f is random, the optimum f will be inferred via the Bayesian inference. Through this GP regression technique, various kinds of function can be modelled, not only a linear function but also a complex non-linear function, by modifying the mean and covariance function of this normal distribution. This property highlights the flexibility of a GP regression model. More details about GP regression will be discussed in Chapter 2.

In addition to its flexibility, the benefit of using GP regression is summarised below:

- Flexible. We should see that the GP regression has a flexible modelling capability through kernel modification. Hence, it can model a time series with a broad range of complexity.
- Probabilistic model. Prediction from a GP regression model is a distribution instead of a point forecast. Therefore, we will have a point forecast (the average of the distribution) and its uncertainty (the variance of the distribution).
- More robust to overfitting. GP is a special class of Bayesian probabilistic modelling, where we can benefit from the Bayesian regularisation in a closed-form solution.

There are two main approaches for GP regression in time series forecasting. First is the *autoregressive* approach, where the input to the function is the autoregressive values up to order p, which is similar to the AR process of the ARIMA model

$$\mathbf{x}_t = [y_{t-1}, \dots, y_{t-p}]^T \tag{1.22}$$

The different is that the function f is not limited to a linear function like ARIMA. Frigola refers this technique as the non-linear autoregressive (NAR) [18].

The second approach is similar to the regression in time model, where the input is the time index

$$\mathbf{x}_t = t \tag{1.23}$$

To extend the model beyond a linear function of time, a complex covariance function is used, where this covariance function is a combination of several covariance functions

$$k(t,t') = k_1(t,t') + k_2(t,t') \times k_3(t,t') + \dots \tag{1.24}$$

Each covariance function is responsible for modelling a certain structure in the time series. For example, the first covariance function in the above equation corresponds to a linear trend while the second and third covariance functions correspond to a medium-term season. This technique is popularised by

Duvenaud et al., where they named this technique the *structure modelling*. Both of these approaches will be described in more details in Chapter 3 and 4.

The autoregressive approach is first proposed by Brahim-Belhouari and Belmak [8]. Their research reports that the GP forecasting method outperforms the Radial Basis Function (RBF) neural network on a non-linear time series data. Since then, this autoregressive technique becomes widespread as several similar autoregressive methods are proposed. Girard et al. extend this autoregressive approach to perform a multiple-step-ahead forecasting [19]. Hachino and Kadirkamanathan aim to improve the existing autoregressive GP method by employing a genetic algorithm during the training process [23]. They argue that the standard training method is susceptible to local minima, so they propose a genetic algorithm as an alternative. More advanced scenarios involving multivariate time series forecasting are also studied. Chapados and Bengio incorporate several exogenous variables in addition to the autoregressive vector for forecasting commodity demands [11]. Osborne et al. augment the information gathered from various neighbouring sensors to forecast the weather by exploiting the correlation between sensors through GP kernels [39]. Mori and Kurata apply autoregressive GP for forecasting wind speed and report that GP has better forecasting performance than the RBF and feedforward neural network [35].

The idea of structure modelling is first proposed by Rasmussen and Williams [44, p.118]. They perform a regression in time and modelling the covariance function by hand, where each covariance function corresponds to a certain pattern that exists in the time series. Duvenaud et al. improve the solution of Rasmussen and Williams by proposing an algorithm to construct the complex covariance function automatically.

1.4.6. Other Methods

Unfortunately, due to the limited amount of time during this research, comparison with another time series forecasting, possibly a more complicated state-of-the-art models, is not possible. A neural network (NN) based technique is probably an ideal candidate for comparison if time permit. In the early 2000's, many types of research aim to push neural network as an alternative to the more classical ARIMA, with feedforward neural network as the most dominant architecture [27, 57]. After the resurgence of deep learning, interests in neural network time series forecasting get renewed. This time, the architecture based on Recurrent Neural Network (RNN) is prominent. The feedback loop structure in the RNN is deemed more suitable for learning from time series than the feedforward architecture [22]. RNN has shown promising results in several sequential data such as speech recognition [26] and language modelling [34].

1.5. Problem Statement

With time series forecasting problem in mind, the ARIMA model remains a popular forecasting model that is still being used in the recent years. However, the choice of ARIMA has to be reconsidered for long-term forecasting model. The last paragraph of Section 1.4.4 outlines the problem of mean convergence that ARIMA suffers for long-term forecasting. We might argue that this convergence is still acceptable, because in long-term forecasting the uncertainty grows larger, thus the mean convergence is the best compromise that we can get. Nevertheless, having a more powerful model that can forecast ahead in the future with greater confidence is highly desirable. In wind energy forecasting, for example, an accurate long-term forecast can help with the maintenance decision of wind turbines.

In Section 1.4.5, the GP regression is discussed as a potential alternative to ARIMA due to its flexibility. Two approaches of GP-based time series forecasting are explored, the *autoregressive* and the *structure modelling*. Both of these approaches have merits and show promising forecasting results in their respective research. Still, most of the research works mentioned above focus on short-term, not on long-term forecasting. Yan et al. [56] provide a long-term forecasting study but more focused on the *autoregressive* approach, and no comparison is made with another time series forecasting technique.

1.6. Research Objective

The previous section underlines the problem of ARIMA for long-term forecasting and how two approaches of GP regression for time series forecasting, the *autoregressive* and the *structure modelling* are potential for a better long-term time series forecasting performance. Unfortunately, previous works on these two approaches focus on short-term forecasting. Moreover, to the best of our knowledge, there is no empirical study ever done to compare the performance of these GP regression techniques with the ARIMA model. These propositions underline the importance of this research.

This research aims to provide an empirical study to compare the long-term forecasting performance between the ARIMA and Gaussian process regression techniques. The objective of this research is expanded into these following research questions.

1. Between the two Gaussian process regression approaches, the autoregressive and the structure modelling, how do their long-term forecasting performances compare with the ARIMA model?
2. Between the two Gaussian process regression approaches, the autoregressive and the structure modelling, which has better long-term forecasting performance?

1.7. Outline

This research report will be structured as follows. Chapter 2 will elaborate the theory behind Gaussian process. The next two chapters, Chapter 3 and Chapter 4, will discuss in detail about the structure modelling and the autoregressive approach, respectively. In Chapter 5, the experiment methodology will be outlined, and then the results of the experiment will also be reported. Lastly, this report will be closed with some conclusions about the research, as well as future discussion points in Chapter 6.

2

Gaussian Process

This chapter discusses what a Gaussian process is and how it can be used for regression.

2.1. Bayesian Linear Regression

Let us consider a general linear regression model that follows the notation from Section 1.4.1. The input vector $\mathbf{x}$, function f and parameters $\boldsymbol{\theta}$ are given by

$$\begin{aligned} \mathbf{x}_t &= [x_1, \dots, x_d]^T \\ f(\mathbf{x}_t) &= \beta_0 + \beta_1 x_1 + \dots + \beta_d x_d \\ \boldsymbol{\theta} &= [\beta_0, \beta_1, \dots, \beta_d]^T \end{aligned}$$

Inferring the parameter $\boldsymbol{\theta}$ from data is through the Bayes' rule

$$p(\boldsymbol{\theta}|\mathbf{X}, \mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathbf{y}|\mathbf{X})} \tag{2.1}$$

where the denominator can be calculated as

$$p(\mathbf{y}|\mathbf{X}) = \int p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta})p(\boldsymbol{\theta})d\boldsymbol{\theta} \tag{2.2}$$

Each component of the Equation 2.1 is usually referred as

$$\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{evidence or marginal likelihood}} \tag{2.3}$$

If we disregard the normalising factor of the evidence, we can explain the Bayes' rule by

$$p(\boldsymbol{\theta}|\mathbf{X}, \mathbf{y}) \propto p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta})p(\boldsymbol{\theta}) \tag{2.4}$$

This equation actually summaries all the intuition of the Bayesian inference. We start with the prior belief of the parameter without seeing the data. Once we have seen the data, our belief about the parameter is updated into a posterior belief through the likelihood.

In the Bayesian linear regression, the prior function over the parameter is Gaussian distributed, $p(\boldsymbol{\theta}) \sim \mathcal{N}(0, \boldsymbol{\Sigma}_p)$ where here it is assumed that the covariance of the prior $\boldsymbol{\Sigma}_p$ is given. Assuming that the

observations are I.I.D, the likelihood function $p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta})$ is another Gaussian distribution and is given by

$$p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) = \prod_{t=1}^{T} p(y_t|\mathbf{x}_t, \boldsymbol{\theta}) = \prod_{t=1}^{T} \frac{1}{\sqrt{2\pi}\sigma_N} \exp\left(-\frac{(y_i - \mathbf{x}_i^T\boldsymbol{\theta})^2}{2\sigma_N^2}\right)$$
$$= \frac{1}{(2\pi\sigma_N^2)^{T/2}} \exp\left(-\frac{\|\mathbf{y} - \mathbf{X}\boldsymbol{\theta}\|_2^2}{2\sigma_N^2}\right)$$
$$p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) \sim \mathcal{N}(\mathbf{y}|\mathbf{X}\boldsymbol{\theta}, \sigma_N^2\mathbf{I}) \tag{2.5}$$

where $\|\cdot\|_2$ denotes the Euclidean norm.

Since both the prior and likelihood are Gaussian, the parameter posterior $p(\boldsymbol{\theta}|\mathbf{X}, \mathbf{y})$ is also Gaussian distributed

$$p(\boldsymbol{\theta}|\mathbf{X}, \mathbf{y}) \sim \mathcal{N}(\boldsymbol{\theta}|\bar{\boldsymbol{\mu}}, \bar{\boldsymbol{\Sigma}}) \tag{2.6}$$
$$\bar{\boldsymbol{\mu}} = \frac{1}{\sigma_N^2}\mathbf{A}^{-1}\mathbf{X}^T\mathbf{y}$$
$$\bar{\boldsymbol{\Sigma}} = \mathbf{A}^{-1}$$

where $\mathbf{A}^{-1} = (\frac{1}{\sigma_N^2}\mathbf{X}^T\mathbf{X} + \boldsymbol{\Sigma}_p^{-1})^{-1}$.

With the parameter posterior, we can predict the unseen data by computing the predictive posterior

$$p(y_*|\mathbf{x}_*, \mathbf{X}, \mathbf{y}) = \int p(y_*|\boldsymbol{\theta}, \mathbf{x}_*)p(\boldsymbol{\theta}|\mathbf{X}, \mathbf{y})d\boldsymbol{\theta} \tag{2.7}$$

This probability distribution is called the *predictive posterior* or *predictive distribution*. Broadly speaking, a predictive posterior is a result of predicting over all possible parameters (the left side of the integral) weighted by the parameter posterior (the right side of the integral). The parameter posterior is again Gaussian distributed

$$p(y_*|\mathbf{x}_*, \mathbf{X}, \mathbf{y}) \sim \mathcal{N}(y_*|\mu_*, \sigma_*^2) \tag{2.8}$$
$$\mu_* = \mathbf{x}_*^T\bar{\boldsymbol{\mu}}$$
$$\sigma_*^2 = \mathbf{x}_*^T\bar{\boldsymbol{\Sigma}}\mathbf{x}_*$$

If we inspect the left term of the integral in Equation 2.7, we can see that the prediction does not rely on our training data. Equation 2.8 similarly indicates that the predictive posterior makes use the information from the test data and the parameter posterior. In a parametric model such as the linear regression that has been discussed previously, all the information that the model get from training data are compressed into the parameter vector $\boldsymbol{\theta}$ and predictions are made through these parameters only. This situation is limiting, because no matter how large our training set is, our prediction is determined solely by this finite set of parameters. Frigola considers the parameters as an information *bottleneck* for prediction [18].

Discussion in the previous paragraph highlights the inflexibility of a parametric model, which encourages the utilisation of the non-parametric model. In the non-parametric model, the training data is not condensed into a finite set of parameter. Instead, it requires the full dataset to make a prediction on

the new data. Because of that, we can consider the non-parametric model has an infinite size of parameters since the 'parameters' of a non-parametric model grows with the scale of the training data. The next section will discuss one class of a non-parametric regression model which is the Gaussian process regression.

2.2. Gaussian Process Regression

If we look at the general model from Equation 1.7, the primary goal of the forecasting model is to compute the underlying function f from the noisy observation y. In a parametric model such as the Bayesian linear regression from Section 2.1, we obtain the function f through the parameter $\boldsymbol{\theta}$. We are often more interested in the predictive accuracy of the model and less about the parameter of the function [1]. So, instead of inferring the parameters to get the latent function of interests, an alternative is to infer the function f directly, as if the function f is the 'parameters' of our model. We shall infer the function f from the data through the Bayesian inference.

The function f is random and we need to assign a probability distribution over f. Rasmussen and Williams suggest the Gaussian processes (GP) to describe the probability distribution over function f [44]. A formal definition of a Gaussian process according to Rasmussen and Williams [44] is

> A Gaussian process is a collection of random variables, any finite number of which have a joint Gaussian distribution.

By this definition, a value of a function f evaluated at a point $\mathbf{x}$ is distributed according to a multivariate Gaussian distribution

$$\begin{bmatrix} f(\mathbf{x}_i) \\ f(\mathbf{x}_j) \\ f(\mathbf{x}_k) \\ \vdots \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} m(\mathbf{x}_i) \\ m(\mathbf{x}_j) \\ m(\mathbf{x}_k) \\ \vdots \end{bmatrix}, \begin{bmatrix} k(\mathbf{x}_i,\mathbf{x}_i) & k(\mathbf{x}_i,\mathbf{x}_j) & k(\mathbf{x}_i,\mathbf{x}_k) & \cdots \\ k(\mathbf{x}_j,\mathbf{x}_i) & k(\mathbf{x}_j,\mathbf{x}_j) & k(\mathbf{x}_j,\mathbf{x}_k) & \cdots \\ k(\mathbf{x}_k,\mathbf{x}_i) & k(\mathbf{x}_k,\mathbf{x}_j) & k(\mathbf{x}_k,\mathbf{x}_k) & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \right) \tag{2.9}$$

where m and k denote the mean and covariance functions respectively. As an alternative notation to Equation 2.9, we define a function that is distributed according to a Gaussian process as

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')) \tag{2.10}$$

By assuming that the function is generated according to a GP, we inherit some properties of the Gaussian distribution that will help the computation during the inference. Moreover, we have a more flexible definition of our function. The only assumption made by GP is that the function is *sufficiently smooth*. Intuitively, if two points, $\mathbf{x}$ and $\mathbf{x}'$, are similar, then $f(\mathbf{x})$ and $f(\mathbf{x}')$ should also be similar, which explains why the function generated by a GP is smooth. Our assumption of this similarity or smoothness is encoded by the covariance function k. We can define a broad range of functions just by defining different covariance functions. Figure 2.1 shows three random functions sampled from four different kinds of covariance functions. It can be seen that the functions can have varying characteristics, ranging from a linear to a noisy non-linear function. This trait demonstrates how a GP regression can be very flexible in modelling a function. Different types of covariance functions will be discussed in Section 2.3.

We follow a similar inference steps from Section 2.1 to make a prediction using a Gaussian process regression. It has the same two-step inference processes, computing the posterior over function, then making predictions by computing the predictive posterior. The posterior over function is given by

$$p(\mathbf{f}|\mathbf{X}, \mathbf{y}) \propto p(\mathbf{y}|\mathbf{f})p(\mathbf{f}|\mathbf{X}) \tag{2.11}$$

which is proportional to the likelihood of the observation $\mathbf{y}$ given a random function $\mathbf{f}$ times the prior over function.

[1] unless we are interested in making an interpretable model, which has gained interested in the machine learning community recently [45]

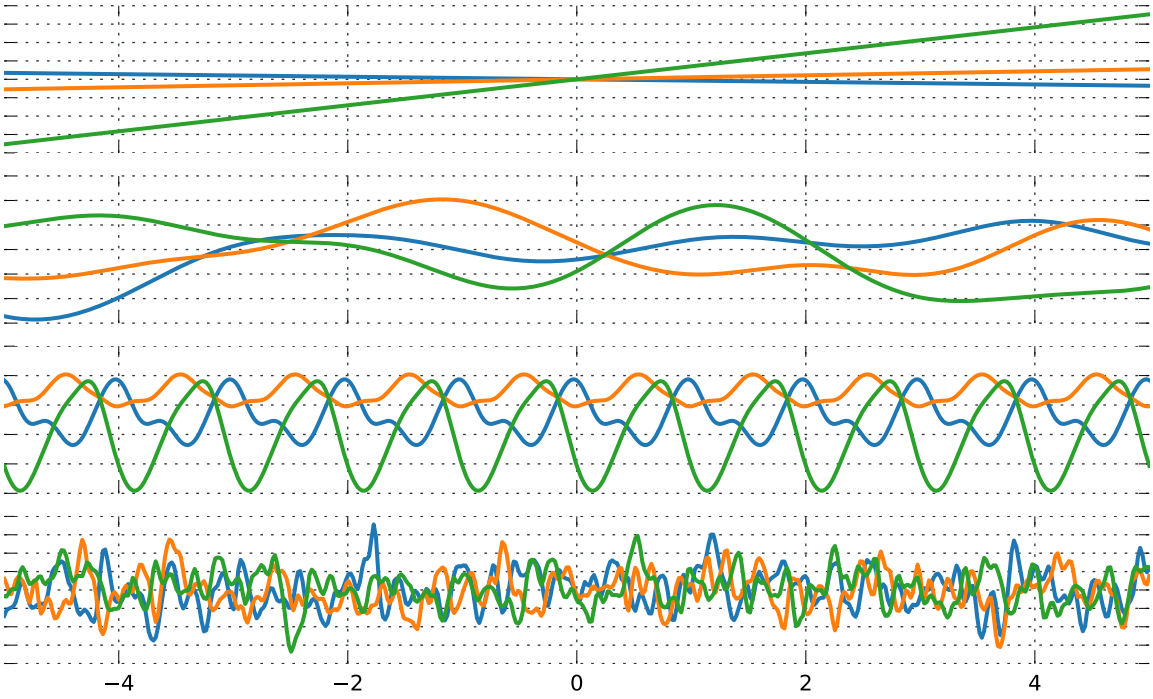


Figure 2.1: This figure displays three random functions that are sampled from Gaussian processes with different kinds of covariance functions. First row: Linear. Second row: RBF. Third row: Periodic. Fourth row: Rational Quadratic.

As it has been discussed earlier in this section, the prior over function is distributed according to a Gaussian process which follows Equation 2.9

$$p(\mathbf{f}|\mathbf{X}) \sim \mathcal{N}(\mathbf{f}|\boldsymbol{\mu}(\mathbf{X}), \mathbf{K}(\mathbf{X},\mathbf{X})) \tag{2.12}$$

where $\mathbf{f}$ is a vector containing all function values evaluated at our training input, $\mathbf{f} = [f(\mathbf{x}_1, \dots, f(\mathbf{x}_T)]^T$. $\boldsymbol{\mu}(\mathbf{X})$ is a mean vector containing the mean function evaluated at training input, $\boldsymbol{\mu}(\mathbf{X}) = [m(\mathbf{x}_1), \dots, m(\mathbf{x}_T)]^T$. Similarly, $\mathbf{K}(\mathbf{X},\mathbf{X})$ is a covariance matrix in which the elements are covariance functions between all training inputs, $\mathbf{K}(\mathbf{X},\mathbf{X})_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$. Most of the time, the mean of the prior is set to zero, because the GP is flexible enough the model the mean arbitrary well just from the covariance function

$$p(\mathbf{f}|\mathbf{X}) \sim \mathcal{N}(\mathbf{f}|\mathbf{0}, \mathbf{K}(\mathbf{X},\mathbf{X})) \tag{2.13}$$

where $\mathbf{0}$ is a vector with zero as elements.

The likelihood function is similar to the likelihood function in Equation 2.5, where now the mean of the likelihood function is centred on an arbitrary $\mathbf{f}$

$$p(\mathbf{y}|\mathbf{f}) \sim \mathcal{N}(\mathbf{y}|\mathbf{f}, \sigma_N^2\mathbf{I}) \tag{2.14}$$

Both the likelihood and prior are Gaussians, so the posterior over function is another Gaussian distribution

$$\begin{aligned} p(\mathbf{f}|\mathbf{X},\mathbf{y}) &\sim \mathcal{N}(\mathbf{f}|\bar{\boldsymbol{\mu}}, \bar{\boldsymbol{\Sigma}}) \\ \bar{\boldsymbol{\mu}} &= \mathbf{K}(\mathbf{X},\mathbf{X})[\mathbf{K}(\mathbf{X},\mathbf{X}) + \sigma_N^2\mathbf{I}]^{-1}\mathbf{y} \\ \bar{\boldsymbol{\Sigma}} &= \mathbf{K}(\mathbf{X},\mathbf{X})[\mathbf{K}(\mathbf{X},\mathbf{X}) + \sigma_N^2\mathbf{I}]^{-1}\sigma_N^2\mathbf{I} \end{aligned} \tag{2.15}$$

Once we have the posterior over function, we can predict new unseen data by computing the predictive posterior, similar to the predictive posterior found on Equation 2.7

$$p(\mathbf{y}_*|\mathbf{X}_*, \mathbf{X}, \mathbf{y}) = \int p(\mathbf{y}_*|\mathbf{X}_*, \mathbf{f}, \mathbf{X}) p(\mathbf{f}|\mathbf{X}, \mathbf{y}) d\mathbf{f} \tag{2.16}$$

Here we assume that the new unseen data is a vector $\mathbf{y}_*$ with input matrix $\mathbf{X}_*$. The term on the right side of the integral is the posterior over function that is found in Equation 2.15. The term on the left is the likelihood of new data given a function $\mathbf{f}$. This likelihood can be computed analytically by calculating the conditional distribution the $\mathbf{y}_*$ given the function $\mathbf{f}$ which is another Gaussian distribution. The reader can refer to Rasmussen and Williams [44, p.16] for more detail about the calculation of this likelihood. The likelihood can be calculated as

$$\begin{aligned} p(\mathbf{y}_*|\mathbf{X}_*, \mathbf{f}, \mathbf{X}) &\sim \mathcal{N}(\mathbf{y}_*|\tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{\Sigma}}) \\ \tilde{\boldsymbol{\mu}} &= \mathbf{K}(\mathbf{X}_*, \mathbf{X})\mathbf{K}(\mathbf{X}, \mathbf{X})^{-1}\mathbf{f} \\ \tilde{\boldsymbol{\Sigma}} &= \mathbf{K}(\mathbf{X}_*, \mathbf{X}_*) - \mathbf{K}(\mathbf{X}_*, \mathbf{X})\mathbf{K}(\mathbf{X}, \mathbf{X})^{-1}\mathbf{K}(\mathbf{X}, \mathbf{X}_*) \end{aligned} \tag{2.17}$$

where $\mathbf{K}(\mathbf{X}, \mathbf{X}_*)$ is the covariance matrix between training and test points, and $\mathbf{K}(\mathbf{X}_*, \mathbf{X}_*)$ is the covariance matrix between test points. $\mathbf{K}(\mathbf{X}, \mathbf{X}_*) = \mathbf{K}(\mathbf{X}_*, \mathbf{X})^T$.

With both terms are Gaussians, the predictive posterior is another Gaussian and is given by

$$\begin{aligned} p(\mathbf{y}_*|\mathbf{X}_*, \mathbf{X}, \mathbf{y}) &\sim \mathcal{N}(\mathbf{y}_*|\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*) \\ \boldsymbol{\mu}_* &= \mathbf{K}(\mathbf{X}_*, \mathbf{X})[\mathbf{K}(\mathbf{X}, \mathbf{X}) + \sigma_N^2\mathbf{I}]^{-1}\mathbf{y} \\ \boldsymbol{\Sigma}_* &= \mathbf{K}(\mathbf{X}_*, \mathbf{X}_*) - \mathbf{K}(\mathbf{X}_*, \mathbf{X})[\mathbf{K}(\mathbf{X}, \mathbf{X}) + \sigma_N^2\mathbf{I}]^{-1}\mathbf{K}(\mathbf{X}, \mathbf{X}_*) \end{aligned} \tag{2.18}$$

or in the case of a single test point y_*

$$\begin{aligned} p(y_*|\mathbf{x}_*, \mathbf{X}, \mathbf{y}) &\sim \mathcal{N}(y_*|\mu_*, \sigma_*^2) \\ \mu_* &= \mathbf{k}(\mathbf{x}_*, \mathbf{X})[\mathbf{K}(\mathbf{X}, \mathbf{X}) + \sigma_N^2\mathbf{I}]^{-1}\mathbf{y} \\ \sigma_*^2 &= k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}(\mathbf{x}_*, \mathbf{X})[\mathbf{K}(\mathbf{X}, \mathbf{X}) + \sigma_N^2\mathbf{I}]^{-1}\mathbf{k}(\mathbf{X}, \mathbf{x}_*) \end{aligned} \tag{2.19}$$

$\mathbf{k}(\mathbf{X}, \mathbf{x}_*)$ denotes a $T \times 1$ covariance vector between training points $\mathbf{X}$ and a single test point $\mathbf{x}_*$ with $\mathbf{k}(\mathbf{X}, \mathbf{x}_*) = \mathbf{k}(\mathbf{x}_*, \mathbf{X})^T$.

Equation 2.18 is the necessary equation for making predictions using Gaussian process regression. At this stage, it should be clear why a Gaussian process regression is considered a non-parametric regression model and how it differs from the earlier linear regression. In GP regression, we infer the function as the 'parameters' of the model. It can be seen from Equation 2.15 that the function depends on the training input $\mathbf{X}$, hence what we have is parameters that grow with the size of the training data. This characteristic is a contrast to the linear regression where the parameter size is fixed. If we examine Equation 2.18, then we can see that the GP regression also needs all the training data to make predictions of future data. Because of this, we can utilise the full information that is available in the training data to predict new data without having to be restricted by the finite set of parameter like in the parametric model. This shows that the GP regression can be a more capable regression model than the parametric one. In this section, we also have a glimpse on how the GP regression can model diverse functions by just defining a covariance function. More detail about covariance function and several kinds of covariance functions will be discussed in the next section.

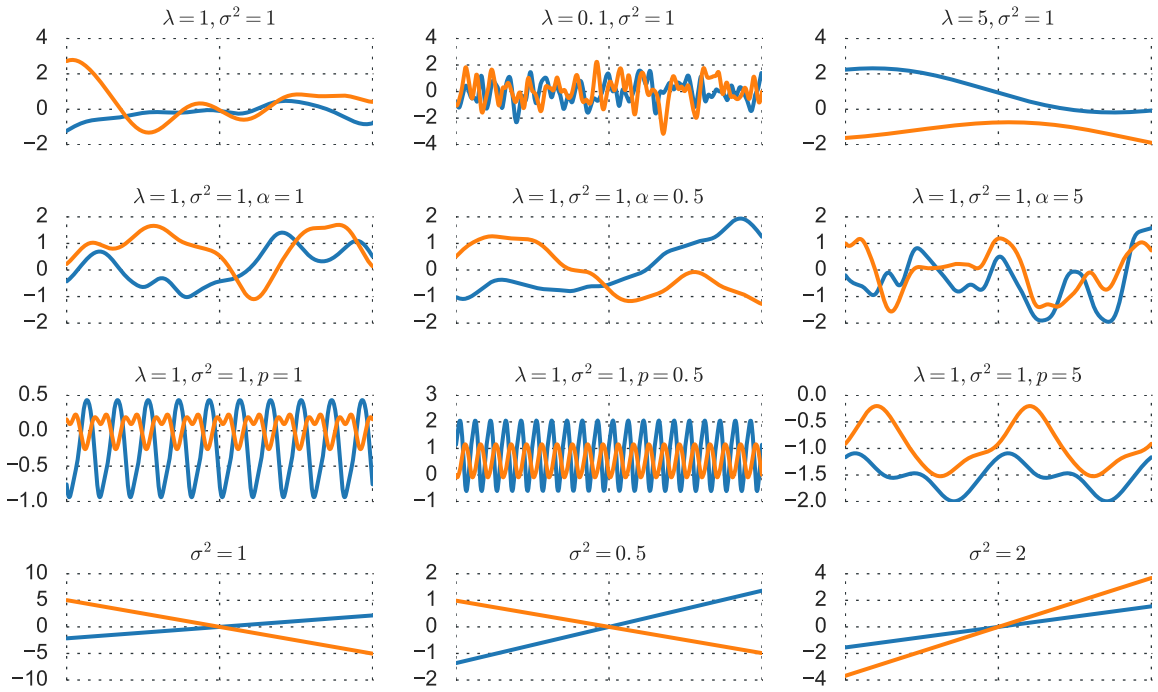


Figure 2.2: Random samples from several covariance functions. First row: RBF, second row: RQ, third row: Periodic, fourth row: Linear. Each column denotes different variation of parameters.

2.3. Covariance Function (Kernel)

Covariance function measures the nearness or similarity between two points. A covariance function states that if two points x and x' are similar, then $f(x)$ and $f(x')$ should also be similar. Covariance function encodes our initial belief over the function that we want to regress. This initial belief could be how smooth our function is or whether the function is periodic. The covariance function is also known as the *kernel* since it measures a degree of similarity between points in the dataset. This report will write both covariance function and kernel interchangeably. Any function could be a covariance function as long as the resulting covariance matrix is *positive semi-definite*.

2.3.1. Common Covariance Functions

Four standard covariance functions will be discussed in this section. Those are the radial basis function (RBF), rational quadratic (RQ), periodic, and linear kernel. For the discussion of the kernel in this section, it is assumed that the input is a scalar instead of a vector.

Radial Basis Function (RBF)

A radial basis function (RBF) kernel, also commonly known as the squared exponential kernel, is given by

$$k_{\text{RBF}}(x, x') = \sigma^2 \exp\left(-\frac{(x - x')^2}{2\lambda^2}\right) \tag{2.20}$$

An RBF kernel has two parameters, λ and σ^2. The former is known as the lengthscale parameter, which controls the horizontal scale over which the function changes and the latter is the noise variance, which controls the vertical scale changes [36]. One prominent characteristic of an RBF kernel is the smoothness assumption. RBF assumes that the underlying function is smooth and infinitely differentiable. We should be aware of this because the assumption might not hold for our data. If our function is rough,

the Rational Quadratic (RQ) kernel can be chosen instead [56]. The RQ kernel will be discussed after this section.

The first row of Figure 2.2 shows how the lengthscale parameter λ affects the sampled function from an RBF kernel. An RBF covariance function value falls within 0 and 1, $k_{\text{RBF}}(x, x') = [0, 1]$. RBF defines similarity as closeness, where the closer the point x with x', $k(x, x') \to 1$. In contrast, when the difference $|x - x'|$ becomes higher, the value decays to zero. The rate of decay is controlled by the λ parameter.

A small lengthscale causes a more rapidly changing, 'wiggly' looking function. Small lengthscale makes the covariance decays to zero very fast as the distance between points goes higher, which explains the rapid variation changes. Conversely, a large lengthscale causes a slow change, which in effect creates a very smooth function. The lengthscale parameter plays an important role on our assumption about the function would be. To model a rapidly changing function, small lengthscale RBF kernel should be used. In contrast, to extrapolate a value that is far from the training data, a large lengthscale should be chosen instead.

Rational Quadratic (RQ)

A rational quadratic (RQ) kernel is given by

$$k_{\text{RQ}}(x, x') = \sigma^2 \exp\left(1 + \frac{(x - x')^2}{2\alpha\lambda^2}\right)^{-\alpha} \tag{2.21}$$

An RQ kernel can be seen as an infinite sum of RBF kernels with multiple lengthscales [44]. An RQ kernel does not assume that the function f is smooth, unlike the RBF kernel. Because of that, the RQ kernel is more appropriate to model a non-smooth, rough function. In addition to the λ and σ parameter, an RQ kernel has another parameter, the power parameter α, which defines how quick the local variation is.

The second row of Figure 2.2 shows randomly sampled function from RQ kernels with different α. It can be noted from the Figure 2.2 in the first column of the second row that even with the same λ and σ, the function shows local variations inside a larger variation. This effect can be seen as a combination of multiple RBF kernels with different lengthscales interacting with each other. The local variation is controlled by the α, while the long-term variation is by λ. The larger the α, the more rapid the local variation is.

Periodic

A periodic covariance function is defined as follows

$$k_{\text{Per}}(x, x') = \sigma^2 \exp\left(-\frac{\sin^2(\pi(x - x')/p)}{2\lambda^2}\right) \tag{2.22}$$

A periodic kernel gives a repeating structure over the function which is controlled by the period parameter p. The lengthscale λ and variance σ^2 have the same effect that is found in an RBF kernel. Larger p causes a slower oscillation while smaller p causes a higher oscillation. The second row of Figure 2.2 illustrates the effect of p on the function. It can also be seen from the figures that a periodic kernel cause an exact repeating pattern. This aspect is important to note since repeating patterns that occur in real world data usually do not have precise oscillations.

Linear

The linear covariance function is defined as follows

$$k_{\text{Lin}}(x, x') = \sigma^2 x x' \tag{2.23}$$

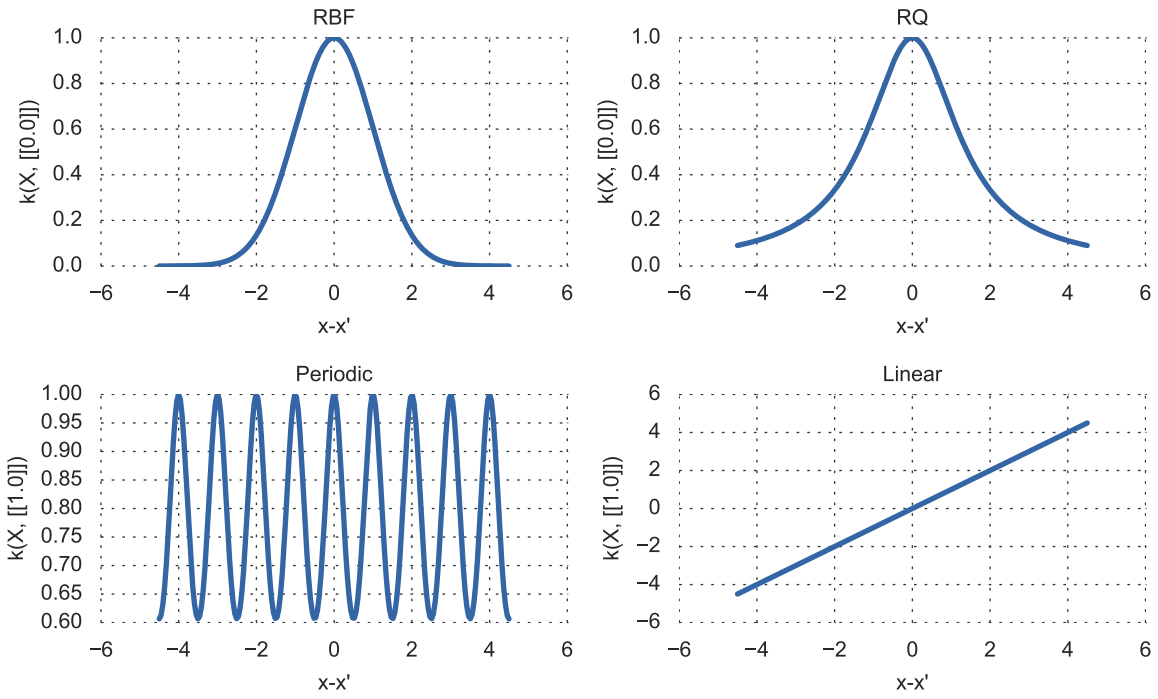


Figure 2.3: The plot of $x - x'$ against the covariance function k. It can be seen that both RBF and RQ kernel are local kernels.

It is a very simple kernel, where the only parameter is the noise variance σ^2 which controls the vertical lengthscale of the function. Different σ affects the scale on the output axis, where the larger the σ, the larger vertical scale will be.

2.3.2. Kernel Composition

It is possible to combine several covariance functions together to model a more complex function. The only thing to remember is that the resulting covariance matrix must be a positive semi-definite matrix. The two operations below, addition and multiplication, are two ways to combine covariance functions together while keeping the positive semi-definite property intact. These operations are critical in Chapter 3.

$$k(x, x') = k_1(x, x') + k_2(x, x') \tag{2.24}$$

$$k(x, x') = k_1(x, x')k_2(x, x') \tag{2.25}$$

2.3.3. Stationary and Local Kernel

One important characteristic that RBF, RQ, and periodic kernels share in common is stationarity. Stationary means that a kernel has a translation invariant property, where the function value only depends on the difference between data points, not the data points themselves. A pair of $x = 1, x' = 2$ has the same covariance value as a pair of $x = 100, x' = 101$, for example. In contrast, a linear kernel does not have translation invariant property, and thus it can be classified as a non-stationary kernel.

Another important aspect of the kernel that needs to be understood is the locality of the kernel. Figure 2.3 shows the plot of $k(x, 0)$ for various covariance kernels. These plots show how the similarity or covariance between data points are computed by the kernel. The RBF and RQ kernel exhibit strong covariance between similar data points ($|x - x'| \to 0$) but as the difference goes larger, the covariance decays to zero. This is an indicator of a local kernel, where it shows high similarity only to nearby

points. In contrast to the RBF and RQ kernel, both linear and periodic kernel do not display a locality characteristic. Locality characteristic of a kernel is important when we want to extrapolate because we regress to a region where no data point is available when we extrapolate. Therefore, we want to have a GP model with a non-local kernel. Otherwise, all the extrapolated data points will have zero similarities with other data points and consequently, our GP model will be dominated by the mean function only.

2.4. Training a GP Regression Model

Training or learning a GP regression model means finding the optimum parameter of the covariance functions. Choosing the parameter can be done manually if we have a sufficient domain knowledge about our problem and our covariance function. For example, we can refer to Figure 2.2 to understand the effect of the parameter λ on the RBF kernel. This manual approach will not work for most cases. Therefore, we need a systematic approach to parameter selection. In GP, this systematic approach is done by maximising the evidence or the marginal likelihood of a GP model [44]. The explanation for this is available in detail in Appendix B. In short, given that the parameters of a kernel are stored in the parameter vector $\boldsymbol{\Theta}$, the optimum parameter $\boldsymbol{\Theta}_*$ is given by maximising the log-marginal likelihood w.r.t to $\boldsymbol{\Theta}$

$$\boldsymbol{\Theta}_* = \arg\max_{\boldsymbol{\Theta}} \quad -\frac{1}{2}\mathbf{y}^T[\mathbf{K}(\mathbf{X},\mathbf{X};\boldsymbol{\Theta}) + \sigma_n^2\mathbf{I}]^{-1}\mathbf{y} - \frac{1}{2}\log\det[\mathbf{K}(\mathbf{X},\mathbf{X};\boldsymbol{\Theta}) + \sigma_n^2\mathbf{I}] - \frac{n}{2}\log 2\pi \tag{2.26}$$

Here we make the covariance matrix $\mathbf{K}$ explicitly depends on the kernel parameter $\boldsymbol{\Theta}$.

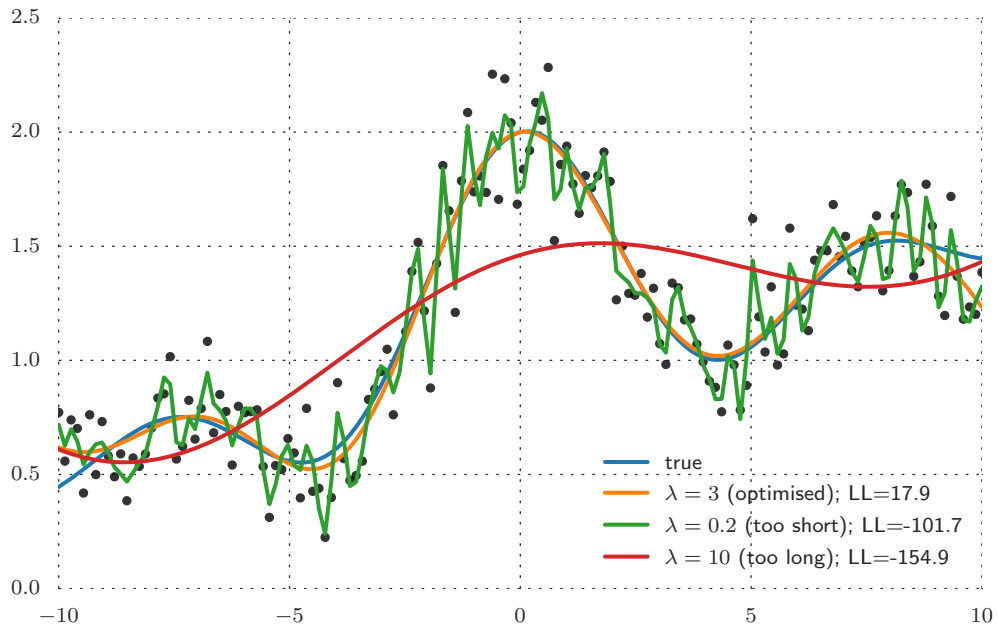


Figure 2.4: A GP regression model is fitted on the noisy data (shown by dots) to model the true function which is shown by the blue curve. The marginal likelihood favours the model with the orange curve, which closely resembles the true function. The parameter of this model is given by an RBF kernel with $\lambda = 3, \sigma = 1$. If we shorten the λ (green curve), the model overfits the data while the model with long λ (red curve) underfits the data. Both models are assigned with lower marginal likelihood values (LL).

Maximising the marginal likelihood provides a way to choose the right model GP regression while protecting against overfitting [44]. Figure 2.4 depicts an example of how the marginal likelihood selects the right model. This example illustrates a regression problem which is fitted using a Gaussian Process

regression with an RBF kernel, $\mathbf{\Theta} = [\lambda, \sigma]^T$. The true function is depicted by the blue curve. We wish to fit a GP regression on the noisy data which are shown by the dots such that the regression approximate the true function. The best approximating fit is given by an RBF kernel with $\lambda = 3, \sigma = 1$, which is demonstrated by the orange curve. The marginal likelihood favours this model, as it gets the highest marginal likelihood value. Now, if we keep the σ fixed, but shorten the λ to 0.2, we obtain the fit that is shown by the green curve. This model overfits the data, as it models the noise as opposed to the true function. This model gets a lower marginal likelihood value. Similarly, the model with longer lengthscale, which is shown by the red curve, is set with a low marginal likelihood, as it underfits. This example indicates that marginal likelihood maximisation is robust to overfitting. This phenomenon is explained in Appendix B.

The first term of the marginal likelihood is the main computation bottleneck because it involves an inversion of a matrix. To simplify the notation, let $\mathbf{C} = \mathbf{K}(\mathbf{X}, \mathbf{X}; \mathbf{\Theta}) + \sigma_n^2 \mathbf{I}$. In practice, Cholesky decomposition is performed instead of directly inverting the $\mathbf{C}$ matrix, because $\mathbf{C}$ is a positive semi-definite matrix

$$\begin{aligned} \mathbf{C} &= \mathbf{L}\mathbf{L}^T \\ \mathbf{C}^{-1} &= (\mathbf{L}^{-1})^T \mathbf{L}^{-1} \end{aligned}$$

and the first term of the log-likelihood equation can be simplified to

$$\begin{aligned} \frac{1}{2}\mathbf{y}^T \mathbf{C}^{-1} \mathbf{y} &= \frac{1}{2}\mathbf{y}^T (\mathbf{L}^{-1})^T \mathbf{L}^{-1} y \\ &= \frac{1}{2}\mathbf{y}^T (\mathbf{L}^T)^{-1} (\mathbf{L}^{-1}\mathbf{y}) \end{aligned}$$

where we can solve the $(\mathbf{L}^T)^{-1}(\mathbf{L}^{-1}\mathbf{y})$ using the forward substitution instead of inverse because $\mathbf{L}$ is a lower triangular matrix

$$\frac{1}{2}\mathbf{y}^T \mathbf{C}^{-1} \mathbf{y} = \frac{1}{2}\mathbf{y}^T \mathbf{L}^T \setminus (\mathbf{L} \setminus \mathbf{y})$$

The second term of the log-likelihood equation can also be simplified knowing the fact that the determinant of a triangular matrix is a product of its diagonal elements, so

$$\begin{aligned} \frac{1}{2}\log\det \mathbf{C} &= \log\det \mathbf{L}\mathbf{L}^T \\ &= \log\det \mathbf{L} \det \mathbf{L}^T \\ &= \log \prod_{i=1}^{n} \mathbf{L}_{ii}^2 \end{aligned}$$

The log-likelihood function now becomes

$$\log p(\mathbf{y}|\mathbf{X}) = -\frac{1}{2}\mathbf{y}^T \mathbf{L}^T \setminus (\mathbf{L} \setminus \mathbf{y}) - \frac{1}{2}\log \prod_{i=1}^{n} \mathbf{L}_{ii}^2 - \frac{n}{2}\log 2\pi \tag{2.27}$$

The cost of computing the log-marginal likelihood is $\mathcal{O}(n^3)$ because of the Cholesky decomposition operation, therefore training a Gaussian process regression has the complexity of $\mathcal{O}(n^3)$.

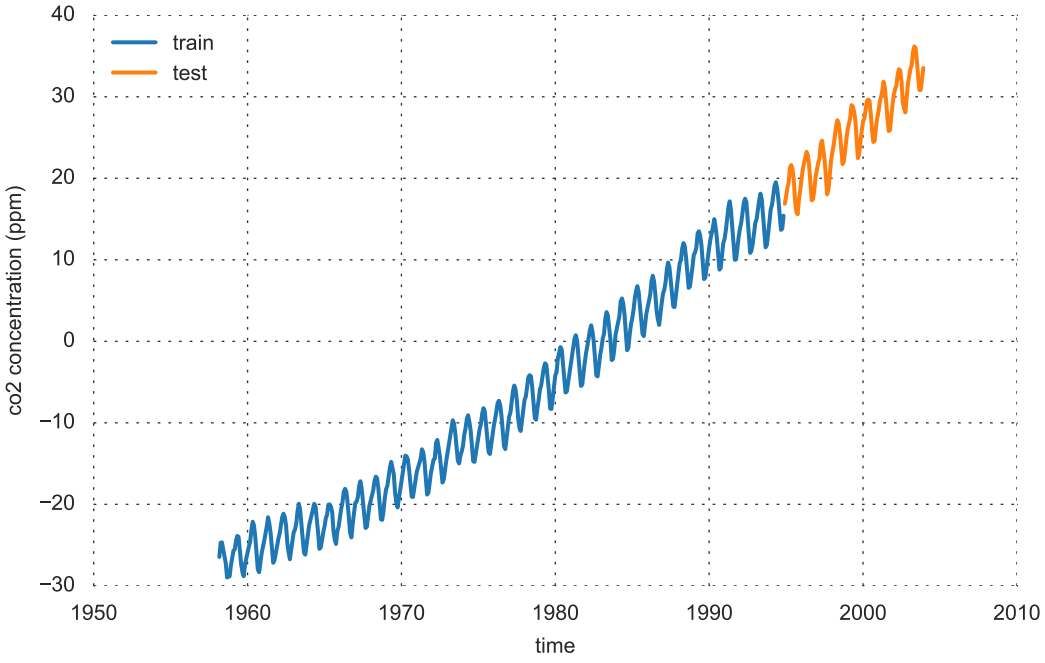


Figure 2.5: CO_2 concentration (in ppm) recorded at the Mauna Loa observatory, USA, from March 1958 until December 2001

2.5. Forecasting the CO_2 Data

In this section, we perform a time series forecasting using a simple Gaussian Process regression model. The time series data that is utilised for this purpose is the Mauna Loa atmospheric CO_2 data, which is the CO_2 concentration (in ppm) recorded at the Mauna Loa Observatory, USA, from March 1958 until December 2001. The first 80% of the data will be used for training and last 20% of the data will be utilised for testing, which is 109 horizons ahead in the future. Figure 2.5 shows the plot of the dataset.

We take advantage of the similar regression-in-time setting that is found on Section 1.4.3, but using the GP regression instead

$$\mathbf{x}_t = t$$
$$f(\mathbf{x}_t) \sim \mathcal{GP}(0, k_{\text{RBF}}(\mathbf{x}_t, \mathbf{x}'_t))$$

Forecasts up to horizon h, $\mathbf{y}_* = [y_{T+1}, y_{T+2}, \dots, y_{T+h}]^T$, are calculated using Equation 2.18, where the test input is a vector of time index $\mathbf{X}_* = [T+1, T+2, \dots, T+h]^T$. The kernel parameter is learned by maximising the marginal likelihood from Section 2.4.

The result is presented in Figure 2.6. It can be seen that the GP regression model is only able to forecast correctly up to the tenth horizon. After that, the forecast starts to converge to the mean of the function which is zero. The reason for this finding is caused by the local kernel, which has been discussed in Section 2.3. As we have already known, the RBF kernel is a local kernel, where strong covariance is assigned to the nearby points. As a result, the covariance starts to vanish until it becomes zero as the point goes farther apart. In this problem, we utilise the time index as input, so as $|t_T - t_{T+h}|$ goes larger, $k_{\text{RBF}} \to 0$. That is the reason why the forecast result converge to the mean for greater horizons, because when the covariance becomes insignificant, the mean function will dominate the prediction.

This result clearly shows that the simple GP model is only acceptable for short-term forecasting, just like the ARIMA model that is discussed earlier in Section 1.4.4. For forecasting to higher horizons,

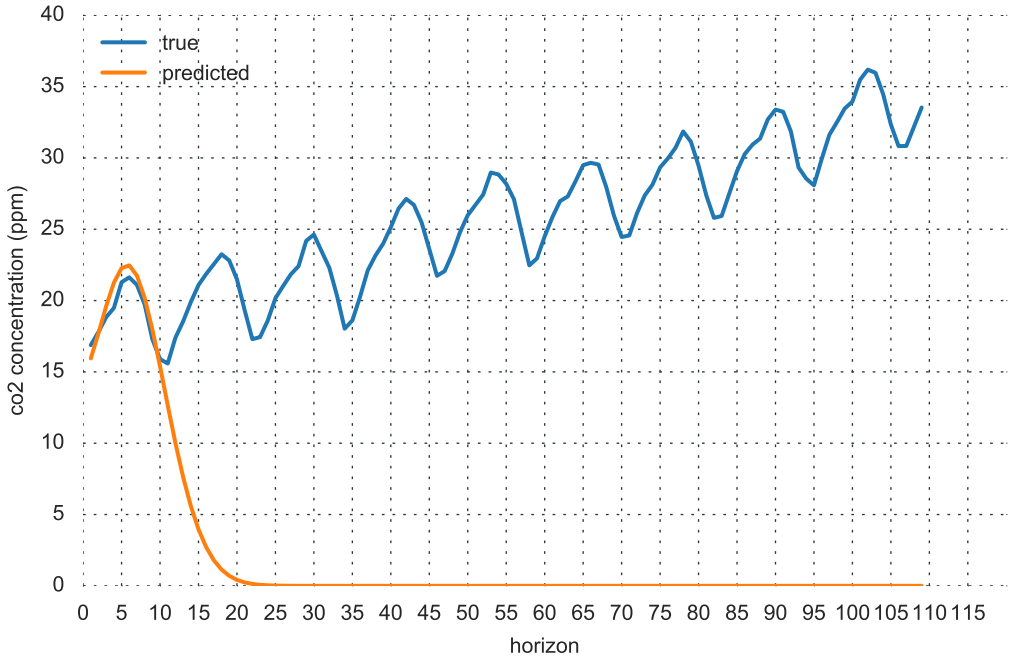


Figure 2.6: The result demonstrates that the forecasts only fairly accurate up to the tenth horizon, after that the forecasts start to converge to the mean of the GP which is zero

a better forecasting method is needed to utilise GP as a long-term time series forecasting technique. In Section 1.4.5, two approaches are discussed, first is the *autoregressive* approach and second is the *structure modelling* approach. The first approach involves the composition of a complex kernel from several base kernels through addition and multiplication. The second approach engineers the time series into a multi-dimensional input where those inputs are the autoregressive values of the time series. The next two chapters will discuss each of these techniques in details.

3

Structure Modelling

Chapter 2 has briefly outlined the theory behind Gaussian processes (GP) and how the GP can be used for regression. The last section of Chapter 2 indicates that a simple GP regression model with a single RBF kernel is not sufficient for long-term time series forecasting problem. The reason is the locality property of the RBF kernel causes the forecasting model to suffer the same mean convergence problem of the ARIMA model.

The same regression-in-time setting from Section 2.5 is employed for this approach. To enhance the model, a more complex covariance function $\hat{k}(t, t')$ is utilised

$$\mathbf{x}_t = t$$
$$f(t) \sim \mathcal{GP}(0, \hat{k}(t, t'))$$

where the covariance function $\hat{k}(t, t')$ is a combination of kernels by adding and multiplying them together. For example, the covariance function could have the form of

$$\hat{k}(t, t') = k_1(t, t') + k_2(t, t') \times k_3(t, t') + \ldots \tag{3.1}$$

where $k_i(t, t'), i = 1, 2, \ldots$ is either a radial basis function (RBF), linear, and periodic covariance function which have been discussed in Section 2.3. These three kernels are the *base* kernels that make up our complex kernel $\hat{k}$. In this report, these kernels are sometimes abbreviated as RBF, Lin, and Per kernel, respectively.

The idea behind this complex kernel composition is that each covariance function is responsible for modelling a certain *structure* or *pattern* that exists in the time series. Hyndman and Athanasopoulos state that the key to a successful forecast is to model and extrapolate the underlying pattern that exists in the past data and ignore the random fluctuation that occurs [28]. By recovering those core structures from our noisy time series observations, we can extrapolate those patterns far ahead into the future to make a long-term forecast.

So the question now is what patterns that we should be interested in. The *additive model* states that a time series can be assembled from *trend*, *season*, and *residual* components through addition

$$y_t = t_t + s_t + e_t \tag{3.2}$$

with t_t, s_t, and e_t denote the trend, season, and residual components respectively. A time series contains a trend when it exhibits a downward or upward movement over time. The trend does not have to be linear; it can be quadratic or another non-linear trend. A seasonal pattern is a repeating wave-like

pattern, where the frequency of the oscillation is exact. The residual is the component of the time series that cannot be explained by the trend and season.

A Gaussian process regression is an excellent choice to implement the additive model because we can model the trend and season using the covariance function. A linear trend can be modelled exactly with the linear kernel while the season can be modelled with the periodic kernel. In this case, a time series with a linear trend and a seasonal pattern can be modelled with a GP regression using the following kernels

$$\hat{k}(t,t') = k_{\text{Lin}}(t,t') + k_{\text{Per}}(t,t') \tag{3.3}$$

The above example is very basic, and we can do a more elaborate modelling by taking advantage of a more sophisticated combination of kernels to model various kind of patterns. By multiplying two linear covariance functions, for example, we can model a quadratic trend. If we have a seasonal pattern which does not have an exact repetition, then adding an RBF kernel should model this pattern sufficiently. The covariance function of the GP regression grows into

$$\hat{k}(t,t') = k_{\text{Lin}}(t,t') \times k_{\text{Lin}}(t,t') + k_{\text{Per}}(t,t') + k_{\text{RBF}}(t,t') \tag{3.4}$$

As we will see in Section 3.1, the base kernels can be added and multiplied to model a complex pattern of time series, and the possible combinations are limitless as we can keep adding and multiplying the kernels. This idea drives the first proposed approach for long-term time series forecasting, which is called the *structure modelling*. The idea of combining covariance functions to model complex structures in a time series is suggested by Rasmussen and Williams [44, p.118]. They do it by visually inspecting the time series, then model the kernel by hand based on the pattern that they find. Clearly, the problem with their approach is that the procedure is tedious which causes Duvenaud et al. to propose an automated mechanism to search the best combination of kernels [16]. The next two sections will discuss two main components that are central to the *structure modelling* approach. One, what kind of kernel combinations are possible to model various structures that exist in a time series. Two, how can we automate the kernel modelling without having to do it manually.

3.1. Expressing Structure Through Kernels

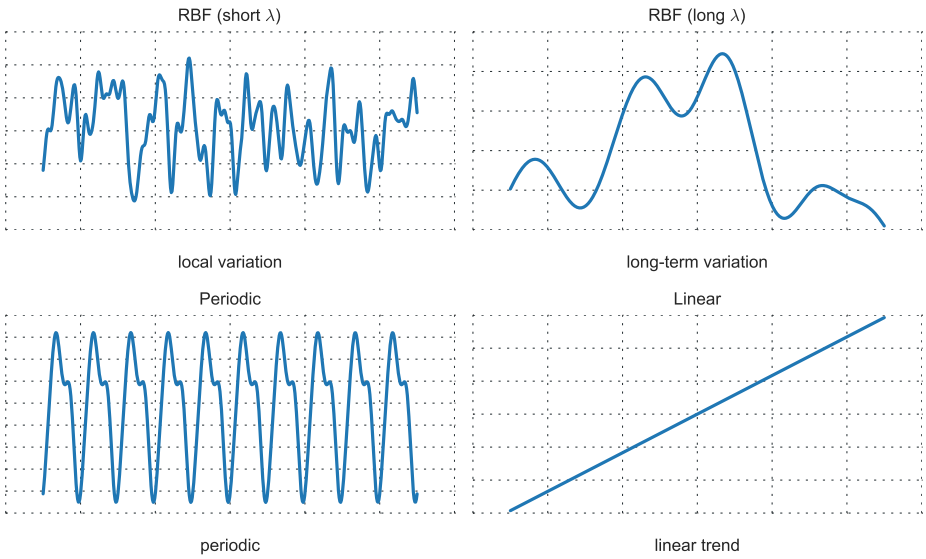


Figure 3.1: How the RBF, periodic, and linear kernel model several basic structures of time series. These three core covariance functions serve as a building block of a more complex kernel

Earlier in this chapter, it was mentioned that the *RBF*, *periodic*, and *linear* kernel are used as the *base* kernels. The rational quadratic kernel is not used as one of the base kernels because it can be regarded as a combination of multiple RBF kernels, hence redundant. Figure 3.1 shows how RBF, periodic, and linear kernels can model several basic structures of time series. An RBF kernel with a short lengthscale is capable of modelling a short-lived variation while an RBF with long lengthscale is suitable for longer-term variation. A periodic and a linear kernel have a straightforward effect, with the former models a periodic structure with an exact frequency and the latter models a linear trend.

The base covariance functions from Figure 3.1 are useful on their own, but as Duvenaud et al. show in their research, we could model a more detailed structure of time series by combining several kernels together through addition and multiplication [16]. Combining covariance functions can be understood intuitively, as addition and multiplication create a distinct effect. In the addition process, the characteristic of two added kernels is retained, and both characteristics are strongly apparent in the combined structure. In other words, the combined structure shows an *independent* work of each base kernel. In the multiplication process, each base kernel characteristic is *fused* instead of working independently. Unlike additive, the two base kernels merge their traits instead of showing off their strong influence on the combined effect.

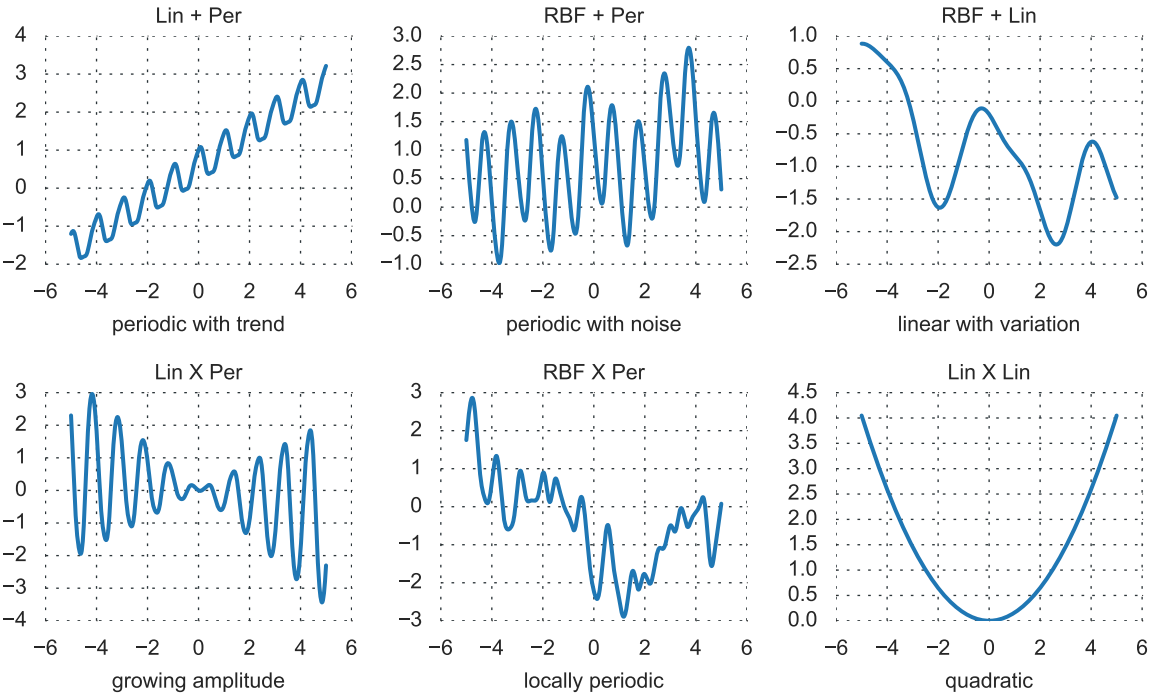


Figure 3.2: Combined kernel through addition and multiplication creates different structures. First row 1: additive. Second row: multiplicative.

Figure 3.2 displays how several different structures can be modelled by combining two base kernels through multiplication and addition. Notice the difference between the effect of addition (row 1) and multiplication (row 2), where the former gives an independent effect while the latter gives a fused effect of the base kernel structure. Let us consider the first column of Figure 3.2, which shows the difference between adding and multiplying a linear and periodic kernel. On the one hand, the $\mathrm{Lin} + \mathrm{Per}$ kernel exhibits both trend and periodic structures, which in effect creates a periodic structure which is increasing or decreasing. We could see that the linear trend and the periodic are evident and working independently. The linear does not affect the periodicity and vice versa. On the other hand, the $\mathrm{Lin} \times \mathrm{Per}$ fuses an increasing or decreasing effect to a periodic structure, which creates a growing amplitude. It can be seen from the figure that the periodic effect is affected by the linear growth of the linear kernel, and thus we have a periodic signal which amplitude is growing.

With many possible combinations of kernels, we can express the different complex structure of time

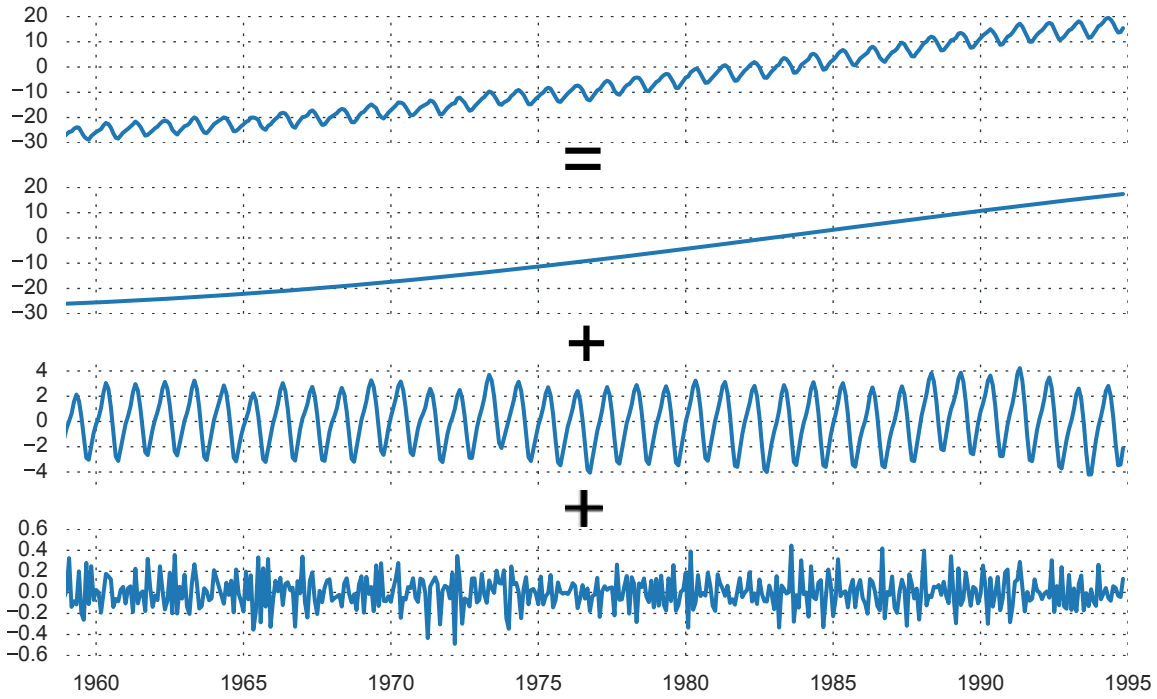


Figure 3.3: Decomposition of the CO_2 data reveals two dominant structures: a non-linear increasing trend and periodic with varying amplitude. The bottom row is the residual, which is much smaller in value in comparison with the other signal.

series. If we go back to our CO_2 dataset that has been introduced in Section 2.5, we can manually design a complex kernel based on our inspection of the underlying structure that is illustrated in Figure 3.3. We see that the additive decomposition of the CO_2 time series consists of an increasing trend, varying-amplitude periodic signal, and random variation as residual. Based on this decomposition, we could model the kernel as follows

$$\underbrace{\mathrm{RBF} \times \mathrm{Lin}}_{\text{increasing trend}} + \underbrace{\mathrm{RBF} \times \mathrm{Per}}_{\text{varying-amplitude periodic}} + \underbrace{\mathrm{RBF}}_{\text{residual}}$$

Designing kernel by hand might work if you only have a handful of time series to work on. But if you have several datasets or if you do an online forecasting where the data are updated gradually over time, then the manual method will be tedious and will not scale so well. It will be more fitting if there is a technique to search for all combination of kernel efficiently without having to enumerate all possible combinations in the search space. This search method will be the main discussion point of the next section.

3.2. Searching for the Optimum Kernel Combination

Automating the kernel composition is equal to a search problem. We want to get the *best* kernel combination out of all possible combinations by some *metrics*. Now the first question that we want to answer is how large is the search space of possible kernels. One way to build the space of possible covariance functions combinations is to devise a search tree. Figure 3.4 shows how the kernel space is represented as a tree structure.

3.2.1. Building the Search Tree

We start with the base kernel for the top level structure of the tree, which is the RBF, Per, and Lin kernel. Then for the second level of the tree, we start adding or multiplying the base kernel with another

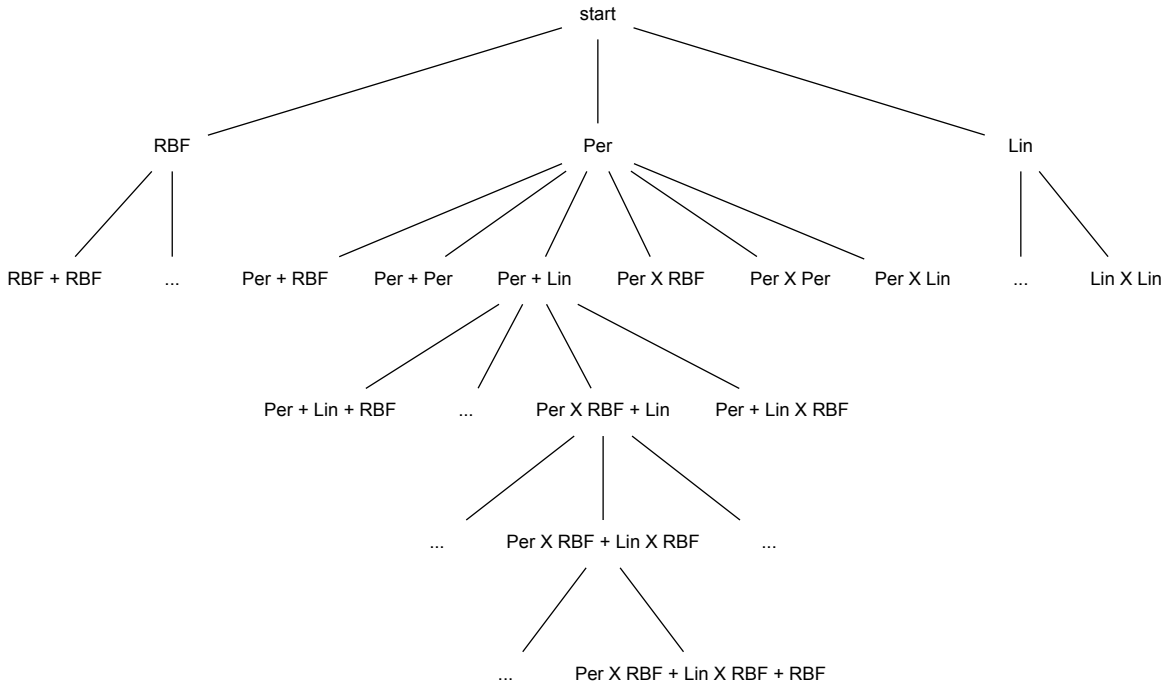


Figure 3.4: The kernel search space which is represented by a tree structure

base kernel to create a kernel combination consists of two kernels. So, from the Per parent, we have $\mathrm{Per} + \mathrm{RBF}, \mathrm{Per} + \mathrm{Per}, \mathrm{Per} + \mathrm{Lin}, \mathrm{Per} \times \mathrm{RBF}, \mathrm{Per} \times \mathrm{Per}, \mathrm{Per} \times \mathrm{Lin}$ branches. Similar branches are found for the RBF and Lin kernel as parents. The process is followed by adding or multiplying a base kernel to the already established kernel combinations from the previous level. From the $\mathrm{RBF} + \mathrm{RBF}$ branch for example, we find branches of $\mathrm{RBF} + \mathrm{RBF} + \mathrm{RBF}$, $\mathrm{RBF} + \mathrm{RBF} + \mathrm{Per}$ and so on.

In general, there will be $n \times (2n)^{d-1}$ number of nodes, where n is the number of base kernel and d is the maximum level or depth of the tree. By this calculation, a three-level, four-level, and five-level tree will have 108, 648, and 3888 number of nodes. We can see that the number of nodes grows exponentially with the number of search level. The level or depth of the tree depends on the complexity of the time series.

Each node of the tree represents one possible kernel combination. Alternatively, a node in the search tree corresponds to a single *model* M. In this case, our problem of finding the best kernel combination is equal to a *model selection* problem, and that is finding the best M from a set of models $\mathcal{M} = \{M_1, \dots, M_k\}$. To properly search the tree, three essential elements must be considered; those are the *search strategy*, the *search metric*, and the *stopping criterion*. Search strategy defines method or algorithm to select the nodes of the tree that qualify as the candidate model. In other words, the search strategy determines the element of the model set $\mathcal{M}$. Search metric is needed to quantitatively evaluate every model M and select the best one in the set of models $\mathcal{M}$. The stopping criterion decides whether the search should continue to next search level or stop.

3.2.2. Search Strategy

To search for the candidate model, we consider three search strategies: the exhaustive, random, and greedy search strategy. The *exhaustive search* finds all nodes of the search tree as a candidate model. As a consequence, the size of $\mathcal{M}$ can become enormous, especially when the tree level is deep. If we train a GP regression in every evaluation of the model, then the search will be very expensive. As it was mentioned in Section 2.4, training a GP regression requires an $\mathcal{O}(n^3)$ computing complexity, and thus the model selection will be costly for a large tree. A better strategy is indeed required.

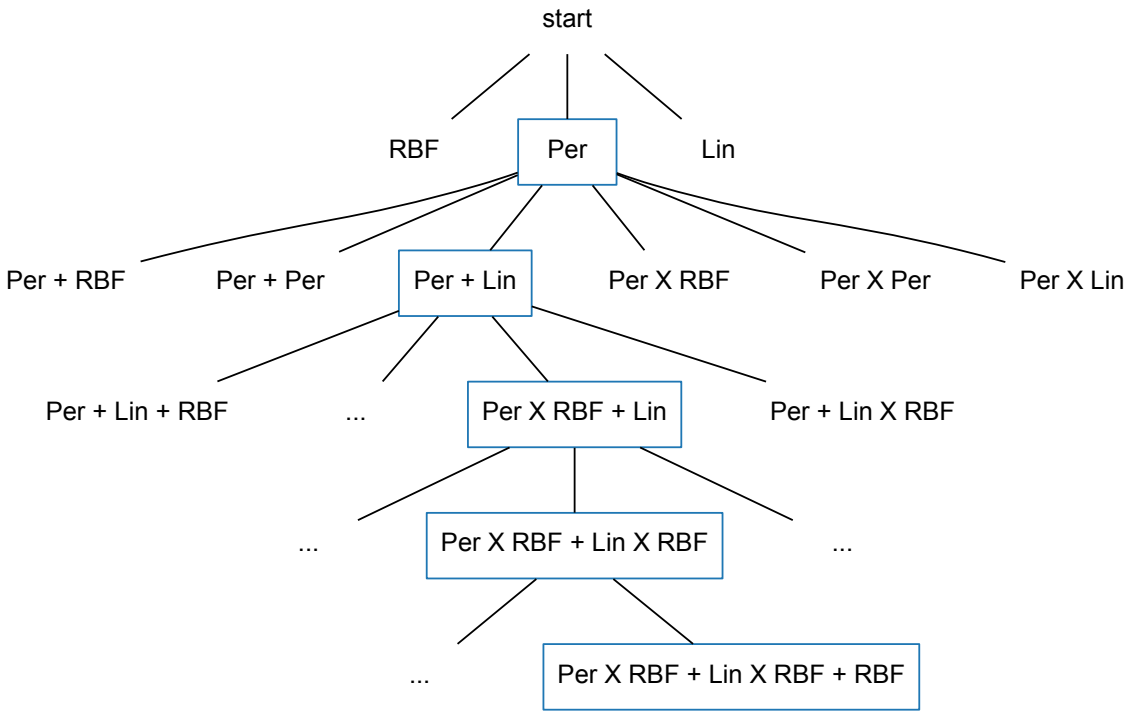


Figure 3.5: A diagram depicting the greedy search tree. The greedy search selects a local optimum kernel in every level, which is denoted by the blue rectangle. All other non-local optimum branches are pruned.

As the complete opposite of the exhaustive search, we have the *random search* strategy. Instead of enumerating all the nodes in the tree, this method samples n nodes randomly, so we are given only n models that are needed to be evaluated. Clearly, this approach does not have any optimality guarantee. As the search depth increases, the reliability of this solution will significantly decrease, since the size of $\mathcal{M}$ increases, thus the probability of getting the optimum model from the model set, if any, will be much lower.

We have discussed two different approaches for searching, those are the exhaustive search and the random search, where the exhaustive search is expensive to compute while the random search is unreliable. It is more sensible if we aim to find a middle ground over these two contradictory strategies. This approach should explore a large number of combinations with a reasonable computing resource, even if not all nodes are explored. This proposal becomes the motivation of the *greedy search* strategy, which is proposed by Duvenaud et al. [16].

The idea of *greedy search* is straightforward: it does not search for everything. Instead, it finds a sensible heuristic to select some candidate models. This approach is suboptimal because there is no guarantee that the chosen candidates will lead to the optimum solution. In the feature selection problem, for example, we might be familiar with the forward search heuristic. The forward search greedily adds a single feature at a time to the feature vector, then evaluates according to an evaluation metric and whichever feature gives the best result will be kept in the feature vector. This similar concept is adopted to the kernel search problem.

The method starts with evaluating a single kernel out of all base kernels. The base kernel that returns the best score will be chosen. In the next step, the method starts adding and multiplying another base kernel to the *previously selected kernel only*. Notice the important difference with the exhaustive search that was discussed before. The greedy search evaluates local optimum choice in every iteration. In the search tree context, the greedy search selects only a single branch to continue the search iteration and disregards every other branch. This heuristic effectively reduces the number of candidates to evaluate as opposed to the exhaustive search, hence reduces the needed computation time. The number of candidates from a greedy search is now $n \times (2n)$ instead of $n \times (2n)^{d-1}$ from an exhaustive search.

Figure 3.5 shows how the search tree of greedy search is modified from the exhaustive search. Here, it can be seen that the greedy search selects a local optimum node for every iteration, which is denoted by blue rectangles. All non-local optimum branches are pruned and will be ignored for the subsequence iterations. This mechanism makes the greedy search much more efficient than the exhaustive search. One drawback of the greedy search, which also applies to other suboptimal heuristics, is that there is no guarantee that the search will give rise to the optimum result because it might be that the optimum solution lies within the pruned branches. Even so, the computational complexity of the greedy technique is much lower than the exhaustive search. Because of that, the trade-off could be worth it, especially considering the cubic complexity of training a Gaussian Process model.

3.2.3. Search Metric

Model M_i is chosen over M_j if M_i is better than M_j according to a search metric, or $g(M_i) > g(M_j)$, where g is a search metric function. In this section, two search metrics are discussed, the Bayesian information criterion (BIC) and cross-validation.

Bayesian Information Criterion

A Bayesian solution to model selection is to compute the posterior probability of a model given our training data

$$g(M_i) = p(M_i|\mathcal{D}) \tag{3.5}$$

where $\mathcal{D} = (\mathbf{X}, \mathbf{y})$. We might want to approximate the $p(M_i|\mathcal{D})$ with the marginal likelihood that can be calculated efficiently using Equation 2.27 from Section 2.4, but it is not recommended to do so. The reason is that the marginal likelihood favours a complex model than a simpler one. A detailed discussion of this is explained in Appendix B Section B.2. The following example supports this statement. A greedy search is run on the 2-kernel dataset from Section 5.2. This dataset is a random function generated from a Gaussian process with a known combination of two kernels, $\mathrm{RBF} + \mathrm{Per}$. Figure 3.6a displays the negative log-marginal likelihood values. We can clearly see that the log-marginal likelihood favours a model with four kernel combinations even though the true generating Gaussian process uses two kernels.

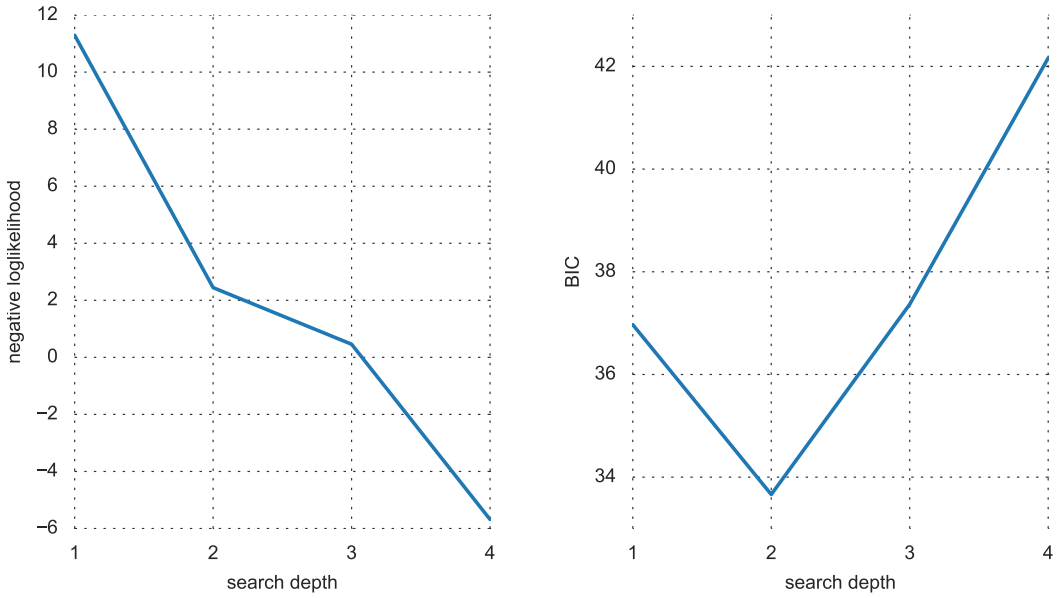
An asymptotic approximation to $p(M_i|\mathcal{D})$ is the Bayesian information criterion (BIC) [47]. The BIC is given by

$$\text{BIC} = -2\text{LL} + m \log n \tag{3.6}$$

The LL is the log-marginal likelihood of the model parameter, m is the complexity controlling number, and n is the number of data points. In our problem, m is equal to the number of parameters that the covariance function $\hat{k}$ has. The lower the BIC, the better the model is. The right-hand term can be understood as a penalty term to the log-marginal likelihood for preferring a complex model. Figure 3.6b shows that the BIC is capable of selecting the true number of the kernel (2) while it penalises the model with 3 and 4 kernels. Duvenaud et al. suggest using this criterion due to its simplicity [16]. Since training a GP model corresponds to maximising the marginal likelihood of the model parameter which is the parameter of the covariance function, computing a BIC is simply done by adding the right-hand term of the Equation 3.6 with the already calculated maximum likelihood from the training step (Section 2.4). Moreover, the benefit of using the BIC search metric is that all training data can be utilised for model selection. This is the opposite of the cross-validation metric, which will be discussed later, where a portion of the training data cannot be used.

Behind its simplicity, BIC comes with several attributes that deserve some attentions. BIC is an asymptotic approximation to the Bayesian posterior $p(M_i|\mathcal{D})$. This posterior computes the plausibility of the model M_i given the data at hand. By computing this posterior, we are interested in finding the true *generative* model of the data. The model M from the set $\mathcal{M}$ with the highest posterior corresponds to the model that best approximate the true generative model. Now the problem with this approach is two-fold. First, the asymptotic assumption of BIC might be too strict for our problem as we would need a

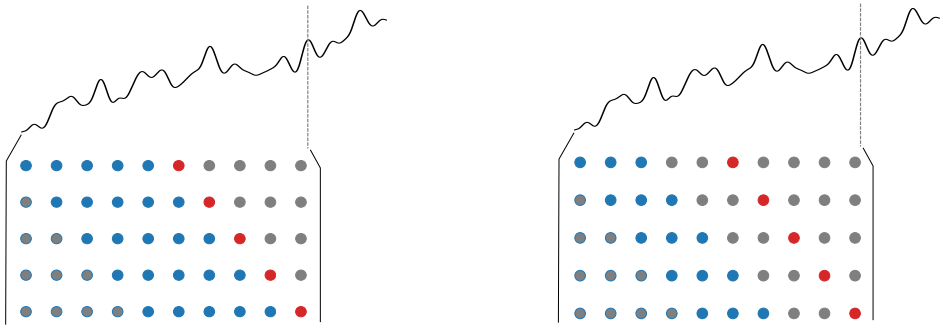
large sample $n \gg m$. Second, having a sensible generative model does not automatically translate to a good predictive model. It can be the case that the model is wrong yet can predict decently. Wasserman [54] and also Neath and Cavenaugh [37] recommend the BIC for an explanation problem, for example, to analyse prominent features that affect the data. Since this research is focused on forecasting, it is better to shift the focus to find the best *predictive* instead of a generative model. Because of this, we propose a cross-validation search metric, which will be discussed next.



(a) Negative log-marginal likelihood (b) BIC

Figure 3.6: A comparison between the log-likelihood and BIC as search metric. A greedy search is performed on a dataset which is generated by a Gaussian process with two covariance functions. The graph displays the score of the best model in each search depth where the lower the value, the better. It is clear that BIC penalises the model with kernel more than two.

Cross-Validation



(a) 1-step-ahead cross validation (b) 3-step-ahead cross validation

Figure 3.7: The cross-validation scheme. Here, the data training data is split into five folds. Blue dots denote the data that is used to train the model in each fold. The model is then evaluated on the test point indicated by the red dot. The evaluations are averaged over the number of the folds, and this score is utilised for model comparison. The grey dots signify the portion of the data that is not employed by the cross-validation.

We turn to the pragmatic approach of cross-validation

$$g(M_i) = \frac{1}{k}\sum_{i}^{k} e_i(M_i) \tag{3.7}$$

where the model score is the average of forecasting error computed with the error function e across k folds. Some error measurements are discussed in Section 5.1.2. The training data is split into several folds, and for each fold, a training and test set is made. The model is trained on a training set of each fold, and then the predictive accuracy is evaluated on the test set of the fold.

The fold structure is different from the standard k-fold cross-validation that is commonly found in machine learning evaluation. Cross-validation in time series forecasting should be treated differently due to serial dependencies between observations [2]. Bergmeir and Benitez recommend several folds for time series embedding scenarios [5]. We settle for a hold-out-sample scheme, which is shown in Figure 3.7. In the example, we split the data into five folds. Every blue dots in the fold are the data that is used to train the model. The red dot signifies the test point for evaluating the model. The computed performance is averaged by the number of the folds. This number is used to compare the predictive performance of the model M_i. The grey dots are points that are not used for training the model. Although simple, this scheme has a computational advantage. The training set in each fold differs only by the addition of a new data point at the end and the subtraction of a single data point at the start of the training set. We do not have to recompute from scratch the Cholesky decomposition for every fold. What we need is to recycle the Cholesky factor matrix from subsequent fold through the Cholesky factor update and downdate operation. More detail about this procedure can be found in Osborne et al. [40]. The computation burden of the first fold is $\mathcal{O}(n^3)$ and $\mathcal{O}(n^2)$ for the rest of the fold.

There are two benefits of using the cross-validation as opposed to the BIC. One, it forces the model selection to focus on the forecasting performance. Furthermore, the cross-validation gives the model the ability to consider the 'urgency' between horizons. As it is shown in Figure 3.7, we have different cross-validation settings for different horizons. Because of that, the model selection can choose a separate model for independent forecasting horizon. As an illustration, a model for a 1-step-ahead forecast should be different from a model for 20-step-ahead. The former should model wiggles and short-term variation while the latter should focus on modelling the long-term interaction. The BIC does not provide the mechanism to embed the horizon information.

Just like with any cross-validation, the downside of using a cross-validation metric is that it is more expensive, even with the computation shortcut that is given previously, to compute than the BIC because we have to train the model k times. Moreover, we have to perform separate cross-validation for the different horizon h. This becomes a problem if we want to forecast for more than one horizon. In addition to that, another drawback of using the cross-validation metric is that portion of the training data cannot be used for training the model because of the fold structure. This can be seen in Figure 3.7, where the data in grey denotes the portion that cannot be used for training.

3.2.4. Stopping Criterion

Table 3.1: The strength of the model against the model with higher BIC values [31]

ΔBIC	Strength against current BIC
0 - 2	Not worth more than a bare mention
2 - 6	Positive
6 - 10	Strong
>10	Very Strong

We implement a simple stopping criterion based on the difference between the best score of the current search level with the best score of the previous search level. Given the best search metric of the current search level l, g_l, and the best search metric of the last search level, g_{l-1}, the search will stop if the improvement of the model is below a certain threshold, $g_l - g_{l-1} < \tau$. The threshold depends on the

search metric that is used. For BIC, we can refer to the Table 3.1 that is proposed by Kass and Raftery [31, p.777]. Two model with ΔBIC between 0 and 2 are statistically not different, and thus we can use the values between 0 and 2 as our threshold for the stopping criterion. As for the cross-validation, there is no definitive rule that we can utilise for the threshold, so it should be determined by the user.

3.3. Forecasting the CO_2 Data: Structure Modelling Approach

We apply the structure modelling approach to the CO_2 data that was found in Section 2.5. The final covariance function that is returned by the greedy search with BIC as search metric is as follows

$$\hat{k}(t,t') = k_{\text{RBF}}(t,t') \times k_{\text{Lin}}(t,t') + k_{\text{RBF}}(t,t') \times k_{\text{Per}}(t,t') + k_{\text{RBF}}(t,t') \tag{3.8}$$

The result is displayed in Figure 3.8. We can see that the long-term forecasts are much better than the simple model found in Section 2.5. The forecast can follow the general trend and periodic structure of the dataset. The forecasts become inaccurate as the horizons go further, which is expected because as the horizon goes far, the change in the patterns become difficult to forecast. Nevertheless, the model still does a good forecasting job up to $h = 35$. The performance of the structure modelling approach will be thoroughly tested in Chapter 5.

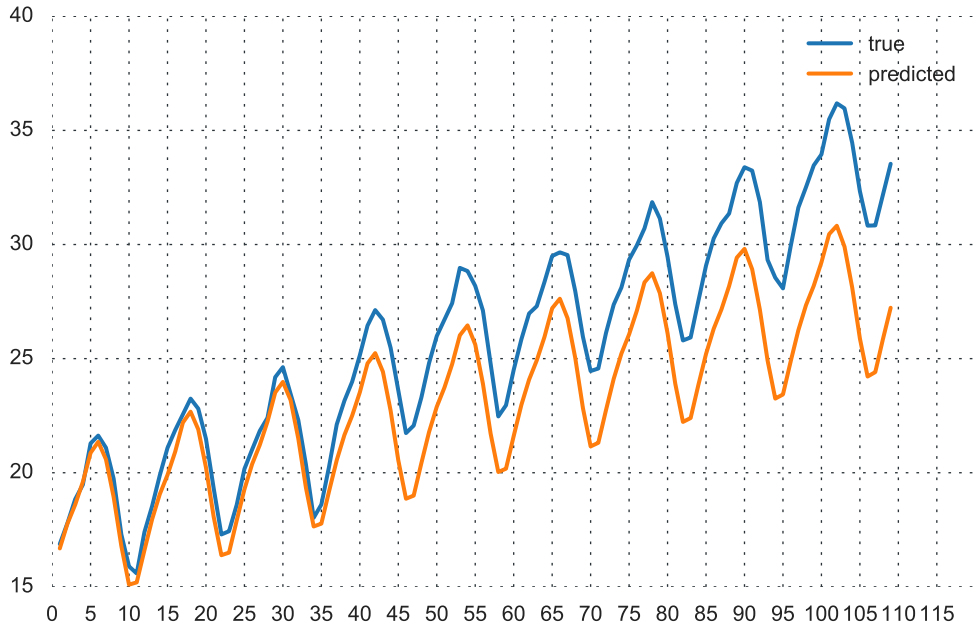


Figure 3.8: The forecasting result on the CO_2 dataset using the structure modelling approach. The long-term forecasts are much better than the simple model found in Section 2.5

4

Non-Linear Autoregressive with Gaussian Processes

We have seen how the structure modelling approach from the last chapter improves the simple GP model that is found in Section 2.5 by compositing a complex covariance function. This model provides a satisfactory forecasting result, as it has already been shown in Section 3.3. Still, the delicate modelling capability of the structure modelling approach comes at a high cost. In Section 3.2 it has been discussed that the structure modelling must evaluate a lot of candidate covariance functions which consumes a large computing resource. Due to this reason, an alternative GP regression model is proposed. This time, the complexity of the model is encoded in the feature, while the covariance function is kept simple. This alternative model is named the non-linear autoregressive (NAR) with Gaussian processes model.

A non-linear autoregressive (NAR) with Gaussian processes model is a GP regression model in which the inputs are its past observations

$$\mathbf{x}_t = [y_{t-1}, \cdots, y_{t-p}]^T$$
$$f(\mathbf{x}_t) \sim \mathcal{GP}(0, \hat{k}(\mathbf{x}_t, \mathbf{x}_t{}'))$$

Since a GP can model both a linear and non-linear function, Frigola refers this model as a non-linear AR process [18]. For naming purposes and to avoid confusion with similar NAR models with other regression techniques, a NAR with GP regression will be named as the *GP-NAR*. This model is relatively straightforward. Once we have transformed a time series into an autoregressive representation through a procedure called the *time series embedding*, we can use a GP regression like a standard multivariate regression. A one-step-ahead forecast can be computed using Equation 2.19 where $\mathbf{x}_* = [y_{T-p}, y_{T-p+1}, \cdots, y_T]^T$.

This model has three important design decisions. Firstly, how the multiple-step-ahead forecast is done. The two-step-ahead forecast cannot be computed directly because we need to know the actual value of the y_{T+1} as input. A similar situation happens for a $h = 3$ forecast since we need to know y_{T+1} and y_{T+2} and this will happen for all $h > 1$. Strategies to handle a h-step-ahead forecast for $h > 1$ will be discussed in Section 4.2. Secondly, how to choose the appropriate covariance function and this will be discussed in Section 4.3. Lastly, how the proper order of the autoregressive p is selected. This is important because when p is too small, the forecasting accuracy will be undermined because of the lack of predictive information. In contrast, when p is too large, the model will become too complicated. Section 4.4 will discuss methods to determine the correct order p.

4.1. Time Series Embedding

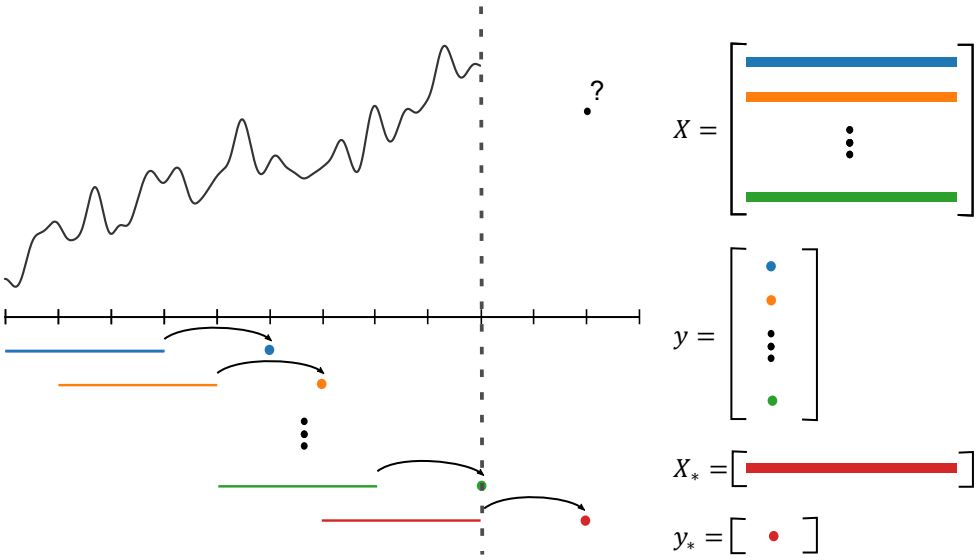


Figure 4.1: Time series embedding of an $AR(4)$ process with a two-step-ahead forecast

Transforming a time series into a p-th order of autoregressive representation is often referred as *embedding*. Figure 4.1 shows how the embedding is done. A window of length p is shifted on the time series, and values that fall within the window are used as a row of the input matrix $\mathbf{X}$. In Figure 4.1, it can be seen that the coloured rectangle represents the window of past values. In this example, $p = 4$, so the window covers four-time steps. The shaded circle represents the output value. Our case aims to forecast two-step-ahead. Hence there are two time step jumps for each of the output value. The last window from the time series becomes the test set.

In general, the training inputs and outputs of a time series embedding with p-lagged value and h-step-ahead forecast will have a dimension of $(T - h - p + 1) \times p$ and $(T - h - p + 1) \times 1$, respectively and are given by

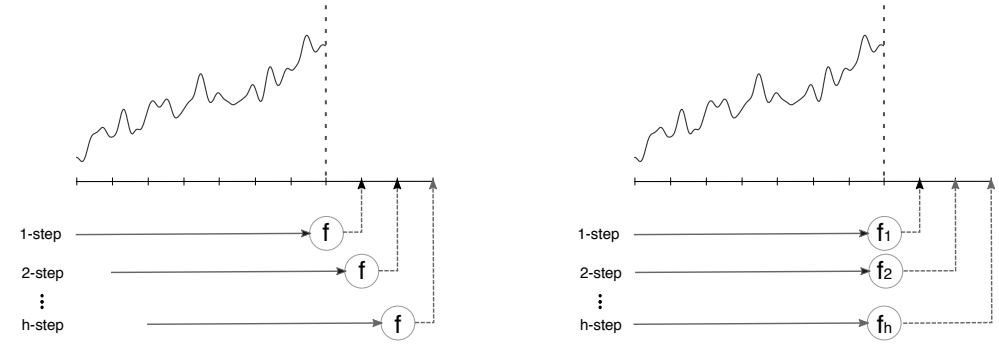
$$\mathbf{X} = \begin{bmatrix} y_1 & y_2 & \cdots & y_p \\ y_2 & y_3 & \cdots & y_{p+1} \\ \vdots & \vdots & \ddots & \vdots \\ y_{T-h-p+1} & y_{T-h-p+2} & \cdots & y_{T-h} \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y_{p+h} \\ y_{(p+h)+1} \\ \vdots \\ y_T \end{bmatrix} \tag{4.1}$$

while the test input and output are

$$\mathbf{X}_* = \begin{bmatrix} y_{T-p+1} & y_{T-p+2} & \cdots & y_T \end{bmatrix} \qquad \mathbf{y}_* = \begin{bmatrix} y_{T+h} \end{bmatrix} \tag{4.2}$$

4.2. Forecasting Strategies

As it was mentioned earlier, a two-step-ahead forecast or more cannot be computed directly using this GP-NAR approach. To calculate y_{T+2}, we need the input from y_{T+1} which is not available since it is in the future. A similar situation happens for all forecasts where $h > 1$. In this NAR setting, there are two strategies to perform a multiple-step-ahead forecasting, the *recursive* and *direct*.



(a) Recursive (b) Direct

Figure 4.2: Illustrations that show the difference between the recursive and direct strategy. The former trains a single one-step-ahead model while the latter trains separate model for every h-step-ahead forecast.

Recursive approach trains a single one-step-ahead forecasting model to make a multiple-step-ahead prediction. The illustration is shown in Figure 4.2a. First, we train a one-step-ahead forecasting model and predict y_{T+1}

$$y_{T+1} \sim \mathcal{N}(y_{T+1}|\mu_1, \sigma_1^2) \tag{4.3}$$

using Equation 2.19, where $\mathbf{x}_* = [y_{T-p}, y_{T-p+1}, \dots, y_T]^T$. Then we compute a two-step-ahead forecast using the similarly trained one-step-ahead model from the previous forecast

$$y_{T+2} \sim \mathcal{N}(y_{T+2}|\mu_2, \sigma_2^2) \tag{4.4}$$

but now our test input is $\mathbf{x}_* = [y_{T-p+1}, y_{T-p+2}, \dots, \mu_1]^T$. We use the mean value of μ_1 to estimate the real value of the y_{T+1}. This calculation is repeated h times to get the h-step-ahead forecast.

On the one hand, the recursive strategy is simple, because it requires a single one-step-ahead model, and takes advantage of the model to forecast repeatedly up to the horizon of interest. On the other hand, this model suffers from error propagation, because the forecast value is computed from the previously predicted value, not the actual one. As a consequence, if the $h-1$-step-ahead prediction has a large error, it will be accumulated in the h-step-ahead.

Another disadvantage of the recursive strategy is that the uncertainty does not get properly passed on to further forecast horizon. Here, we only use the predictive posterior mean of previous forecasts and disregards the variances completely. Consequently, every new forecast has the uncertainty of a new one-step-ahead forecast. So, even a $h = 24$ forecasts will have a small one-step-ahead uncertainty even though logically a $h = 24$ should have a much larger uncertainty than the $h = 1$ forecast. Girard et al. propose to include both the mean and variance of the predictive posterior from the previous forecasts to calculate the current forecast [19]. They solved the problem of uncertainty propagation but the downside of their method is that the predictive posterior integral becomes analytically intractable thus an approximation is needed. Even so, their method still does not solve the problem of the error propagation. Everything depends on the accuracy of the one-step-ahead forecast so if the model forecast poorly, then the error for further forecasts gets inflated very quickly.

Direct strategy is the complete opposite of the recursive one. Here, instead of using a single one-step-ahead repeatedly, we create different model for each h-step-ahead forecast. Figure 4.2b shows the illustration. This approach is generally more expensive than the recursive one, because we are required to train different models for different forecasting horizons. But unlike the recursive, the direct strategy does not suffer from error propagation, which becomes its favourable justification against the recursive approach. The drawback of this method is that we have to assume that the forecast for every different horizon is independent [4]. This of course is not true, because the forecast of $\hat{y}_{T+1}$ most likely will influence the forecast of $\hat{y}_{T+2}$. Nonetheless, it is reported that the direct strategy shows better empirical time series forecasting performance with neural network [51] and also GP-NAR. Yan et al. perform an

extensive experiment with 111 time series datasets to compare the performance of GP-NAR with the recursive and direct strategy [56]. Their result suggests that direct strategy is significantly better than the recursive approach, in term of forecasting performance for short to medium-term forecasts.

Choosing the forecasting strategy will ultimately depend on the forecasters. But, seeing the empirical evidence that the direct strategy performs better due to no error propagation, the trade-off is worth it. For this reason, the direct strategy will be chosen as the forecasting strategy for the GP-NAR.

4.3. Kernels

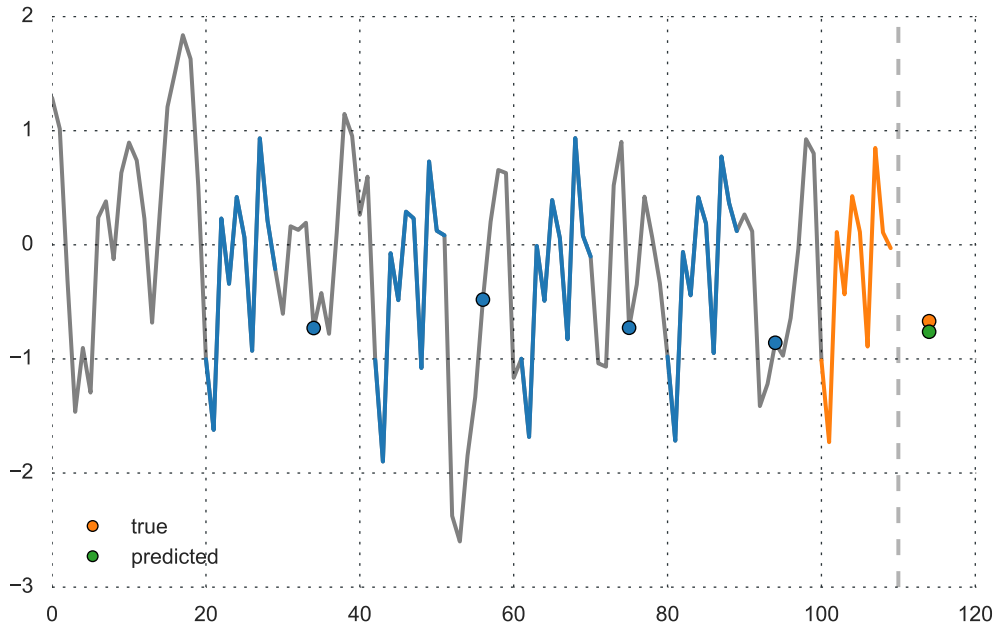


Figure 4.3: An example of a 5-step-ahead forecasting problem which is solved using GP-NAR(10) with RBF covariance function. The test input vector is denoted by the orange line while the test output is the orange dot. Since there are many similar autoregressive vectors on the training data (denoted in blue), the RBF kernel can pick up the similarity and forecast accurately.

Modelling a GP-NAR can be well approached by local kernels such as the RBF and RQ which have been discussed in Section 2.3. Remember that the local kernels assign high similarities to the inputs that are close. In the GP-NAR case, similar inputs are assigned to the similar autoregressive vectors. Consequently, if there are autoregressive vectors found in the training set which are akin to the autoregressive vector of the test set, then the forecast can be made with high confidence. Figure 4.3 depicts how the RBF kernel models a time series data. In the illustration, the task is 5-step-ahead forecast with $p = 10$. The test set input and output are denoted in orange. Since there are many similar autoregressive vectors on the training data (indicated in blue), the RBF kernel can recognise the similarity and forecast accurately. In this example, the time series which exhibits a periodic-like pattern can be picked up with a local covariance function such as the RBF. The key is that the training data must contain many *similar* autoregressive patterns in order to make an accurate forecast.

Section 2.3 defines both the RBF and RQ kernel that are instrumental to the GP-NAR model. The definition for both kernels is written for scalar input, while in GP-NAR, for autoregressive order $p > 1$, the input are vectors. The RBF and RQ kernels can be extended to vector input as follows

$$k_{\text{RBF}}(\mathbf{x}, \mathbf{x}') = \sigma^2 \exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{x}')^T \mathbf{M}(\mathbf{x} - \mathbf{x}')\right) \tag{4.5}$$

$$k_{\text{RQ}}(\mathbf{x}, \mathbf{x}') = \sigma^2 \exp\left(1 + \frac{1}{2\alpha}(\mathbf{x} - \mathbf{x}')^T \mathbf{M}(\mathbf{x} - \mathbf{x}')\right)^{-\alpha} \tag{4.6}$$

The lengthscale parameter λ is contained within the matrix $\mathbf{M}$, and there are several ways to specify the matrix $\mathbf{M}$, which is outlined below

$$\mathbf{M} = \begin{cases} \lambda^{-2}\mathbf{I} & \text{isotropic kernel} \\ diag(\lambda_1^{-2}, \ldots, \lambda_p^{-2}) & \text{ARD kernel} \end{cases} \tag{4.7}$$

The first way is the *isotropic* kernel where all elements of the vector will be assigned to the same lengthscale parameter. The second way is to assign different lengthscale parameters to each autoregressive value where the lengthscales are the diagonal elements of the matrix $\mathbf{M}$. This arrangement assigns relevance weight to each element of the autoregressive vector. The most relevant element will be assigned the smallest lengthscale and vice versa. Due to this weighting mechanism, this approach is commonly referred as the *Automatic Relevance Determination (ARD)*. In ordinary multivariate regression, the ARD kernel can provide information of which feature contributes the most to the variation. In the GP-NAR, the ARD kernel could be more beneficial than the isotropic kernel, because the features are the autoregressive value of the time series. Through the ARD kernel, higher relevance can be assigned to the first few autoregressive elements than the p-th one because the first few elements are the most correlated with the current value.

4.4. Determining the Order of the Autoregressive Model

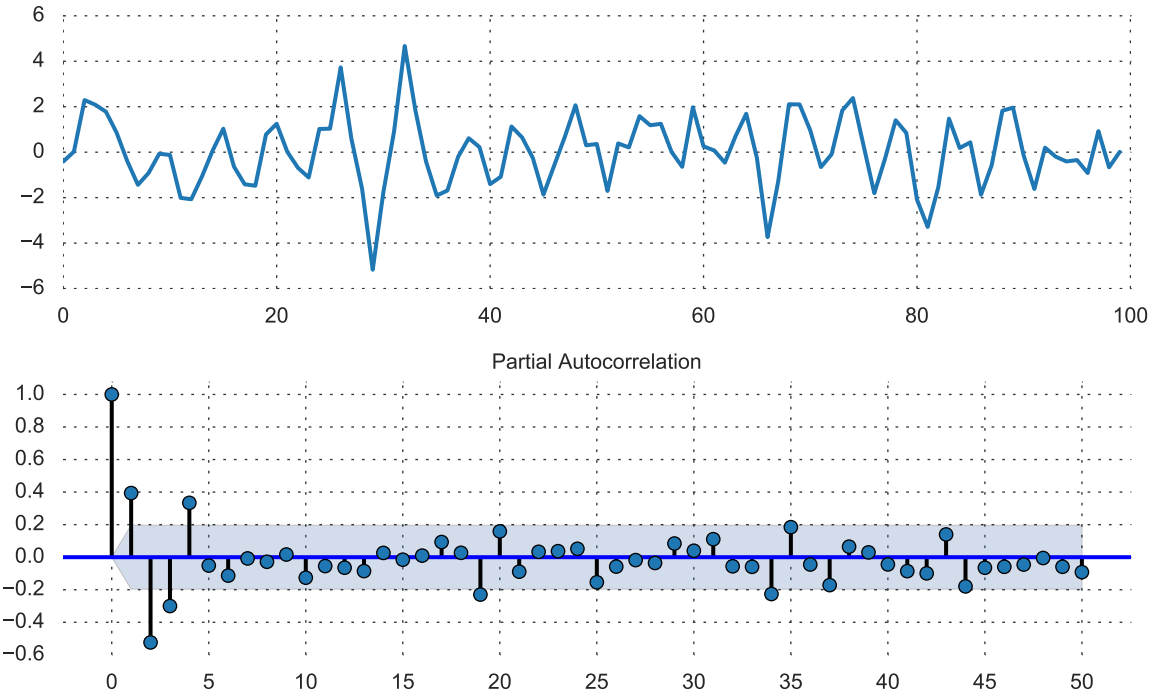


Figure 4.4: Top row: A plot of a random $AR(4)$ time series. Bottom row: A corresponding PACF plot for the $AR(4)$ time series. The PACF shows significant autocorrelation values up to the fourth lag, and then the values become insignificant. The area shaded in blue indicates 95% confidence interval. The PACF plot displays a strong indicator to use $p = 4$ for GP-NAR.

The last design decision to be made is how to choose the order of the autoregressive model (p). Choosing the right order is important. An order that is too small will undermine the accuracy of the GP-NAR. In contrast, using an order that is too large order will add computation complexity while adding no real improvement to the model. Choosing the order is a non-trivial matter because it is a problem dependent issue. Different time series has different dynamics. Therefore, a different order is needed to model those dynamics sufficiently. Although there are several kinds of research on GP-NAR [8, 10, 19, 23, 35, 56], these works do not explicitly outline how they determine the order of the AR. Because of that, this section will discuss methods of choosing the right AR order.

One way to select the order is through a model selection procedure, similar to how the structure modelling approach determines the best kernel (Section 3.2). The algorithm starts with $p = 1$, repeatedly adds the order by one, and keeps progressing until a little change in the score is shown. The similar stepwise approach has been adopted by Hyndman and Khandakar in their automatic model selection approach for ARIMA model [29]. They add the order of the AR and MA of an ARIMA model in a stepwise manner while evaluating the fit of the model. The discussion on Section 3.2.3 recommends that pragmatic cross-validation approach is used as search metric instead of BIC, therefore a cross-validation metric will also be used for GP-NAR.

The approach as mentioned earlier should be working well for most cases since it empirically evaluates the order from the training data. However, checking different order in a stepwise manner might be too exhaustive to perform. As an alternative, there is a more systematic way. We could consult to the autocorrelation function (ACF) plot and the partial autocorrelation function (PACF) plot to determine the order of the autoregressive model [9]. This method is commonly used in time series analysis to ascertain the order of an AR and MA in the ARIMA model.

In practice, determining the order of the AR is to find p *significant lags* before the value becomes insignificant in the PACF plot. Figure 4.4 shows an example of a PACF plot of a random $AR(4)$ process. The PACF plot shows significant autocorrelation values up to the fourth lags, after that the value start to decay to zero and become insignificant. This sign tells us that it is recommended to choose $p = 4$. Please note that the first autocorrelation value in the plot is the correlation with itself and that explains why it always has a value of one. The PACF plot is a useful diagnostic tool to find the optimum autoregressive order of the GP-NAR. But, mind two drawbacks of PACF plot before using it. First, the PACF assumes that the time series is *stationary*. Therefore, any non-stationarities such as trend and seasonal that exist within the time series must be corrected prior using. Second, the autocorrelation on the PACF plot measures the linear correlation. Therefore, it might be the case that the PACF returns no significant lag although there is a strong non-linear correlation in the time series.

It is recommended to combine both the empirical evaluation and PACF diagnostic to determine the order of the autoregressive. On the one hand, using a PACF plot alone is limited to stationary and linear time series. On the contrary, using a model selection approach is extensive, since we are enumerating the order in a stepwise manner. During the experiment of this research, it is found that combining both approaches works well. The procedure starts by identifying the order from the PACF plot. Then, the optimal lag as a result of the PACF plot reading will be used as a starting point for the empirical evaluation. From this research experience, the optimum empirical order is usually found within 1 up to 5 lags from the PACF-optimum lag.

4.5. Forecasting the CO_2 Data: GP-NAR Approach

Similar to Chapter 3, we demonstrate the long-term forecasting capability of the GP-NAR on the CO_2 dataset. The model uses an isotropic RBF kernel with $p = 10$. The forecasting result is depicted in Figure 4.5. From the result, we can see that the long-term forecasting performance of the GP-NAR is worse than the structure modelling, especially for longer horizons. We will report the long-term forecasting performance on various datasets in Chapter 5 and from there we shall see whether the finding that is found in this section will hold in general.

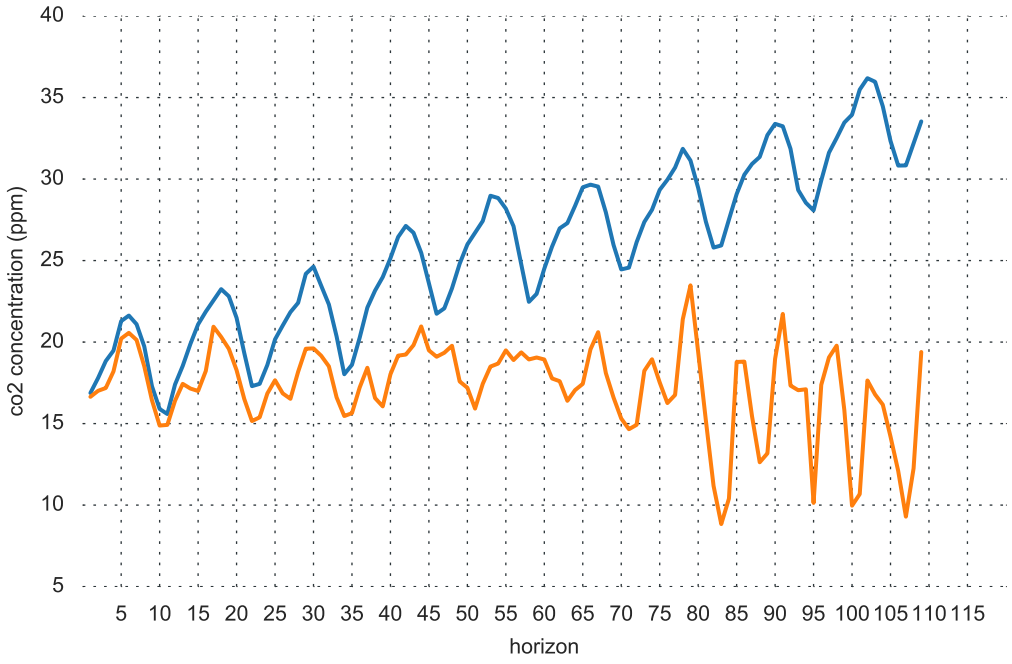


Figure 4.5: Long-term forecasting result on the CO_2 dataset using the GP-NAR approach. The forecasting performance is worse than the structure modelling, especially for longer horizons.

5

Experiment

5.1. Setup

5.1.1. Window

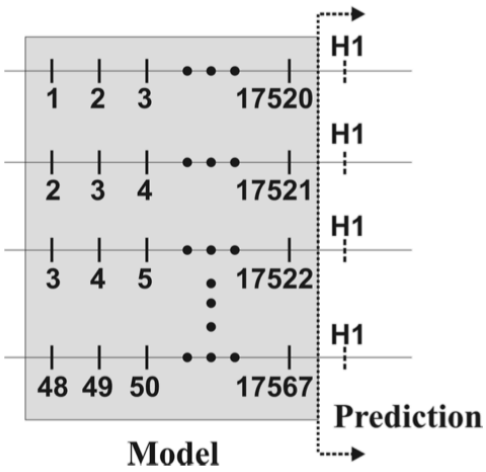


Figure 5.1: The test on a time series dataset is run on several windows to provide a fair evaluation [25]. This diagram illustrates how the windowed test is performed.

To prevent a 'lucky' prediction, where a prediction looks better at one point in the future by coincidence, all the test will be run over several windows. Here, a *window* of size N is shifted on the time series data. Figure 5.1 shows the schematic of a window-based test. In the example, we run a 48-window test. In the first window, we use time series from $t = 1$ up to $t = 17520$ for training our model, then forecast the value on $t = 17521$. For the second window, we shift the window by one time unit, so now we train a model with a time series from $t = 2$ up to $t = 17521$ and forecast $t = 17522$. The amount of shift, which is called the *stride*, does not have to be one. With this testing scheme, we make sure that our tested model forecast consistently.

By default, all of our tests will use the windowing system. This type of evaluation is widespread in wind energy forecasting literature and dissimilar from the kind of testing that we often find in econometric research [9, 28, 48]. In econometric, the typical evaluation setting is using a single window, but the forecast is run on multiple horizons. Hence, we would have many forecasts across different horizons. In contrast, the window-based test is undertaken at multiple windows but done on a single horizon. This research will perform the latter type of test. Evaluating in multiple windows is more fitting since we can see the average forecasting performance on different horizons.

5.1.2. Evaluation Metric

Given a time series observation at time t, y_t, and the corresponding forecast $\hat{y}_t$, the *forecast error* is given by $e_t = y_t - \hat{y}_t$. In our GP model, we have a distribution of outputs, so $\hat{y}_t$ is equal to the mean of the distribution. When we evaluate multiple forecasts results, we would have a set of forecast errors, $\{e_t\}$. Two methods are commonly used to evaluate the set, and those are the *mean squared error (MSE)* and the *mean absolute error (MAE)*. MSE computes the mean of the squared error over the error set, while the MAE calculates the mean of the absolute error

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{5.1}$$

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{5.2}$$

This set of errors signifies a different kind of errors, where these can be forecasting errors over different horizons, forecasting errors over a single time series on multiple windows, or forecasting errors over different time series. Both MSE and MAE have the benefit of removing the difference between negative and positive errors. One difference between MSE and MAE is that the MSE is more reactive to extreme errors because of the square operation. Large error gets amplified while small error gets contracted. MAE does not have that effect as all errors have equal significance.

One small problem when using either MSE or MAE is that both are scale dependent. Therefore, they cannot be used to compare the performance between distinct time series. For that reason, the *percentage error* p_t is developed, where $p_t = 100e_t/y_t$. Again, it is common to evaluate multiple percentage errors. *Mean absolute percentage error (MAPE)* is one metric to evaluate a set of percentage errors $\{p_t\}$ and is given by

$$\text{MAPE} = \frac{100}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \tag{5.3}$$

where multiple error values are evaluated across different horizons, windows, or series. The problem that comes with MAPE is zero division. When we have zero values in our data, the denominator of MAPE becomes zero and causes a zero division error. In this experiment, all datasets contain no zero value, so the utilisation of MAPE is justified.

Another frequently used error evaluation in time series forecasting is to calculate the *improvement* over a baseline model [15, 30]. This method calculates the error (MAE for example) from both our proposed model and a baseline model, and calculates the difference in error made by these two models. In general, the improvement (I) can be computed as

$$\text{I} = 100\frac{\text{EC}_* - \text{EC}}{\text{EC}_*} \tag{5.4}$$

where EC is the forecasting error made by our forecasting model while EC_* is the error made by the baseline model. The baseline method can be any forecasting method, although it is common to use the *persistence* or naïve method as a baseline (Section 1.4.2). In this experiment, the MAE, MAPE, and improvement evaluation score will be used to evaluate our forecasting performance, unless it is mentioned otherwise.

5.1.3. Software

The software is published on the following link `https://github.com/bagasabisena/mscthesis`. The Gaussian process regression is implemented using the `GPy` [21] framework with our own implementations for Chapter 3 and Chapter 4. This experiment uses Bayesian linear regression implementation from the `scikit-learn` library [41]. For ARIMA, we utilise the `forecast` package [29].

5.2. Dataset

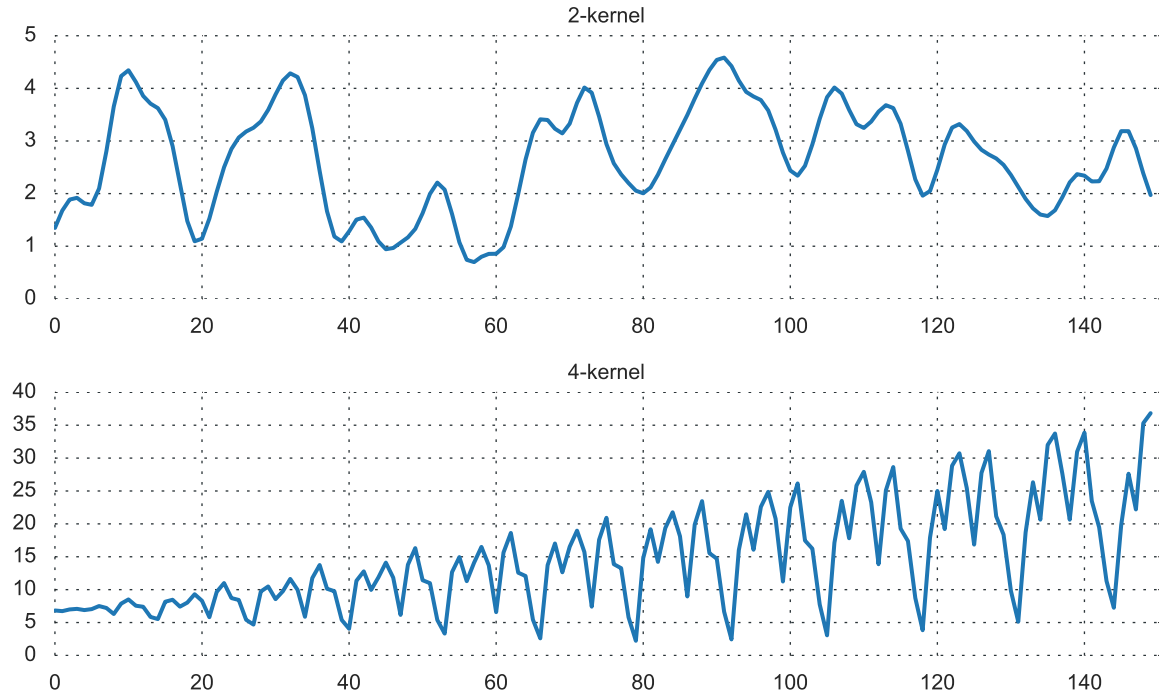


Figure 5.2: Synthetic datasets, where the data are generated from a Gaussian Process with known prior. Top row: 2-kernel dataset (RBF + Per). Bottom row: 4-kernel dataset (Lin + RBF + Per × Lin)

Evaluation is done on two types of datasets. The first type is a synthetic time series, which is generated from a known generative model. This time series is generated from a Gaussian process with a predefined covariance function. In this research, two synthetic datasets are generated. One dataset is generated from a Gaussian process with a $\mathrm{RBF} + \mathrm{Per}$ kernel. The other one is generated from a $\mathrm{Lin} + \mathrm{RBF} + \mathrm{Per} \times \mathrm{Lin}$ kernel. These datasets will be called the *2-kernel* and *4-kernel* dataset respectively. Figure 5.2 shows the generated time series. Each dataset contains 150 data points.

In addition to the two synthetic datasets, other experiments are done on two real-world datasets. The first dataset is the TU Delft *wind speed* dataset. The plot of the wind speed dataset is shown in Figure 5.3 top row. This dataset is collected from several weather stations in the Rotterdam area, The Netherlands. The station captures a variety of weather variables, and one of them is the wind speed. The stations sample the data every 5 minutes, and thus it gives a uniformly sampled discrete-time time series data. The wind speed is chosen because of its non-stationarities, with many trends and periods that are changing over time. The dataset is publicly available for download [1]. For this experiment, the data is upsampled into 1-hour sample, and values within the 1-hour period are aggregated by taking the average. This operation is done to remove any noises that are generated by the sensors and might obfuscate the underlying signal. The data is taken at 2015-12-09 10:00:00. The size of the training data is equal to 60 days of past observations from the point the data is taken, which means we are going to use data from 2015-10-10 10:00:00 until 2015-12-09 10:00:00 with $T = 1440$.

The second real-world dataset is the electricity net generation measured in billions of kilowatt hours (kWh) in the United States [28]. From now on, this dataset will be referred as the *electricity* dataset. The observation is taken monthly from 1973 until 2010, where $T = 454$. This dataset is chosen because it has strong non-stationarities with yearly trend and seasonality patterns. The bottom row of Figure 5.3 shows the plot of the *electricity* dataset.

[1] http://weather.tudelft.nl/csv/

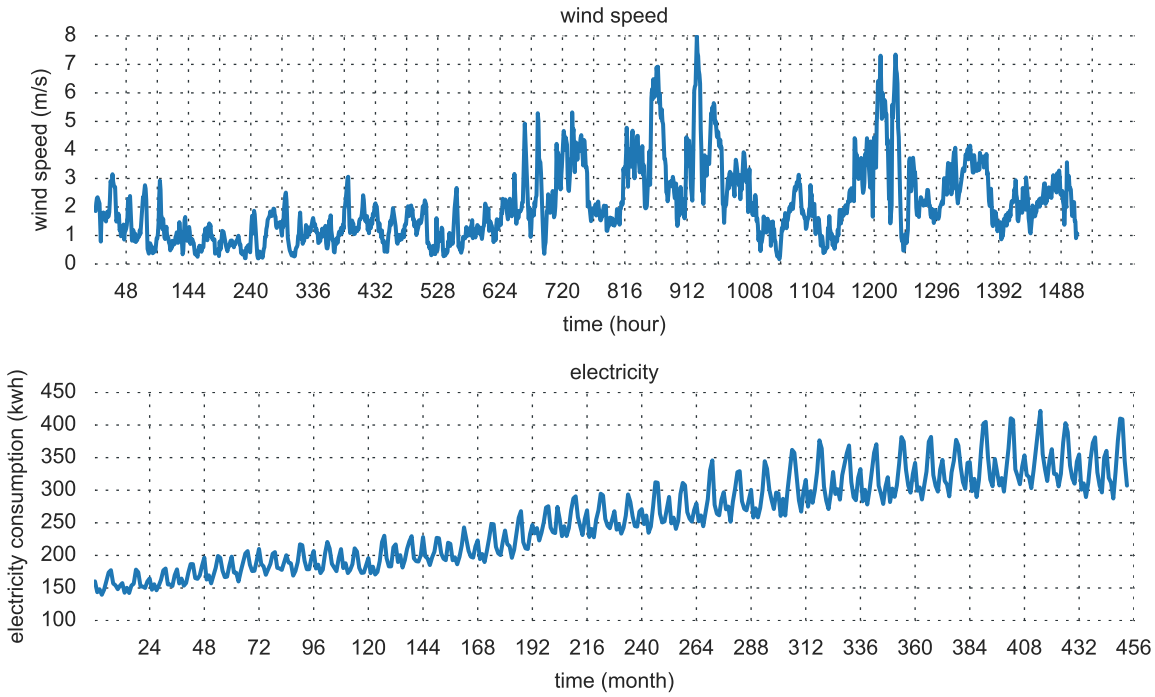


Figure 5.3: Real-world datasets. Top row: wind speed dataset. Bottom row: electricity dataset.

5.3. Forecasting Model

Table 5.1: List of the forecasting models that are used in the experiment

Model	Description
ARIMA	Autoregressive integrated moving average methods
persistence	$y_{T+h} = y_T$ for all h
BLR	Bayesian linear regression
RBF-iso	GP-NAR with isotropic RBF kernel
RQ-iso	GP-NAR with isotropic rational quadratic kernel
RBF-ard	GP-NAR with automatic relevance determination RBF kernel
RQ-ard	GP-NAR with automatic relevance determination rational quadratic kernel
greedy	structure modelling using the greedy search strategy
exhaustive	structure modelling using the exhaustive search strategy

Evaluation of the proposed GP-based methods is not complete without the comparison with other existing probabilistic techniques. In this experiment, we focus on the point forecasting performance. In other words, we compare the mean of the distribution that is returned by each of the probabilistic models. There is four main probabilistic forecasting model that will be used for the experiment: Bayesian linear regression (BLR), ARIMA, GP regression with the structure modelling approach, and GP-NAR. The detail behind the BLR has been discussed in Section 2.1. Instead of using the time index as input, The BLR in this experiment will utilise similar autoregressive structure as the GP-NAR

$$\mathbf{x}_t = [y_{t-1}, \dots, y_{t-p}]^T \tag{5.5}$$

We are using the automatic ARIMA approach that is proposed by Hyndman and Khandakar [29]. The traditional ARIMA requires the forecasters to choose the order of AR (p), differencing (d), and MA (q)

manually. This automatic method implements an automatic selection of those order via a step-wise search algorithm. For structure modelling, the only required user parameter is the number of search level, and the search level is set to 6 for all datasets. The last forecasting method is the GP-NAR. We are using two kernels, the RBF and RQ kernels with both the isotropic and ARD setting. This gives us 4 GP-NAR models in total. One important parameter that defines both the GP-NAR and Bayesian linear regression is the order of the AR. In this experiment, the order is determined using the hybrid of cross validation and PACF diagnostic method, as it was discussed in Section 4.4. Results of both the GP-NAR and Bayesian linear regression are already trained using the optimum lag according to the cross-validation. Table 5.1 lists all the forecasting models that are employed in the experiment, along with their abbreviated model name.

5.4. Test 1: Search Strategy Evaluation for Structure Modelling

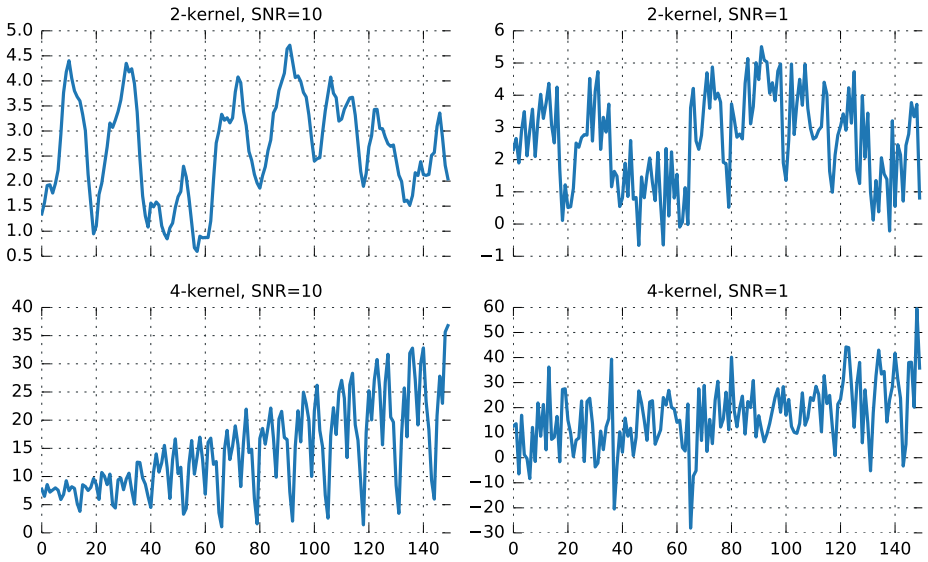


Figure 5.4: Plots of synthetic datasets corrupted by noise with varying level

This test assesses two aspects of the structure modelling approach, which are the search strategy and the search metric. We compare the covariance function that is returned by our search strategy to the true generating one. We utilise the 2-kernel and 4-kernel datasets, where the true generating process is known. Ideally, these search strategies should be able to return the same generating covariance function. These datasets will be corrupted by two different levels of noises. First, noise with $1/10$ of the variances of the signal will be added, such that the signal-to-noise ratio is equal to 10 ($SNR = 10$). Second, the signal-to-noise ratio is equal to 1 ($SNR = 1$), hence the noise variance is equal to the signal variance. Figure 5.4 depicts the plot of the noisy synthetic datasets.

Table 5.2 and Table 5.3 show how the optimum kernel are returned our kernel search. Table 5.2 reports the result for SNR $= 10$ synthetic dataset while Table 5.3 reports the result for SNR $= 1$ dataset. Both tables display the returned kernel for both the greedy and exhaustive strategy, using two different search metric: the BIC and cross-validation (CV) search metric (Section 3.2). The BIC returns the same kernel for all forecasting horizons and utilises all the training data while the CV does not, as different forecasting horizons require training separate models. Because of that, we might see different kernel outputs for different forecasting horizons for the cross-validation method.

When the noise level is low (SNR $= 10$), the dominant structure can be recovered by the search with the *BIC* search metric although not the exact kernel is returned. For the 2-kernel dataset, the dominant structure is a periodic with some noise. Both search algorithms return the 'locally periodic' $\mathrm{RBF} \times \mathrm{Per}$ covariance function, which is close to what the true generating kernel $\mathrm{RBF} + \mathrm{Per}$. A similar observation is found in the 4-kernel dataset. The most dominant covariance function from this dataset

Table 5.2: The optimum kernel that is chosen by the greedy and exhaustive search strategy, under two different model selection metric: BIC and cross-validation (CV). Here, the result for the SNR = 10 noise level is reported.

	greedy (BIC)			exhaustive (BIC)		
True Kernel	$h = 10$	$h = 20$	$h = 40$	$h = 10$	$h = 20$	$h = 40$
RBF + Per	RBF × Per			RBF × Per		
Lin + RBF + Per × Lin	Per × Lin + Lin × Lin			Per × Lin + Lin × Lin		

	greedy (CV)			exhaustive (CV)		
True Kernel	$h = 10$	$h = 20$	$h = 40$	$h = 10$	$h = 20$	$h = 40$
RBF + Per	Per	Per	Per + Per	Per	Per + Per	Per + Per
Lin + RBF + Per × Lin	Lin × Lin	Lin × Lin	Lin × Lin	Lin × Lin	Lin × Lin + Lin × Lin	Lin × Lin + Per + Per

Table 5.3: The optimum kernel that is chosen by the greedy and exhaustive search strategy, under two different model selection metric: BIC and cross-validation (CV). Here, the result for the SNR = 1 noise level is reported.

	greedy (BIC)			exhaustive (BIC)		
True Kernel	$h = 10$	$h = 20$	$h = 40$	$h = 10$	$h = 20$	$h = 40$
RBF + Per	RBF			RBF		
Lin + RBF + Per × Lin	Lin			Lin		

	greedy (CV)			exhaustive (CV)		
True Kernel	$h = 10$	$h = 20$	$h = 40$	$h = 10$	$h = 20$	$h = 40$
RBF + Per	Per	Per	Per + Per	Per	Per + Per	Per × Per
Lin + RBF + Per × Lin	Lin × Lin	Lin × Lin + Per	Lin × Per	Lin × Lin + Per	Lin × Lin + Per	Lin × Lin + Per

is the $\mathrm{Per} \times \mathrm{Lin}$, which creates a signal with growing amplitude. These kernels can be recovered by both the greedy and exhaustive search. The rest of the kernel cannot be recovered since the signal is insignificant compared with the dominant kernel $\mathrm{Per} \times \mathrm{Lin}$. Figure 5.5 shows the residual of the 4-kernel dataset after removing the $\mathrm{Per} \times \mathrm{Lin}$ part. It can be observed that the amplitude of this signal is subtle in comparison with the actual signal. We can conclude that in the low noise setting, both the greedy and exhaustive strategies can recover the dominant kernel when the BIC search metric is used. This finding confirms the nature of BIC which is finding the best approximate to the true *generating* model.

For the noisy dataset (SNR $= 1$), it can be seen that both the greedy and exhaustive search with the BIC search metric cannot recover the true generating kernel. This can be explained by the fact that the noise hides the true signal that resides in the data. The diagram in Figure 5.6 shows the search path of the greedy search for the 4-kernel dataset. The algorithm always chooses the Lin kernel and cannot find the periodic pattern because of the noise. Adding Lin kernel does not actually mean anything since the addition of linear kernel will result in another linear kernel, so the algorithm should have stopped this. The result shows that both algorithms are incapable of recovering the generating kernel and even running an exhaustive search will not help because the data is too noisy to begin with.

In contrast with the BIC, the cross-validation metric does not return a comparable kernel combination as the true generating kernel for both the SNR $= 10$ and SNR $= 1$ datasets. This is expected, as the cross-validation metric aims to find the model with the best *predictive* capability regardless of whether the model is close to the true generating model. Because of that, we perform a test to directly compare

Figure 5.5: The decomposition of the 4-kernel synthetic dataset, after removing the $\mathrm{Per} \times \mathrm{Lin}$ component. This result in a residual component, which is insignificant

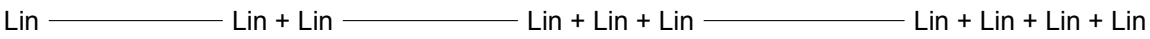


Figure 5.6: How the greedy search progresses for a 4-depth search to find the best kernel of the 4-kernel dataset with SNR=1. The search always chooses the linear kernel as the local optimum and cannot see the periodic pattern because the data is too noisy.

the forecasting performance of the model selected by the BIC and cross-validation, using the greedy search strategy. In this test, we forecast the synthetic dataset, using the same setting that is used in Section 5.5. We forecast on the dataset without noise, unlike the previous test.

The result is reported in Table 5.4. From the table, it is clear that the model that is selected by the BIC provides superior long-term forecasting across several horizons than the cross-validation. This reason is that the cross-validation does not use all training data for forecasting. Consult to Figure 3.7 for better illustration. When we use a larger forecasting horizon, the portion of training data that is unused becomes bigger. Besides that, the fraction of data that has to be given up is the data that are closer to the present time. This drawback wastes the potential learning source, as the most recent data should be the most relevant to the forecast.

Table 5.2 and Table 5.3 reveal that both greedy and exhaustive return the similar covariance function for both the 2-kernel and 4-kernel datasets. The computation time that is used for searching is reported in Table 5.5. Knowing that the greedy can recover the same kernel combinations with much less search time, especially when the search level is deep, the greedy search strategy is highly recommended as the search strategy. This observation also discourages the usage of another sophisticated search methods, such as the genetic algorithm or even the exhaustive search, because the greedy heuristic can recover the exactly same kernel on the fraction of the cost of the exhaustive search. By the end of this section,

Table 5.4: The forecasting performance of the 2-kernel and 4-kernel dataset without noise using the structure modelling approach with greedy search strategy. The purpose of this test is to see the predictive performance of the model chosen by the BIC and cross-validation (CV). It can be seen that the BIC has better forecasting performance across all horizons. The performance is measured by the mean absolute error and the value denoted in the parentheses is the standard deviation.

	2-kernel			4-kernel		
model	$h = 10$	$h = 20$	$h = 40$	$h = 10$	$h = 20$	$h = 40$
BIC	0.99 (0.74)	0.72 (0.67)	0.75 (0.80)	0.27 (0.20)	0.56 (0.56)	2.71 (1.79)
CV	1.32 (0.91)	1.24 (0.92)	0.86 (0.56)	8.41 (7.15)	11.55 (7.16)	20.18 (11.51)

Table 5.5: The search time comparison between the greedy and exhaustive strategy for the synthetic dataset using the BIC search metric

	Search Time (s)	
True Kernel	greedy	exhaustive
RBF + Per	14.797	41.306
Lin + RBF + Per × Lin	121.974	8532.929

we come up with two important findings of structure modelling approach. First, the experiment shows that BIC model selection is better than the cross-validation, both in term of *generative* and *predictive* performance. Second, the greedy search is recommended to be used as a search strategy, for both practical and performance concern. Henceforth, only the greedy strategy combined with the BIC model selection is considered for further tests.

5.5. Test 2: Synthetic Dataset Forecasting Performance

Table 5.6: The forecasting performance of the 2-kernel dataset. MAE denotes the mean absolute error and MAPE signifies the mean absolute percentage error. Those performance measures are evaluated over 20 windows. Values marked in bold are not significantly different from the best-performing forecasting model, in a paired t-test with a p-value of 5%. Value denoted in the parentheses is the standard deviation.

	MAE			MAPE (%)		
model	$h = 10$	$h = 20$	$h = 40$	$h = 10$	$h = 20$	$h = 40$
persistence	**0.98 (0.68)**	0.83 (0.39)	1.34 (0.86)	37.3 (31.8)	29.7 (18.0)	67.7 (53.1)
ARIMA	**0.70 (0.41)**	**0.49 (0.34)**	**0.53 (0.29)**	**21.9 (10.7)**	**16.4 (11.1)**	**26.9 (20.5)**
BLR	0.80 (0.42)	**0.48 (0.34)**	**0.40 (0.32)**	**24.9 (10.0)**	**16.1 (11.5)**	**18.8 (15.7)**
RBF-iso	1.03 (0.67)	**0.48 (0.34)**	0.74 (0.57)	31.9 (17.2)	**16.1 (11.5)**	36.3 (29.6)
RQ-iso	**0.79 (0.66)**	**0.50 (0.34)**	0.80 (0.51)	**25.5 (18.8)**	**17.1 (13.2)**	39.6 (27.3)
RBF-ard	1.08 (0.71)	**0.52 (0.36)**	0.84 (0.56)	33.9 (18.0)	**18.7 (16.4)**	39.9 (26.8)
RQ-ard	**0.79 (0.66)**	**0.51 (0.33)**	0.81 (0.50)	**25.5 (18.4)**	**18.1 (14.5)**	39.7 (25.9)
greedy	0.99 (0.74)	**0.72 (0.67)**	0.75 (0.80)	32.6 (23.8)	**24.4 (22.6)**	38.1 (42.3)

The test is run over 20 windows with three forecasting horizons $h = \{10, 20, 40\}$. The reason for using 20 windows is because if we look at the plot of both the 2-kernel and 4-kernel datasets, we can see that there are periods which last over more or less 20 time-units. Thus, by using 20 windows, we average out the forecast over a single period and assess whether the model could forecast consistently across a single period in the future. Due to this reasoning, $h = 40$ can be considered as a long-term forecast, because it spans for two periods in the signal, while $h = 10$ is short-term and $h = 20$ is medium-term forecast. The forecasting performance will be evaluated using the MAE, MAPE, and improvement metric. The improvement will be compared against the persistence method. Section 5.1.2 reviews those metrics in more details. It is important to remind that the test is run over several windows, hence the mean operation on MAE and MAPE is calculated over several windows. MAE value in the $h = 10$ column signifies the average absolute error that is calculated over 20 windows of forecast where the target is 10-step-ahead forecast.

Table 5.6 and Table 5.7 show the MAE and MAPE forecast evaluation for the 2-kernel and 4-kernel dataset. Furthermore, Table 5.8 reports the improvement over the persistence model for each of the forecasting model. Figure 5.9, 5.10, and 5.11 show the actual forecasting results in comparison to the true observed value per horizon. Similarly, Figure 5.12, 5.13, and 5.14 depict the comparison for the 4-kernel dataset.

Now, let us assess the forecasting performance for the 2-kernel dataset. It can be seen that the linear

Table 5.7: The forecasting performance of the 4-kernel dataset. MAE denotes the mean absolute error and MAPE signifies the mean absolute percentage error. Those performance measures are evaluated over 20 windows. Values marked in bold are not significantly different from the best-performing forecasting model, in a paired t-test with a p-value of 5%. Value denoted in the parentheses is the standard deviation.

	MAE			MAPE (%)		
model	$h = 10$	$h = 20$	$h = 40$	$h = 10$	$h = 20$	$h = 40$
persistence	9.36 (6.48)	10.02 (4.91)	8.10 (6.61)	85.1 (112.6)	67.6 (92.0)	34.1 (25.0)
ARIMA	6.19 (4.23)	7.22 (4.85)	10.07 (5.74)	61.1 (97.8)	44.0 (67.8)	51.5 (42.6)
BLR	0.65 (0.45)	1.70 (1.20)	5.67 (4.60)	7.3 (13.1)	13.5 (26.4)	26.4 (21.6)
RBF-iso	0.74 (0.45)	2.08 (1.38)	11.73 (6.10)	7.7 (11.4)	16.0 (28.0)	50.7 (17.5)
RQ-iso	0.72 (0.44)	1.91 (1.07)	7.03 (4.22)	7.5 (11.2)	14.1 (24.5)	33.4 (22.4)
RBF-ard	7.93 (9.11)	6.38 (6.29)	17.01 (7.05)	46.2 (44.7)	29.6 (28.8)	82.1 (38.5)
RQ-ard	0.86 (0.97)	2.44 (2.27)	7.81 (4.73)	4.6 (4.2)	12.0 (9.9)	42.9 (48.6)
greedy	**0.27 (0.20)**	**0.56 (0.56)**	**2.71 (1.79)**	**2.0 (2.1)**	**3.3 (4.2)**	**16.5 (23.6)**

Table 5.8: The improvements over the persistence method for both the 2-kernel and 4-kernel dataset. The improvement is in percentage and the larger the value, the better it is. A negative value means that the model is worse than the persistence method.

	2-kernel			4-kernel		
model	$h = 10$	$h = 20$	$h = 40$	$h = 10$	$h = 20$	$h = 40$
ARIMA	28.37	40.26	60.66	33.93	27.89	-24.34
BLR	18.04	41.92	70.37	93.11	83.02	29.95
RBF-iso	-4.67	41.63	44.73	92.08	79.27	-44.78
RQ-iso	19.47	40.09	40.02	92.27	80.93	13.16
RBF-ard	-10.46	36.91	37.48	15.28	36.32	-110.05
RQ-ard	19.58	38.01	39.85	90.81	75.66	3.55
greedy	-0.60	12.92	44.19	97.10	94.39	66.52

model like ARIMA and linear regression perform better than the GP-based methods. The ARIMA is the most consistent method throughout all horizons while linear regression shows the best performance for the $h = 40$ forecast. Looking at GP-based methods, we find that the GP-NAR performs well up to $h = 20$ only. The greedy shows disappointing results among others. Figure 5.9, 5.10, and 5.11 provide an insight of why these results are observed. If we look at the forecast trajectories of all GP-based methods, we can see that GP-based methods attempt to model the up and down in the signal, but sometimes they end up wrong, where for example the true signal is going up while the forecast goes down. This observation is especially apparent in the greedy method. In contrast, the linear model of ARIMA and linear regression 'play it safe' by outputting flat forecasts around the mean of the function, where the mean of the dataset is $\bar{y} = 2.65$. Since the function exhibits ups-and-downs around the mean, this can provide the explanation on why forecasts of ARIMA and linear regression are more accurate than the GP-based methods.

The second dataset, the 4-kernel dataset, shows a contrasting result where results report that the greedy is significantly better than other methods. The MAPE of $h = 10$ and $h = 20$ are remarkably lower in comparison with the MAPE from the 2-kernel dataset which suggests an excellent forecast. The results show that only ARIMA performs significantly worse for all horizons. Figure 5.12, 5.13, and 5.14 clearly show the cause. While the GP-based method and linear regression can model the trend and period of the true value, the ARIMA cannot. The output of ARIMA forecasts is roughly flat, which suggests a similar pattern of follow-the-mean. This follow-the-mean strategy that was working nicely in the 2-kernel dataset apparently does not work in this dataset due to the presence of a strong trend. In $h = 40$ forecast, as it is shown in Figure 5.14, only the greedy method that can still follow the pattern

decently. The result of ARIMA's failure on the 4-kernel dataset can be explained by the non-stationarity of the 4-kernel dataset.

In the end, the 2-kernel dataset proves to be harder to forecast than the 4-kernel dataset, judging by the MAPE score. If we look at the discrepancy between results of 2-kernel and 4-kernel, we can draw insight that the greedy approach works particularly well when there are a clear trend and period in the dataset. When such pattern is apparent, the greedy method can forecast very far. The 40-step-ahead forecasting performance using the greedy method is very decent, and that is quite impressive considering we have only 150 number of observation in total. On the contrary, the performance of GP-based methods on the 2-kernel is worse than the simpler method of ARIMA and linear regression although in general, all models are struggling. The reason for this is because the 2-kernel dataset does not have a clear structure that the GP model can learn.

5.6. Test 3: Real World Dataset Forecasting Performance

Table 5.9: The forecasting performance of the wind speed dataset. MAE denotes the mean absolute error and MAPE signifies the mean absolute percentage error. Those performance measures are evaluated over 24 windows. Values marked in bold are not significantly different from the best-performing forecasting model, in a paired t-test with a p-value of 5%. Value denoted in the parentheses is the standard deviation.

	MAE			MAPE (%)		
model	$h = 12$	$h = 24$	$h = 48$	$h = 12$	$h = 24$	$h = 48$
persistence	0.45 (0.32)	0.67 (0.27)	0.72 (0.54)	18.3 (13.0)	25.2 (10.7)	40.0 (45.4)
ARIMA	0.41 (0.32)	0.66 (0.28)	0.65 (0.53)	**16.7 (13.0)**	24.7 (10.5)	36.2 (44.6)
BLR	0.43 (0.28)	0.65 (0.24)	**0.56 (0.44)**	16.9 (10.0)	23.8 (7.6)	**30.5 (35.2)**
RBF-iso	**0.40 (0.34)**	0.50 (0.21)	0.77 (0.58)	**16.2 (13.7)**	18.7 (7.2)	48.7 (59.5)
RQ-iso	0.47 (0.29)	0.47 (0.33)	**0.72 (0.44)**	19.1 (12.1)	17.6 (12.4)	40.8 (35.7)
RBF-ard	0.39 (0.27)	0.47 (0.23)	**0.65 (0.51)**	**15.6 (9.7)**	17.6 (9.1)	40.2 (47.8)
RQ-ard	**0.36 (0.29)**	0.57 (0.32)	**0.70 (0.43)**	**14.4 (12.3)**	21.8 (12.7)	41.2 (40.3)
greedy	**0.27 (0.15)**	**0.29 (0.18)**	1.22 (0.64)	**11.6 (7.2)**	**11.3 (7.9)**	74.0 (69.6)

Table 5.10: The forecasting performance of the electricity dataset. MAE denotes the mean absolute error while MAPE signifies the mean absolute percentage error and smaller value is also better. Those performance measures are evaluated over 24 windows. Values marked in bold are not significantly different from the best-performing forecasting model, in a paired t-test with a p-value of 5%. Value denoted in the parentheses is the standard deviation.

	MAE			MAPE (%)		
model	$h = 12$	$h = 24$	$h = 48$	$h = 12$	$h = 24$	$h = 48$
persistence	**9.32 (7.26)**	10.55 (7.89)	**10.54 (8.99)**	**2.7 (2.1)**	3.1 (2.2)	**3.1 (2.5)**
ARIMA	17.50 (15.59)	17.56 (14.65)	32.58 (18.54)	5.0 (4.0)	5.0 (3.7)	10.2 (6.5)
BLR	**7.46 (7.57)**	9.10 (7.70)	26.94 (12.97)	**2.1 (2.2)**	2.6 (2.0)	8.1 (3.7)
RBF-iso	**7.58 (7.41)**	**8.10 (7.08)**	20.40 (18.82)	**2.2 (2.1)**	**2.3 (1.9)**	6.1 (5.3)
RQ-iso	**7.58 (7.41)**	**9.16 (10.45)**	**12.06 (7.52)**	**2.2 (2.1)**	**2.7 (3.0)**	**3.6 (2.2)**
RBF-ard	**8.09 (7.17)**	**8.78 (6.63)**	16.71 (10.52)	**2.3 (2.0)**	2.5 (1.8)	5.2 (3.4)
RQ-ard	**7.69 (6.29)**	10.32 (6.58)	15.86 (9.64)	**2.2 (1.8)**	3.0 (1.8)	4.9 (3.2)
greedy	**7.10 (6.43)**	**7.02 (6.42)**	22.81 (9.96)	**2.0 (1.9)**	**2.0 (1.7)**	6.9 (3.2)

This section reports the result of the experiment done on two real-world datasets, the *wind speed* and *electricity* dataset. Similar to the experiment on the synthetic dataset, the forecast will be run on several windows. This time, 24 windows are used. The *wind speed* has one hour while the *electricity* dataset has one month as a time unit. A 24-window test suggests that the training is performed on one day

Table 5.11: The improvement over persistence method for both the wind speed and electricity dataset. The improvement is in percentage and the larger the value, the better it is. A negative value means that the model is actually worse than the persistence method.

	wind speed			electricity		
model	$h = 12$	$h = 24$	$h = 48$	$h = 12$	$h = 24$	$h = 48$
ARIMA	8.35	0.71	10.06	-87.82	-66.50	-209.01
BLR	4.60	2.80	22.72	19.99	13.68	-155.58
RBF-iso	9.92	24.38	-7.50	18.66	23.18	-93.54
RQ-iso	-4.21	29.84	0.08	18.69	13.18	-14.38
RBF-ard	13.49	29.87	9.44	13.14	16.80	-58.47
RQ-ard	20.61	14.17	2.19	17.44	2.12	-50.42
greedy	39.99	57.07	-69.70	23.79	33.41	-116.33

and one year of data respectively. This test has the benefit of averaging out the variations that occur during the whole day and year. The target horizons are $h = \{12, 24, 48\}$. 48-step-ahead forecast is a quite challenging horizon in both tasks. In wind speed forecasting, $h = 24$ and $h = 48$ mean one day and two days of forecasting horizons respectively and can be classified as long-term forecasts according to Table 1.1. A year of time series ($h = 24$) in the electricity dataset already exhibits at least two seasonal patterns and an increasing trend, hence forecasting $h = 24$ and $h = 48$ should be a demanding forecasting goal.

Following the previous test, the forecasting performance is evaluated by the MAE, MAPE, and improvement over persistence method. Table 5.9 displays the MAE and MAPE results of wind speed forecasting and Table 5.10 for the electricity dataset. Table 5.11 shows the improvement results from both the wind speed and the electricity datasets. Figure 5.15, 5.16, and 5.17 show the actual forecasting results on the wind speed dataset in comparison to the actual observed value per horizon. Similarly, Figure 5.18, 5.19, and 5.20 depict the comparison for the electricity dataset.

Results of electricity forecasting confirm our finding from the previous section that the greedy method forecasts remarkably in light of apparent trend and seasonal patterns, at least for $h = 12$ and $h = 24$ forecasts. This time, the GP-NAR methods also perform well, especially for the $h = 48$ forecast where no other forecasting methods are significantly better than the GP-NAR with isotropic RQ kernel. The decreasing result of the greedy method for $h = 48$ is explained in the Section 5.7, where we decompose the forecasting result. ARIMA also shows similar performance with the 4-kernel dataset forecasting, which verifies the findings from the previous test that the ARIMA cannot handle the trend and seasonality very well, as depicted in Figure 5.18, 5.19, and 5.20.

The wind speed dataset can be considered as the most challenging dataset in this experiment. From the plot on Figure 5.3, we can see that the wind speed consists of periods of high and low that are not easily predicted as they are changing over time. Moreover, this dataset also exhibits some small scale variations that vary within days. The result of the greedy method for $h = 12$ and $h = 24$ invalidates our analysis from the previous section that the greedy model only excels on an 'easy' dataset with a predictable pattern. The explanation for this outcome can be explained by the learning curve which is displayed on Figure 5.7. Here it can be seen that the greedy method reaps the most benefit from more data in comparison with other methods. Since the greedy method is based on finding trend and seasonality, the more the data we have, the higher chance that we can find those long-term patterns. As an illustration, imagine that there is a trend that we find on a time series dataset. If we expand the time series to the past, we may discover that the trend is a slow moving season. So, we can conclude that the structure modelling approach reaches its best capability when there are significant long-term structures in the time series and having more data can help uncover those patterns.

For more distant forecasting horizons, which is represented by $h = 40$ and $h = 48$ in our experiment, regression techniques display superior performance over the greedy method. Surprisingly, the linear regression is the best performing method for the 40-step-ahead forecast in the 2-kernel dataset and

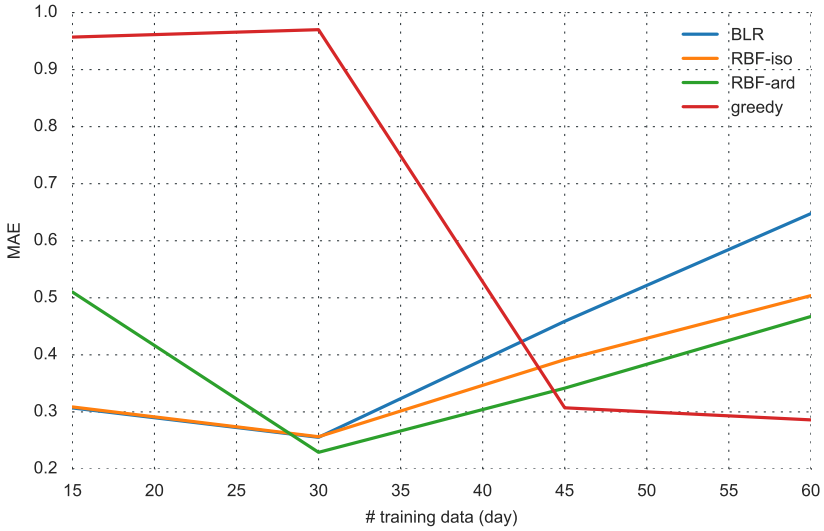


Figure 5.7: The learning curve that shows the 24-step-ahead forecasts error as a function of the number of training data. The dataset used for this curve is the wind speed dataset, and the number of training data is measured in days. The error is measured in mean absolute error (MAE).

48-step-ahead forecast in the wind speed dataset, which can be considered as the hardest forecasting tasks among the rest. There is not enough evidence to elucidate the reason behind this occurrence, but judging from Table 5.6 and 5.9, there is a notable gap between the performance between the MAPE value of 2-kernel (all horizons) and wind speed (particularly the $h = 48$) with the rest. These results indicate that these forecasting tasks are too difficult for our proposed forecasting method to learn due to the absence of pattern and growing uncertainty.

The main building blocks of the GP-NAR are choosing between RBF and RQ kernel and assigning either a single lengthscale (isotropic) or different lengthscale parameter (ARD) to the input. There is no definitive choice for either as each option shows mixed performance score, although in general it is found that GP-NAR with isotropic kernel works slightly better than the ARD kernel across different horizons. This verdict contradicts the hypothesis in Section 4.3 that the ARD kernels should have better forecasting performance than isotropic kernels because in the ARD kernel the weights are assigned to each lag. As a consequence, the ARD kernel requires more parameters than the isotropic, which makes the ARD kernel more susceptible to overfitting than the isotropic one. In that case, we might want to choose the isotropic kernel over ARD because of the reason that was mentioned previously.

5.7. Decomposition

Several experiments in the previous sections focused on forecasting performance. No dominant method surpasses other methods, but the structure modelling method shows a promising result, especially in the case of an evident pattern. One other thing that makes structure modelling approach desirable is it presents a relatively interpretable model.

This section illustrates how we can interpret the result of the forecast of the electricity dataset. After the search, the optimum kernel combination is shown below, with the explanation of the effect of the kernel to the function.

$$\underbrace{\text{RBF} \times \text{Per}}_{\text{slow-varying trend with periodic}} + \underbrace{\text{RBF}}_{\text{short-term variation}} + \underbrace{\text{RBF}}_{\text{residual}}$$

The full expression of the kernel combination is given below

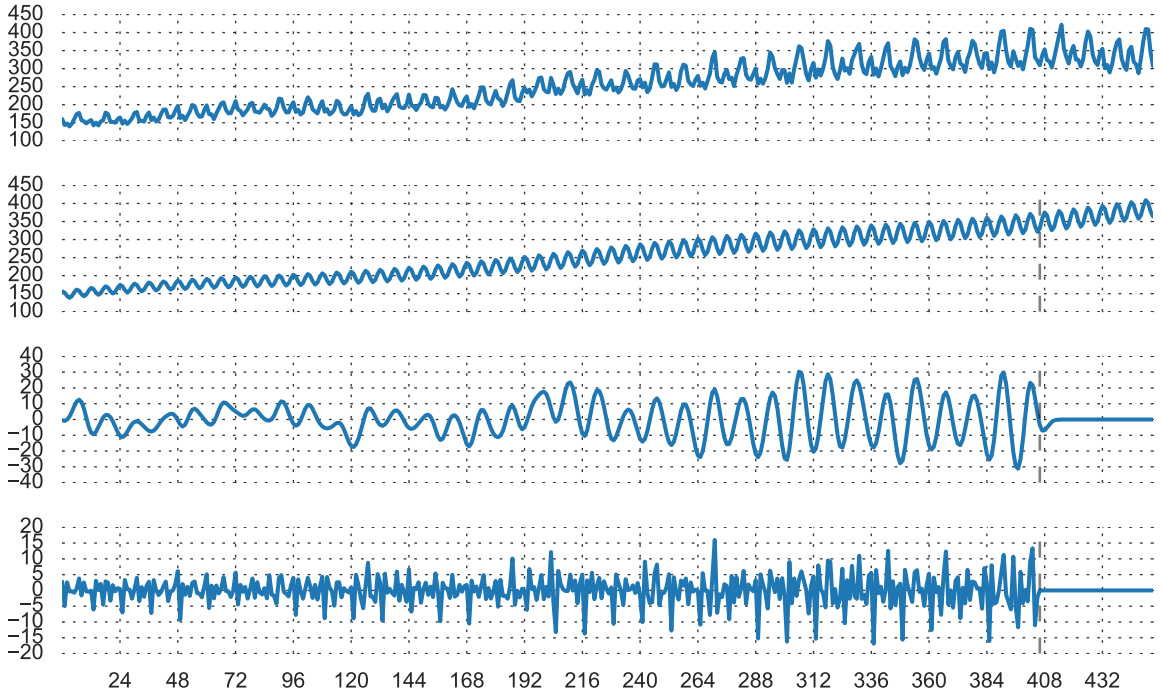


Figure 5.8: The additive decomposition of the electricity dataset. The vertical dashed line denotes the boundary between the training and test set. First row: actual dataset. Second row: RBF × Per component. Third row: Lin + RBF, Fourth row: RBF component.

$$k(t,t') = \sigma_1^2 \exp\left(-\frac{(t-t')^2}{2\lambda_1^2}\right) \times \sigma_2^2 \exp\left(-\frac{\sin^2(\pi(t-t')/p)}{2\lambda_2^2}\right) + \sigma_3^2 \exp\left(-\frac{(t-t')^2}{2\lambda_3^2}\right) + \sigma_4^2 \exp\left(-\frac{(t-t')^2}{2\lambda_4^2}\right) \quad (5.6)$$

where $\sigma_1 = 613.6, \lambda_1 = 191.6, \sigma_2 = 689.3, \lambda_2 = 47.3, p = 0.32, \sigma_3 = 205.4, \lambda_3 = 2.71, \sigma_4 = 41.1, \lambda_4 = 0.01$.

Figure 5.8 displays the additive decomposition of electricity dataset according to the kernel that we found. The first additive component, $\mathrm{RBF} \times \mathrm{Per}$, explains the most dominant feature of the time series, and that is the increasing trend with small period periodic. The trend is caused by the RBF kernel, where the lengthscale is $\lambda_1 = 191.6$. This RBF kernel creates an increasing signal with a slow moving variation which looks like a trend, and when this signal is multiplied by the periodic kernel, we get a periodic signal that is shown in the second row of Figure 5.8. Compared with the rest, this signal is the most dominant, judging by the vertical scale of the signal. Apart from the $\mathrm{RBF} \times \mathrm{Per}$, the other RBF kernels model the variation within the the time series. From the lengthscale, $\lambda_3 = 103.9$, this signal models a small-term variation, where the signal varies every three months. The last component, another RBF kernel, explains a short-lived variation that looks like noise, which can be interpreted from the lengthscale of this kernel which is $\lambda_4 = 0.01$.

Figure 5.8 clarifies the reason why the performance of the structure modelling drops for the $h = 48$ forecast. If we look at the decomposition, we can see that only the RBF × Per component that can extrapolate far ahead. The rest of the components consist of RBF kernels with short lengthscales. Because of that, the forecast shows a mean convergence symptom that is displayed in Section 2.5 after a particular horizon. Therefore, the only available explanation for longer term forecast is the first component. This component does not provide enough predictive power to explain any variability that

occurs in the far future. It can be seen from the first row of Figure 5.8 that the increasing trend slows down after a certain horizon and the first component cannot anticipate this change of pattern.

This decomposition experiment demonstrates the interpretability of the structure modelling model. This feature is very useful beyond forecasting. We can perform a time series decomposition using the structure modelling method and present this technique as an alternative to the other decomposition method such as the STL [12]. Lloyd et al. even take this approach ahead, by proposing an automated reporting of features in a time series using natural language processing through interpreting the parameter of the kernels that are returned by the structure modelling method [33].

5.8. Training Time

Table 5.12: Training time of Gaussian process regression model against ARIMA and linear regression

	training time (second)				
# of training data	ARIMA	BLR	RBF-iso	RBF-ard	greedy
100	1.03	0.01	0.43	1.05	152.67
200	0.28	0.01	0.75	2.59	1148.56
300	0.37	0.01	1.58	6.76	845.41
400	0.33	0.01	2.54	11.20	1341.42
500	0.57	0.01	4.87	17.49	2540.50
600	0.51	0.01	6.59	22.50	7967.58
700	0.51	0.01	8.97	30.56	5407.02
800	0.63	0.01	16.75	46.03	11924.86
900	0.66	0.13	28.80	65.39	11390.19
1000	0.69	0.04	25.50	80.27	12061.39

Table 5.12 shows the time that is required to train a GP-NAR and greedy method in comparison with ARIMA and Bayesian linear regression. The greedy method is trained on 5-level search depth, while both the GP-NAR and Bayesian linear regression use autoregressive of order 5. From the table it is clearly apparent that the greedy method requires a significant amount of time to train, far exceeding other methods.

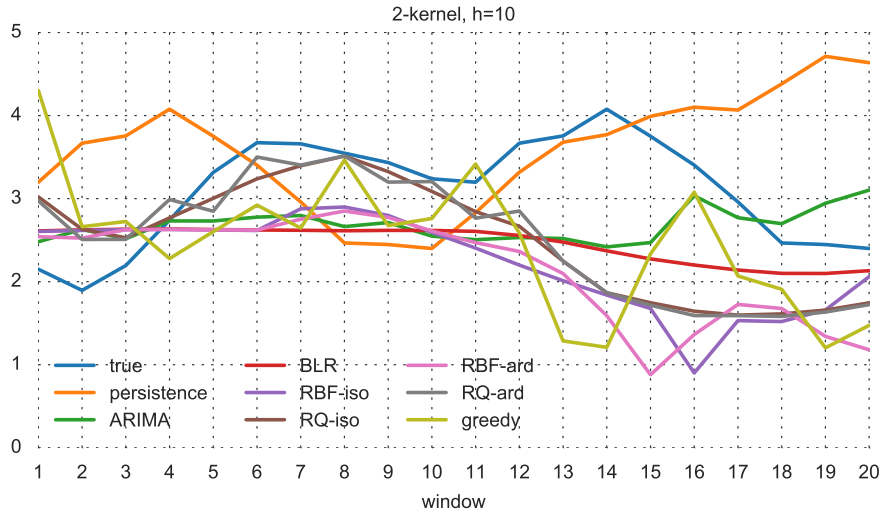


Figure 5.9: Plot of 10-step-ahead forecast results on the 2-kernel dataset over 20 windows

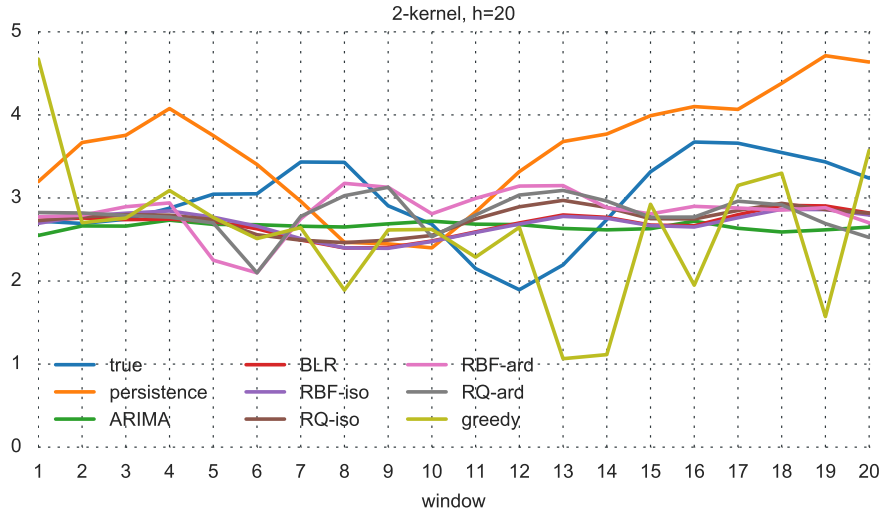


Figure 5.10: Plot of 20-step-ahead forecast results on the 2-kernel dataset over 20 windows

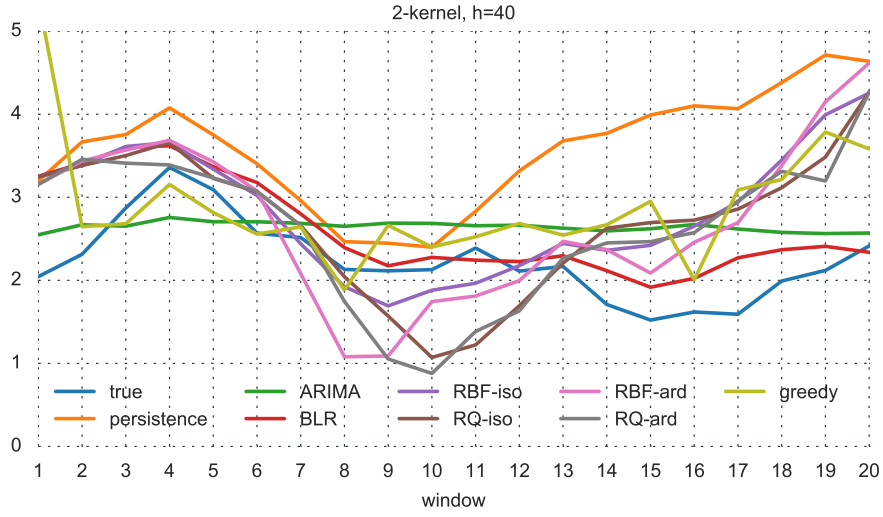


Figure 5.11: Plot of 40-step-ahead forecast results on the 2-kernel dataset over 20 windows

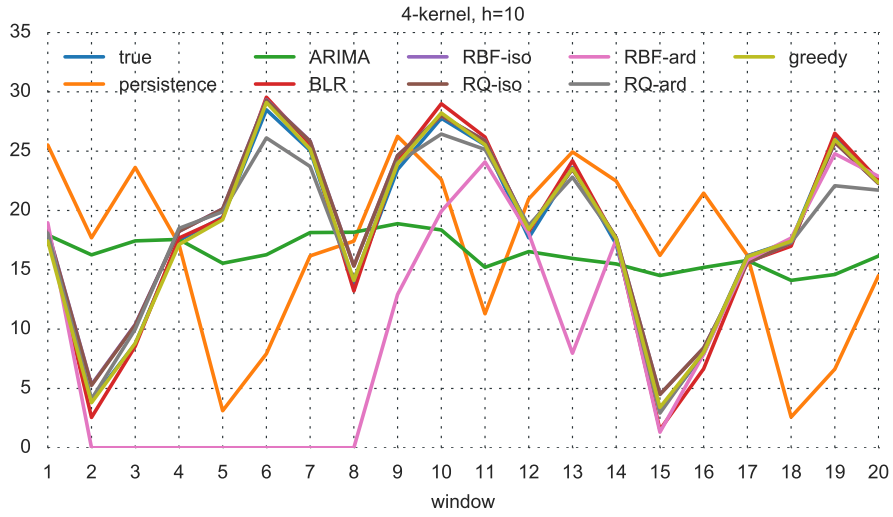


Figure 5.12: Plot of 10-step-ahead forecast results on the 4-kernel dataset over 20 windows

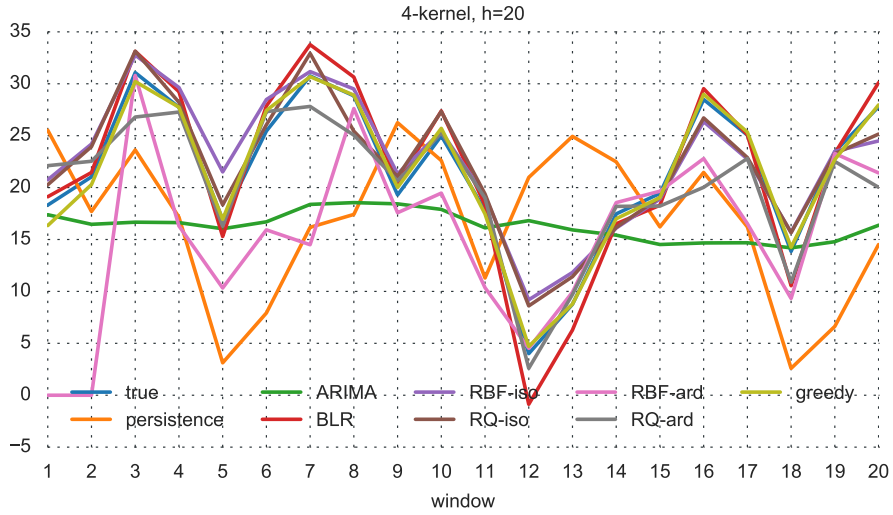


Figure 5.13: Plot of 20-step-ahead forecast results on the 4-kernel dataset over 20 windows

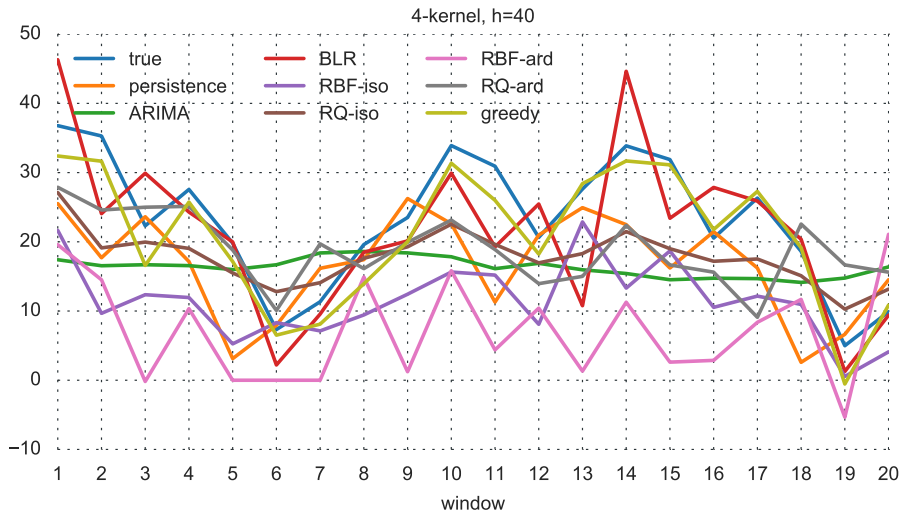


Figure 5.14: Plot of 40-step-ahead forecast results on the 4-kernel dataset over 20 windows

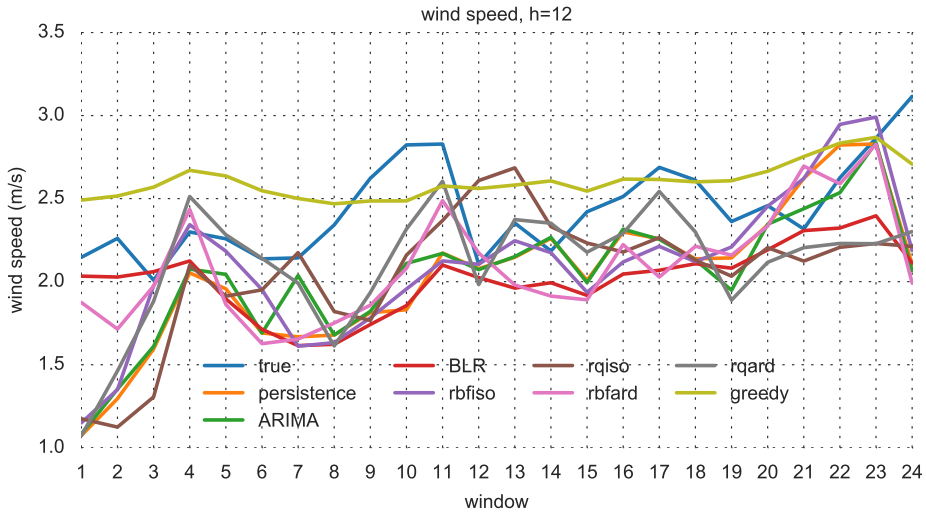


Figure 5.15: Plot of 12-step-ahead forecast results on the wind speed dataset over 24 windows

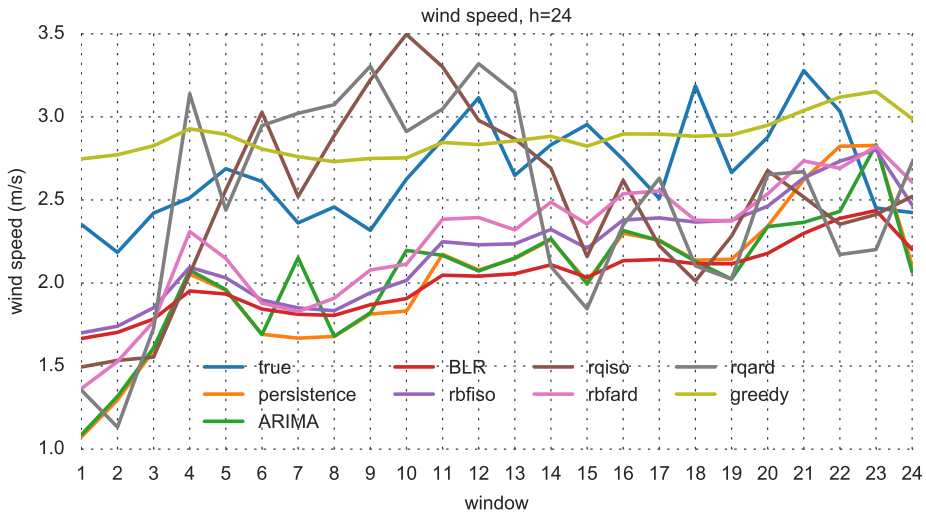


Figure 5.16: Plot of 24-step-ahead forecast results on the wind speed dataset over 24 windows

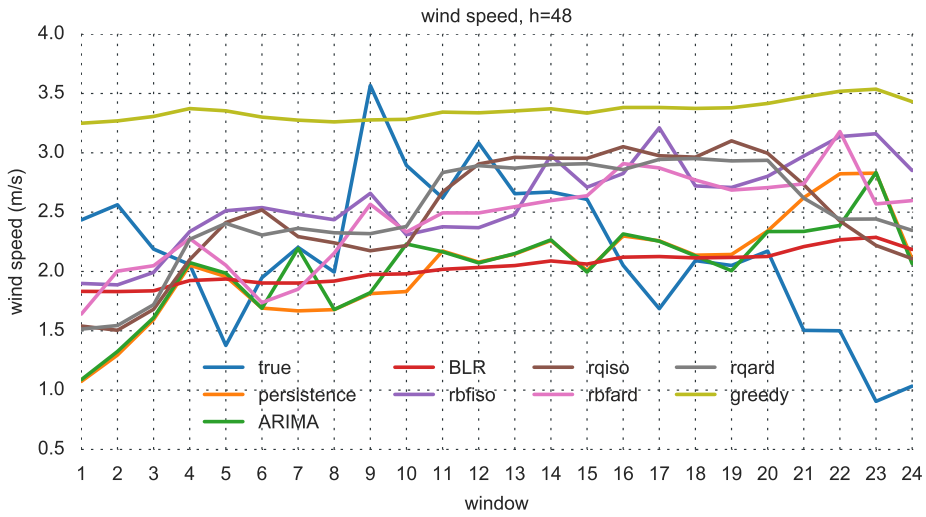


Figure 5.17: Plot of 48-step-ahead forecast results on the wind speed dataset over 24 windows

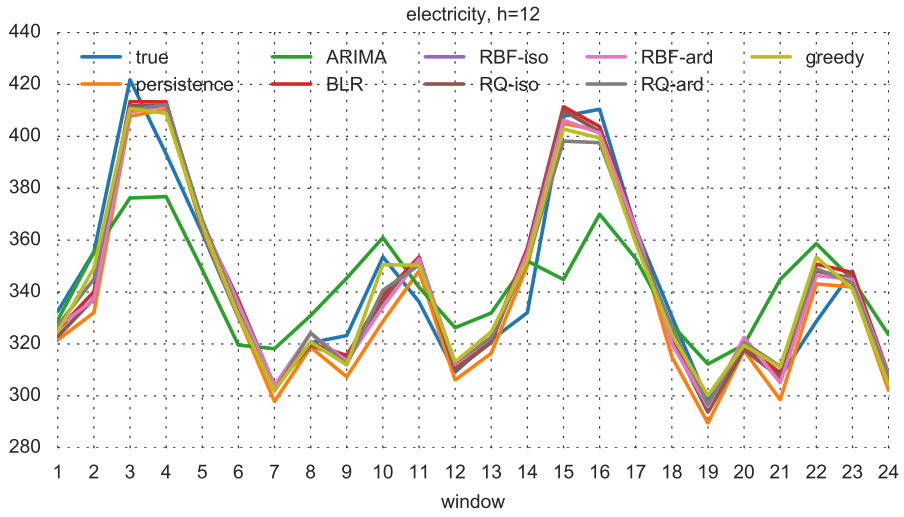


Figure 5.18: Plot of 12-step-ahead forecast results on the electricity dataset over 24 windows

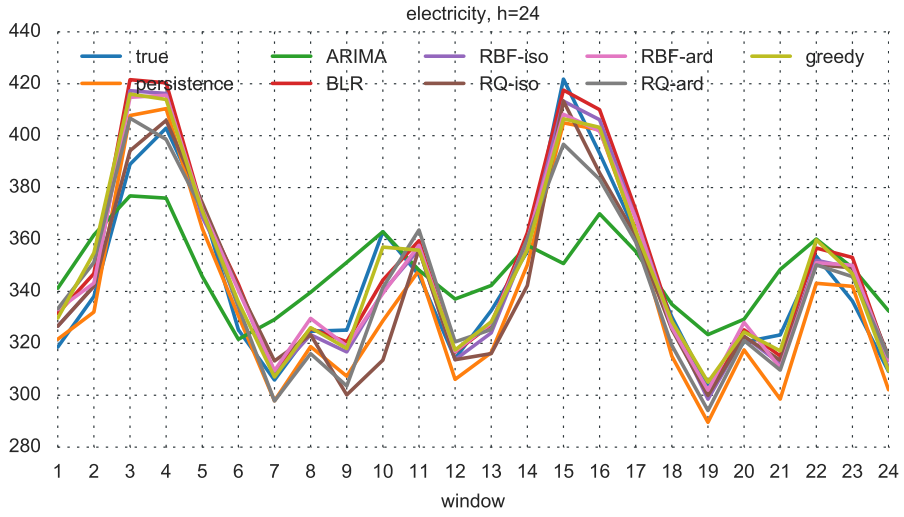


Figure 5.19: Plot of 24-step-ahead forecast results on the electricity dataset over 24 windows

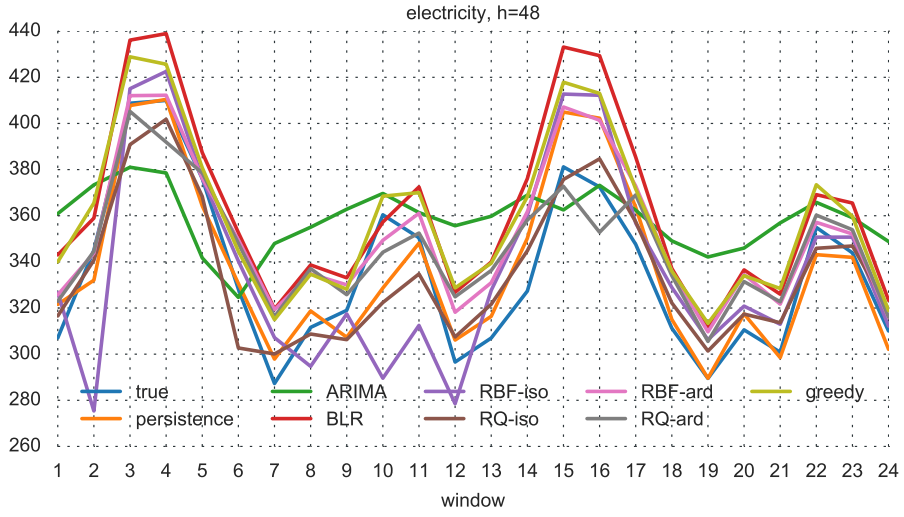


Figure 5.20: Plot of 48-step-ahead forecast results on the electricity dataset over 24 windows

6

Conclusion and Future Work

6.1. Conclusion

Long-term time series forecasting is very useful in many application domains, yet it is a challenging task to perform. As time series forecasting becomes our problem, we might want to use the ARIMA model, which is one of the most prominent time series forecasting models. Unfortunately, the ARIMA suffers from a mean convergence issue which hampers its performance in long-term time series forecasting. Hence, we turn our attention to the Gaussian process regression, a non-parametric probabilistic regression technique. Due to its non-parametric property, a Gaussian process regression possesses a more flexible modelling capability than the ARIMA model. With this flexibility, we hypothesise that the GP regression can be a more potent solution for long-term time series forecasting problem than the ARIMA. This research is among the first to empirically assess the performance of the ARIMA and Gaussian process regression.

We employ two specific techniques that are based on the Gaussian process regression to solve our long-term time series forecasting problem: the structure modelling and the autoregressive. These two methods have been discussed in detail in Chapter 3 and Chapter 4. This research is then followed by running several long-term forecasting experiments to empirically evaluate the long-term performance of our proposed methods against the ARIMA method as can be observed in Chapter 5. Our experiments are conducted to provide the answer to our proposed research questions as follows.

RQ1: Between the two Gaussian process regression approaches, the autoregressive and the structure modelling, how do their long-term forecasting performances compare with the ARIMA model?

The GP-based approaches outperform ARIMA in a particular case where the time series exhibits a clear trend and seasonal structures. In fact, the ARIMA method is the worst method in general, especially when the data display some strong non-stationarities. Both the GP-based approaches as well as the other models do not show an exemplary result on challenging tasks. This cases are marked by the large gap in the MAPE performance score. Examples of such circumstances are the 48-step-ahead forecast on the wind speed and the 40-step-ahead forecast on the 2-kernel dataset. These datasets do not contain a clear trend and season which can be exploited by the forecasting model. The exception is made to the Bayesian linear regression where it surprisingly shows decent forecasting performances in these difficult problems. Even so, the difference in performance between the easy and difficult problems suggests that improvements are necessary. In the end, we can conclude that the GP-based approaches are highly recommended over the ARIMA for long-term time series problem where the regularities are evident although we must take into account the length of the horizon as there is a limit on how far the GP methods can take. In addition to that, we should also consider the performance issue of the structure modelling, as it requires a significant amount of time to train in comparison with ARIMA.

RQ2: Between the two Gaussian process regression approaches, the autoregressive and the

structure modelling, which has better long-term forecasting performance?

There is no clear winner shown between autoregressive (GP-NAR) and the structure modelling approach. The general performance of structure modelling method is preferable, except for a more challenging long-term forecasting case such as 48-step-ahead forecasting of the electricity data, where the GP-NAR outperforms the structure modelling method. Nonetheless, the structure modelling approach has potential that the GP-NAR does not have: the interpretability of the model. This proportion makes the structure modelling method unique. Several domains, such as economic and finance, prefer a more transparent statistical model to black-box one as it can explain why the model can come up with a particular prediction. An interpretable model might help with the acceptance of the model by the human user. In contrast with the structure modelling, the autoregressive setting of GP-NAR is a general set-up that is not exclusive to the GP-NAR technique. With the autoregressive representation, one can easily interchange the GP-NAR with other machine learning regression technique, such as neural network or support vector regression. By this narration, we believe that the structure modelling method has a better potential to be further developed.

Besides these conclusions, we present several discussion points that are discovered during our research.

Uncertainty Quantification The Gaussian process regression and other probabilistic models output a predictive distribution. In the experiment, we focus more on evaluating the performance by comparing the mean of the distribution where the mean acts as a point forecast. A point forecast is, of course, useful on its own, but utilising the probabilistic model comes with a benefit that a non-probabilistic model does not have which is the ability to compute the uncertainty of the forecast.

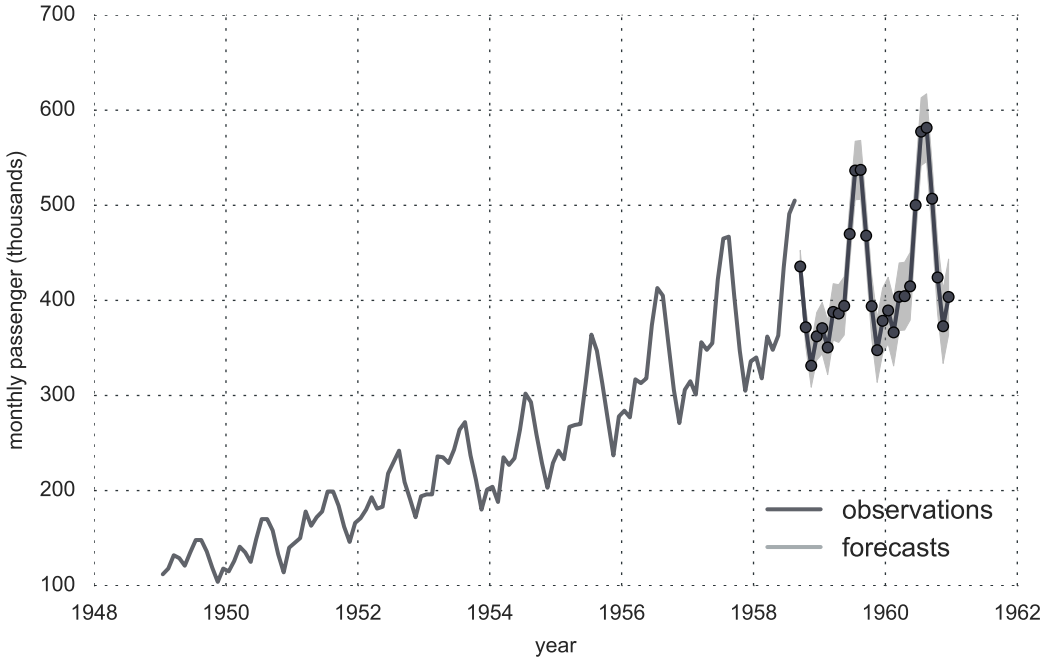


Figure 6.1: A forecasting result with the uncertainty represented by the shaded region. Notice that some future data have lower uncertainty even though they are ahead in the future. Data: Monthly totals (in thousands) of international airline passengers from 1949-1960 [7]

For example, we can quantify the uncertainty of our forecasts by computing the 95% confidence interval. The confidence interval is calculated from the variance of the forecast, which is the diagonal element of the covariance matrix that is obtained using Equation 2.18. Figure 6.1 shows the illustration of the point forecasts and its 95% confidence interval. This confidence interval quantifies the uncertainty of our prediction and can be useful in many applications. Silver writes one

particular example where uncertainty information is neccesary [49]. In April 1997, a small town in North Dakota, the United States of America, named Grand Forks is drowned by floods because a nearby river overtopped the town's levee. The flood had already been anticipated through the forecast, where the forecast stated that the height of the water level was 49 feet, 2 feet below the height of the levee, which was 51 feet. In turned out that the margin of error of the forecast was about ± 9 feet and it was not communicated to the townspeople. This miscommunication left them unprepared when the real flood was above the town's levee but still fell between the uncertainty margin. Had the uncertainty was computed and reported, the flood can be anticipated. This example signifies the importance of calculating the uncertainty of forecast, and GP regression is capable of doing this.

Computation Time The computation time becomes the main weakness of the proposed GP methods. This issue is particularly noticeable for the structure modelling with the structure modelling method, where it requires 200 minutes to train a forecasting model on a time series dataset with 1000 observations. This result is clearly unacceptable when an ARIMA model only needs 0.69 seconds to train on the same data. So, although the structure modelling can outperform the ARIMA model concerning the forecasting performance, this computation issue is the biggest obstacle to the adoption of the structure modelling method as an alternative to ARIMA. The reason behind this problem is contributed by the fact that training a GP regression model requires us to compute a matrix inverse through a Cholesky decomposition (Section 2.4) which has $\mathcal{O}(n^3)$ computation complexity. This fact points out a future problem that must be solved.

Inefficient Search Previously it was mentioned that the structure modelling has a severe performance issue. The greedy search strategy and stopping criterion have considerably help to reduce the computation time in comparison with using the exhaustive search. Still, the current greedy search is not the most efficient search. One example of the inefficiency is shown in Section 5.4. In one test it is found that the search keeps adding a linear kernel although it is meaningless as adding linear kernels result in another linear kernel. Another inefficiency of the current greedy search is shown in Section 5.7. In that section, it is reported that not all kernel combinations can model the long-term forecast. Only the combinations that involve linear, periodic, and RBF with long lengthscale can contribute to the long-term forecast. If we can remove these irrelevant kernels from the search tree, the search could be more efficient.

User-defined Parameter The proposed GP-based methods require a few user-defined parameters. For the structure modelling, several parameters have already determined by design, such as the base kernels. The experiment suggests that the model selection based on the BIC is more appropriate than the cross-validation as it provides superior forecasting accuracy. This left us with the decision of choosing the depth of the search level. The search level depends on the complexity of the time series to be modelled. One thing to be wary of is that the higher the search level, the longer it is to run a kernel search, so the search level should be kept minimum. The parameters GP-NAR are the choice of the kernel (RBF or rational quadratic) and the decision to choose between the isotropic or ARD kernel structure. The kernel option is problem dependent, so it is suggested that both kernel should be tried. The experiment shows that isotropic kernel generally has slightly better forecasting performance than the ARD which means that the isotropic kernel should be prioritised.

6.2. Future Work

In the previous section, we have concluded and discussed several aspects of our research, where some future improvements are hinted. Below are some of the future directions that can be pursued.

Hybrid Approach Rather than choosing between the structure modelling and GP-NAR, an idea of future research is to combine both approaches. The time series is considered to be composed from two additive components: the interpretable component (i_t) and the residual (e_t) component

$$y_t = i_t + e_t$$

where we forecast the interpretable component using the structure modelling and delegates the

forecast of the residual to the GP-NAR. With this hybrid approach, we still retain the interpretability of the structure modelling while being able to forecast the unexplainable pattern with the help of GP-NAR.

Fast Approximation to Gaussian Process One main obstacle to push the GP-based method as a replacement for the ARIMA model is the computation time. Some methods are developed to approximate the inversion of the covariance matrix for a common GP regression. The paper by Quinonero-Candela and Rasmussen [42] provides a detailed review of the approximation methods. The approximation technique revolves around the idea of using m data points to approximate the covariance matrix instead of n, where $m < n$. Some techniques can bring the complexity down to $\mathcal{O}(nm^2)$. These methods have the potential as approximation techniques for the GP-NAR approach. For a GP regression with time index as input, there exists an approximation method where the computational complexity is linear with the number of data ($\mathcal{O}(n)$). This technique is proposed by Hartikainen and Särkkä, where they transform the GP regression problem into a linear-Gaussian state space model, which can be solved exactly using the Kalman Filter [24]. The challenge here is to reformulate the covariance function by applying a spectral Taylor series approximation. They already implement the Taylor approximation for the RBF kernel, therefore if the same approximation can be applied to the linear and periodic kernel, then this fast approximation technique can be utilised by our structure modelling approach.

Multivariate Time Series Learning the future solely from the past is the simplest form of time series forecasting. We have seen some difficult forecasting cases from the experiment, for example the 48-step-ahead forecast of wind speed. Here we see the inadequacy of a model that learn solely from the past. While improving the model can be one solution to this problem, another potential solution is by adding external information from exogenous variables aside from the past data of our time series. These external variables can be deterministic, such as the name of the day or a boolean variable that indicate public holiday. A more complicated model utilises another time series, which might be correlated with the target time series. In the case of wind speed forecasting, wind speed observations from nearby sensors or other weather observations such as wind direction and atmospheric pressure can be employed. Implementing the multivariate time series forecasting on the GP-NAR should be straightforward. Any additional features can be added as columns to the input matrix $\mathbf{X}$. If the exogenous variables are another time series, then those time series can be transformed into the autoregressive representation.

Search Heuristic In the discussion we have outlined the inefficiency of the greedy search. We can implement several heuristics that can reduce the search space of the greedy algorithm. One possible heuristic is to stop evaluating the meaningless combination, such as the addition of linear kernels. Another heuristic that can be useful is to focus solely on the covariance functions that are capable of modelling the long-term pattern and ignoring the short-term and medium-term effect.

A

Mean Convergence for ARIMA

This appendix is written to show that the ARIMA model will converge to the mean for longer horizon. The derivation is shown for ARIMA$(1,0,1)$, although it should generalise for all ARIMA(p,d,q). Given a time series $\mathbf{y} = \{y_t\}$, where $t = 1, \ldots, T$, we wish to predict a h-step-ahead forecast, y_{T+h}, using ARIMA$(1,0,1)$, $h = 1, 2, \ldots$. We assume that the time series is stationary. The ARIMA$(1,0,1)$ is given by

$$y_t = c + \epsilon_t + \phi y_{t-1} + \theta \epsilon_{t-1} \qquad \epsilon \sim \mathcal{N}(0, \sigma^2)$$

and the mean or the expected value of the ARIMA$(1,0,1)$ is

$$\begin{aligned} E[y_t] &= E[c] + E[\epsilon_t] + \phi E[y_{t-1}] + \theta E[\epsilon_{t-1}] \\ &= c + 0 + \phi E[y_t] + 0 \end{aligned}$$

$E[y_t] = E[y_{t-1}]$ because the time series is stationary. Let $E[y_t] = \mu$, then

$$\begin{aligned} \mu &= c + \phi \mu \\ &= \left(\frac{c}{1 - \phi} \right) \end{aligned}$$

1-step-ahead forecast, $h = 1$, is given by

$$y_{T+1} = c + \phi y_T + \theta \epsilon_T$$

Similarly, 2-step-ahead forecast ($h = 2$) is

$$\begin{aligned} y_{T+2} &= c + \phi y_{t+1} \\ &= c + \phi(c + \phi y_T + \theta \epsilon_T) \\ &= (1 + \phi)c + \phi^2 y_T + \phi \theta \epsilon_T \end{aligned}$$

and for $h = 3$

$$\begin{aligned} y_{T+3} &= c + \phi y_{t+2} \\ &= (1 + \phi + \phi^2)c + \phi^3 y_T + \phi^2 \theta \epsilon_T \end{aligned}$$

From the previous three forecast equations, we can derive a general ARIMA$(1, 0, 1)$ model for all h, which is given by

$$y_{T+h} = (1 + \phi + \phi^2 + \ldots + \phi^{h-1})c + \phi^h y_T + \phi^{h-1} \theta \epsilon_T$$

Since the time series is stationary, then $|\phi| < 1$ (refer to Shumway and Stoffer for more detail [48]).

For $h \to \infty$, $\phi^h = 0$ thus the expected value of a h-step-ahead forecast is equal to

$$\begin{aligned} E[y_{T+h}] = \mu = E[(1 + \phi + \phi^2 + \ldots + \phi^{h-1})c] \\ = (1 + \phi + \phi^2 + \ldots + \phi^{h-1})c \\ = \frac{c}{1 - \phi} \end{aligned}$$

as

$$\lim_{h \to \infty} (1 + \phi + \phi^2 + \ldots + \phi^{h-1})c = \frac{c}{1 - \phi} \qquad \text{for } |\phi| < 1$$

This shows that for $h \to \infty$, the forecast will converge to the mean.

B

Bayesian Model Selection for Gaussian Process

Bayesian model selection is an important concept which underlies two aspects of this thesis, which are the covariance function parameter estimation from Section 2.4 and the kernel search from Section 3.2. The discussion of Bayesian model selection is based on the narration of Rasmussen and Williams [44, p.108].

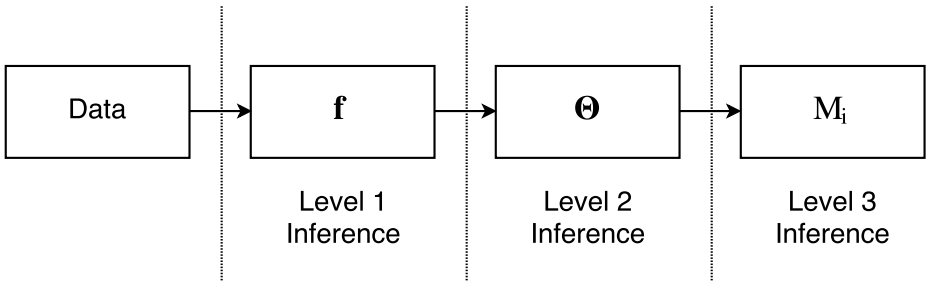


Figure B.1: In Gaussian Process, the parameter posterior is inferred in a hierarchical manner. At the first level, the random function $\mathbf{f}$ is inferred and then followed by the inference of the covariance function parameter $\boldsymbol{\Theta}$. Lastly, the best kernel combination M_i is selected from the third level inference.

In Gaussian Process regression, we work with three levels of parameters. These parameters affect each other hierarchically, where the parameter of the lower level is controlled by the parameter of the higher one. The lowest level of the parameter in the Gaussian Process regression is the function $\mathbf{f}$. The random function $\mathbf{f}$ is distributed by a Gaussian process which is characterised by the covariance function. The covariance function is controlled by the parameters $\boldsymbol{\Theta}$, for example $\boldsymbol{\Theta} = [\lambda, \sigma]^T$ for an RBF covariance function. Since $\boldsymbol{\Theta}$ controls the distribution of the $\mathbf{f}$, we place $\boldsymbol{\Theta}$ on the second level of the parameter hierarchy. $\boldsymbol{\Theta}$ is a parameter of the parameter $\mathbf{f}$, so the parameter $\boldsymbol{\Theta}$ is often named as the *hyperparameter*. On the highest level, we have the model M_i from a discrete set $\mathcal{M}$. An example of the model is found in Section 3.2, where each model corresponds to a different combination of kernels. A full Bayesian inference computes the posterior of these parameters, and the computation is done on three levels. Figure B.1 summarises the hierarchical concept. First, we calculate the posterior of $\mathbf{f}$, followed by the calculation of the posterior of $\boldsymbol{\Theta}$ and then we compute the posterior of M_i. As we will see next, computing the posterior of the higher level parameter requires the computation of the lower level marginal likelihood.

Level 1 Inference Posterior over $\mathbf{f}$ is given by

$$p(\mathbf{f}|\mathbf{y}, \mathbf{X}, \boldsymbol{\Theta}, M_i) = \frac{p(\mathbf{y}|\mathbf{X}, M_i, \mathbf{f})p(\mathbf{f}|\boldsymbol{\Theta}, M_i)}{p(\mathbf{y}|\mathbf{X}, \boldsymbol{\Theta}, M_i)} \tag{B.1}$$

$p(\mathbf{y}|\mathbf{X}, M_i, \mathbf{f})$ is the likelihood of the data given the function $\mathbf{f}$ and $p(\mathbf{f}|\boldsymbol{\Theta}, M_i)$ is the prior over function. The denominator is called the marginal likelihood

$$p(\mathbf{y}|\mathbf{X}, \boldsymbol{\Theta}, M_i) = \int p(\mathbf{y}|\mathbf{X}, M_i, \mathbf{f}) p(\mathbf{f}|\boldsymbol{\Theta}, M_i) d\mathbf{f} \tag{B.2}$$

which is the likelihood that is weighted by the prior over function averaged for all $\mathbf{f}$.

Level 2 Inference Level 2 inference computes the posterior over kernel parameter $\boldsymbol{\Theta}$. The posterior can be calculated as

$$p(\boldsymbol{\Theta}|\mathbf{y}, \mathbf{X}, M_i) = \frac{p(\mathbf{y}|\mathbf{X}, \boldsymbol{\Theta}, M_i) p(\boldsymbol{\Theta}|M_i)}{p(\mathbf{y}|\mathbf{X}, M_i)} \tag{B.3}$$

Notice that the marginal likelihood from the level 1 inference acts as the likelihood of the hyperparameter. $p(\boldsymbol{\Theta}|M_i)$ is the prior of the hyperparameter. The denominator of the hyperparameter posterior is given by

$$p(\mathbf{y}|\mathbf{X}, M_i) = \int p(\mathbf{y}|\mathbf{X}, \boldsymbol{\Theta}, M_i) p(\boldsymbol{\Theta}|M_i) d\boldsymbol{\Theta} \tag{B.4}$$

Level 3 Inference The posterior over model is given by

$$p(M_i|\mathbf{X}, \mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{X}, M_i) p(M_i)}{p(\mathbf{y}|\mathbf{X})} \tag{B.5}$$

and the denominator can be computed as

$$p(\mathbf{y}|\mathbf{X}) = \sum_i p(\mathbf{y}|\mathbf{X}, M_i) p(M_i) \tag{B.6}$$

Again, the marginal likelihood of the level 2 inference is the likelihood of the model posterior. This equation shows that every hierarchy of inference is connected and inferring the parameter depends on the inference on the lower level.

B.1. Level 2 Inference in Section 2.4

Level 2 inference plays an significant role in Gaussian process regression model. In Section 2.4, it is mentioned that training a GP regression model involves in finding the optimum parameter for the covariance function, which we can obtain by computing the posterior over hyperparameter from Equation B.3. Calculating the exact hyperparameter posterior is challenging because the denominator from Equation B.4 usually cannot be computed analytically [44]. As an approximation, the prior of the hyperparameter, $p(\boldsymbol{\Theta}|M_i)$, is considered equiprobable. Because of that, the posterior of the hyperparameter is now proportional to the marginal likelihood of the level 1 inference as the denominator is constant for all hyperparameter

$$p(\boldsymbol{\Theta}|\mathbf{y}, \mathbf{X}, M_i) \propto p(\mathbf{y}|\mathbf{X}, \boldsymbol{\Theta}, M_i) \tag{B.7}$$

We are interested in the value of the most optimum hyperparameter of the covariance function $\boldsymbol{\Theta}_*$. To get $\boldsymbol{\Theta}_*$, we calculate the most probable hyperparameter posterior, $p(\boldsymbol{\Theta}_*|\mathbf{y}, \mathbf{X}, M_i)$, or in other words, we calculate the posterior with the highest value. The hyperparameter that gives the most probable posterior is the most optimum hyperparameter. Since the posterior is proportional to the likelihood, we can maximise the likelihood instead w.r.t to hyperparameter $\boldsymbol{\Theta}$ to get this most optimum hyperparameter

$$\boldsymbol{\Theta}_* = \arg\max_{\boldsymbol{\Theta}} p(\mathbf{y}|\mathbf{X}, \boldsymbol{\Theta}, M_i) \tag{B.8}$$

This above equation explains why in Section 2.4 we maximise the marginal likelihood to train a GP regression model. The marginal likelihood is given by Equation B.2 and can be computed analytically.

The first term of the integral in Equation B.2 is the likelihood of the data given the function $\mathbf{f}$ while the second the is the prior over function. We already have defined the likelihood from Equation 2.14, and it is Gaussian distributed, $p(\mathbf{y}|\mathbf{X}, M_i, \mathbf{f}) \sim \mathcal{N}(\mathbf{f}, \sigma_N^2\mathbf{I})$. The prior, which is shown by the Equation 2.13 is again Gaussian distributed, $p(\mathbf{f}|\boldsymbol{\Theta}, M_i) \sim \mathcal{N}(0, \mathbf{K})$. Now, because both of the likelihood and prior are Gaussian, the integral in Equation B.2 can be computed in closed-form. It is common to work with the logarithm of the marginal likelihood since we are dealing with very small numbers in the likelihood. The log-marginal likelihood is given by

$$\log p(\mathbf{y}|\mathbf{X}) = -\frac{1}{2}\mathbf{y}^T[\mathbf{K}(\mathbf{X}, \mathbf{X}; \boldsymbol{\Theta}) + \sigma_n^2\mathbf{I}]^{-1}\mathbf{y} - \frac{1}{2}\log\det[\mathbf{K}(\mathbf{X}, \mathbf{X}; \boldsymbol{\Theta}) + \sigma_n^2\mathbf{I}] - \frac{n}{2}\log 2\pi \tag{B.9}$$

Here we make the covariance matrix $\mathbf{K}$ explicitly depends on the hyperparameter $\boldsymbol{\Theta}$.

Training a Gaussian process regression model is equivalent to maximising the negative log-marginal likelihood

$$\boldsymbol{\Theta}_* = \arg\max_{\boldsymbol{\Theta}} \quad -\frac{1}{2}\mathbf{y}^T[\mathbf{K}(\mathbf{X}, \mathbf{X}; \boldsymbol{\Theta}) + \sigma_n^2\mathbf{I}]^{-1}\mathbf{y} - \frac{1}{2}\log\det[\mathbf{K}(\mathbf{X}, \mathbf{X}; \boldsymbol{\Theta}) + \sigma_n^2\mathbf{I}] - \frac{n}{2}\log 2\pi \tag{B.10}$$

In Section 2.4 we have seen that maximising a marginal likelihood can select the hyperparameter while avoiding overfitting. The explanation of Rasmussen and Ghahramani give an intuitive understanding of why this is happening [43] (the notation is slightly changed to adapt with the notation of our document):

> The evidence (marginal likelihood) is the probability that if you randomly selected parameter values from your model class, you would generate data set $\mathbf{y}$. Models that are too simple will be very unlikely to generate that particular dataset, whereas models that are too complex can generate many possible data sets, so again, they are unlikely to generate that particular data set at random.

This exactly what happens we assign an RBF kernel with a very short λ. A short λ can model any data. Because of that, the likelihood that our particular dataset will be generated is very unlikely, since a short λ is very flexible that it can generate a various kind of data. Likewise, a long λ is very limited modelling capability, hence the probability that this model will generate $\mathbf{y}$ is very small as well.

B.2. Level 3 Inference in Section 3.2.3

An application of the level 3 inference is found in Section 3.2.3. There, we are faced with the problem of selecting the best model M_i from a discrete set $\mathcal{M}$. Each of the models is the possible kernel combination in the search tree. According to the Bayesian methodology, the best model from the set is the model which maximise the model posterior from Equation B.5. Normally, the prior probability of the model, $p(M_i)$, is equal. In that case, the model posterior is proportional to the likelihood

$$p(M_i|\mathbf{X}, \mathbf{y}) \propto p(\mathbf{y}|\mathbf{X}, M_i) \tag{B.11}$$

where the likelihood of the model posterior is equivalent to the marginal likelihood of the level 2 inference (Equation B.4). The integral in the marginal likelihood $p(\mathbf{y}|\mathbf{X}, M_i)$ is often difficult to solve. As it was already mentioned in Section 3.2.3, an approximation of this integral is the Bayesian information criterion (BIC) [47]

$$\text{BIC} = -2 \log p(\mathbf{y}|\mathbf{X}, \boldsymbol{\Theta}, M_i) + m \log n \tag{B.12}$$

The BIC equation from Equation 3.6 and Equation B.12 is equivalent, with a slight change of notation. The interested reader can follow the derivation of the BIC from Wasserman [54] or Neath and Kavenaugh [37]. BIC is asymptotically equivalent to the model posterior of Equation B.5 under several weak conditions [54]. The most important thing to note is the asymptotic assumption, which makes BIC more effective for medium to large data.

In Section 3.2.3, it was mentioned that using the marginal likelihood of the hyperparameter is not appropriate. By using this marginal likelihood, we approximate the marginal likelihood of the model, $p(\mathbf{y}|\mathbf{X}, M_i)$ by the marginal likelihood of the hyperparameter

$$p(\mathbf{y}|\mathbf{X}, M_i) \approx p(\mathbf{y}|\mathbf{X}, \boldsymbol{\Theta}, M_i) \tag{B.13}$$

In this setting, we completely ignore the integral of Equation B.4 and lose the regularisation effect that is provided by the integral. In Section 2.4 we have seen the effect of the integral that regularises the hyperparameter selection, and thus it avoids overfitting. By disregarding the integral, it is easy for the model selection to overfit. The complex model, which in our case the model with many combinations of base kernels, will get high likelihood $p(\mathbf{y}|\mathbf{X}, \boldsymbol{\Theta}, M_i)$ because it models the data well. Figure 3.6a shows an example of this circumstance. This situation is similar to the case where short λ will fits the noisy data in Section B.1. Favouring a complex model is not desirable, that is why comparing model by their marginal likelihood of the parameter is not suggested. The BIC tries to approximate the integral provided by Equation B.4. The regularisation is provided by the right-hand term of the BIC equation. It penalises model with too many kernel parameters. The effect of this can be clearly seen in Figure 3.6b.

Bibliography

[1] Ratnadip Adhikari and R. K. Agrawal. An Introductory Study on Time Series Modeling and Forecasting. *arXiv:1302.6613 [cs, stat]*, February 2013.

[2] Sylvain Arlot and Alain Celisse. A survey of cross-validation procedures for model selection. *Statistics Surveys*, 4:40–79, 2010. ISSN 1935-7516. doi: 10.1214/09-SS054.

[3] M.Z. Babai, M.M. Ali, J.E. Boylan, and A.A. Syntetos. Forecasting and inventory performance in a two-stage supply chain with ARIMA(0,1,1) demand: Theory and empirical analysis. *International Journal of Production Economics*, 143(2):463–471, 2013. ISSN 0925-5273. doi: 10.1016/j.ijpe.2011.09.004.

[4] 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, June 2012. ISSN 0957-4174. doi: 10.1016/j.eswa.2012.01.039.

[5] Christoph Bergmeir and José M. Benítez. On the use of cross-validation for time series predictor evaluation. *Information Sciences*, 191:192–213, May 2012. ISSN 0020-0255. doi: 10.1016/j.ins.2011.12.028.

[6] Gianluca Bontempi, Souhaib Ben Taieb, and Yann-Aël Le Borgne. Machine learning strategies for time series forecasting. In *Business Intelligence*, pages 62–77. Springer, 2013.

[7] George E. P. Box and Gwilym M. Jenkins. *Time Series Analysis: Forecasting and Control*. Holden-Day series in time series analysis and digital processing. Holden-Day, San Francisco, rev. ed edition, 1976. ISBN 978-0-8162-1104-3.

[8] Sofiane Brahim-Belhouari and Amine Bermak. Gaussian process for nonstationary time series prediction. *Computational Statistics & Data Analysis*, 47(4):705–712, November 2004. ISSN 0167-9473. doi: 10.1016/j.csda.2004.02.006.

[9] Peter J. Brockwell and Richard A. Davis. *Introduction to Time Series and Forecasting*. Springer texts in statistics. Springer, New York, 2nd ed edition, 2002. ISBN 978-0-387-95351-9.

[10] Nicolas Chapados and Yoshua Bengio. Augmented Functional Time Series Representation and Forecasting with Gaussian Processes. pages 265–272, 2007.

[11] Nicolas Chapados and Yoshua Bengio. Forecasting and Trading Commodity Contract Spreads with Gaussian Processes. In *13th International Conference on Computing in Economics and Finance*, 2007.

[12] Robert Cleveland, William Cleveland, Jean Mcrae, and Irma Terpenning. STL: A Seasonal-Trend Decomposition Procedure Based on Loess. *Journal of Official Statistics*, 6(1):3–73, 1990.

[13] Ilhami Colak, Seref Sagiroglu, and Mehmet Yesilbudak. Data mining and wind power prediction: A literature review. *Renewable Energy*, 46:241–247, October 2012. ISSN 0960-1481. doi: 10.1016/j.renene.2012.02.015.

[14] P. Cortez, M. Rio, M. Rocha, and P. Sousa. Multi-scale Internet traffic forecasting using neural networks and time series methods. *Expert Systems*, 29(2):143–155, 2012. ISSN 0266-4720. doi: 10.1111/j.1468-0394.2010.00568.x.

[15] Jan G. De Gooijer and Rob J. Hyndman. 25 years of time series forecasting. *International Journal of Forecasting*, 22(3):443–473, January 2006. ISSN 01692070. doi: 10.1016/j.ijforecast.2006.01.001.

[16] David K Duvenaud, James Robert Lloyd, Roger B Grosse, Joshua B Tenenbaum, and Zoubin Ghahramani. Structure Discovery in Nonparametric Regression through Compositional Kernel Search. In *ICML (3)*, pages 1166–1174, 2013.

[17] Philippe Esling and Carlos Agon. Time-series data mining. *ACM Computing Surveys*, 45(1):1–34, November 2012. ISSN 03600300. doi: 10.1145/2379776.2379788.

[18] Roger Frigola. *Bayesian Time Series Learning with Gaussian Processes*. PhD thesis, University of Cambridge, 2015.

[19] Agathe Girard, Carl Edward Rasmussen, J. Quinonero-Candela, R. Murray-Smith, J. Quinonero-Candela, O. Winther, J. Quinonero-Candela, A. Girard, J. Larsen, C. Rasmussen, and others. Multiple-step ahead prediction for non linear dynamic systems–a gaussian process treatment with propagation of the uncertainty. *Advances in neural information processing systems*, 15:529–536, 2002.

[20] D. Gounopoulos, D. Petmezas, and D. Santamaria. Forecasting Tourist Arrivals in Greece and the Impact of Macroeconomic Shocks from the Countries of Tourists' Origin. *Annals of Tourism Research*, 39(2):641–666, 2012. ISSN 0160-7383. doi: 10.1016/j.annals.2011.09.001.

[21] GPy. GPy: A Gaussian process framework in python. since 2012.

[22] Alex Graves. *Supervised Sequence Labelling with Recurrent Neural Networks*, volume 385 of *Studies in Computational Intelligence*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-24796-5 978-3-642-24797-2.

[23] T. Hachino and V. Kadirkamanathan. Time Series Forecasting Using Multiple Gaussian Process Prior Model. In *IEEE Symposium on Computational Intelligence and Data Mining, 2007. CIDM 2007*, pages 604–609, March 2007. doi: 10.1109/CIDM.2007.368931.

[24] J. Hartikainen and S. Särkkä. Kalman filtering and smoothing solutions to temporal Gaussian process regression models. In *2010 IEEE International Workshop on Machine Learning for Signal Processing*, pages 379–384, August 2010. doi: 10.1109/MLSP.2010.5589113.

[25] D. C. Hill, D. McMillan, K. R. W. Bell, and D. Infield. Application of Auto-Regressive Models to U.K. Wind Speed Data for Power System Impact Studies. *IEEE Transactions on Sustainable Energy*, 3(1):134–141, January 2012. ISSN 1949-3029. doi: 10.1109/TSTE.2011.2163324.

[26] Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara Sainath, and Brian Kingsbury. Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. *IEEE Signal Processing Magazine*, 29(6):82–97, November 2012. ISSN 1053-5888. doi: 10.1109/MSP.2012.2205597.

[27] S. L Ho, M Xie, and T. N Goh. A comparative study of neural network and Box-Jenkins ARIMA modeling in time series prediction. *Computers & Industrial Engineering*, 42(2–4):371–375, April 2002. ISSN 0360-8352. doi: 10.1016/S0360-8352(02)00036-0.

[28] Rob J. Hyndman and George Athanasopoulos. *Forecasting: Principles and Practice*. OTexts, S.l., print edition, april 2014 edition, 2014. ISBN 978-0-9875071-0-5.

[29] Rob J. Hyndman and Yeasmin Khandakar. Automatic Time Series Forecasting: The forecast Package for R. *Journal of Statistical Software*, 27(3), 2008. ISSN 1548-7660. doi: 10.18637/jss.v027.i03.

[30] Jaesung Jung and Robert P. Broadwater. Current status and future advances for wind speed and power forecasting. *Renewable and Sustainable Energy Reviews*, 31:762–777, March 2014. ISSN 1364-0321. doi: 10.1016/j.rser.2013.12.054.

[31] Robert E. Kass and Adrian E. Raftery. Bayes Factors. *Journal of the American Statistical Association*, 90(430):773–795, June 1995. ISSN 0162-1459, 1537-274X. doi: 10.1080/01621459.1995.10476572.

[32] L. Lättilä and O.-P. Hilmola. Forecasting long-term demand of largest Finnish sea ports. *International Journal of Applied Management Science*, 4(1):52–79, 2012. ISSN 1755-8913. doi: 10.1504/IJAMS.2012.044871.

[33] James Robert Lloyd, David Duvenaud, Roger Grosse, Joshua B. Tenenbaum, and Zoubin Ghahramani. Automatic Construction and Natural-Language Description of Nonparametric Regression Models. In *Association for the Advancement of Artificial Intelligence (AAAI)*, 2014.

[34] Tomáˇs Mikolov, Martin Karafiát, Lukáˇs Burget, Jan ˇCernockỳ, and Sanjeev Khudanpur. Recurrent Neural Network Based Language Model. In *Eleventh Annual Conference of the International Speech Communication Association*, 2010.

[35] H. Mori and E. Kurata. Application of Gaussian Process to wind speed forecasting for wind power generation. In *2008 IEEE International Conference on Sustainable Energy Technologies*, pages 956–959, November 2008. doi: 10.1109/ICSET.2008.4747145.

[36] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. Adaptive computation and machine learning series. MIT Press, Cambridge, MA, 2012. ISBN 978-0-262-01802-9.

[37] Andrew A. Neath and Joseph E. Cavanaugh. The Bayesian information criterion: Background, derivation, and applications. *Wiley Interdisciplinary Reviews: Computational Statistics*, 4(2):199–203, March 2012. ISSN 19395108. doi: 10.1002/wics.199.

[38] M. Ordon, D. Urbach, M. Mamdani, R. Saskin, Honey D'A, and K.T. Pace. The surgical management of kidney stone disease: A population based time series analysis. *Journal of Urology*, 192 (5):1450–1456, 2014. ISSN 0022-5347. doi: 10.1016/j.juro.2014.05.095.

[39] M. A. Osborne, S. J. Roberts, A. Rogers, S. D. Ramchurn, and N. R. Jennings. Towards Real-Time Information Processing of Sensor Network Data Using Computationally Efficient Multi-output Gaussian Processes. In *International Conference on Information Processing in Sensor Networks, 2008. IPSN '08*, pages 109–120, April 2008. doi: 10.1109/IPSN.2008.25.

[40] Michael A. Osborne, Stephen J. Roberts, Alex Rogers, and Nicholas R. Jennings. Real-time Information Processing of Environmental Sensor Network Data Using Bayesian Gaussian Processes. *ACM Trans. Sen. Netw.*, 9(1):1:1–1:32, November 2012. ISSN 1550-4859. doi: 10.1145/2379799.2379800.

[41] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12: 2825–2830, October 2011.

[42] Joaquin Quinonero-Candela and Carl Edward Rasmussen. A unifying view of sparse approximate Gaussian process regression. *The Journal of Machine Learning Research*, 6:1939–1959, 2005.

[43] Carl Edward Rasmussen and Zoubin Ghahramani. Occam's razor. *Advances in neural information processing systems*, pages 294–300, 2001.

[44] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. University Press Group Limited, January 2006. ISBN 978-0-262-18253-9.

[45] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 1135–1144, San Francisco, California, USA, 2016. ACM. ISBN 978-1-4503-4232-2. doi: 10.1145/2939672.2939778.

[46] R. Ruiter, L.E. Visser, Herk-Sukel Van, P.H. Geelhoed-Duijvestijn, Bie De, S.M.J.M. Straus, P.G.M. Mol, S.A. Romio, R.M.C. Herings, and B.H.C. Stricker. Prescribing of rosiglitazone and pioglitazone following safety signals: Analysis of trends in dispensing patterns in the Netherlands from 1998 to 2008. *Drug Safety*, 35(6):471–480, 2012. ISSN 0114-5916. doi: 10.2165/11596950-000000000-00000.

[47] Gideon Schwarz. Estimating the Dimension of a Model. *The Annals of Statistics*, 6(2):461–464, March 1978. ISSN 0090-5364, 2168-8966. doi: 10.1214/aos/1176344136.

[48] Robert H. Shumway and David S. Stoffer. *Time Series Analysis and Its Applications: With R Examples*. Springer texts in statistics. Springer, New York, NY, 3. ed edition, 2011. ISBN 978-1-4419-7865-3 978-1-4419-7864-6.

[49] Nate Silver. *The Signal and the Noise : Why so Many Predictions Fail–but Some Don't*. Penguin Press, New York, 2012. ISBN 978-1-59420-411-1 1-59420-411-X 978-0-14-312400-9 0-14-312400-5 978-0-14-312508-2 0-14-312508-7.

[50] S. S. Soman, H. Zareipour, O. Malik, and P. Mandal. A review of wind power and wind speed forecasting methods with different time horizons. In *North American Power Symposium (NAPS), 2010*, pages 1–8, September 2010. doi: 10.1109/NAPS.2010.5619586.

[51] Antti Sorjamaa, Jin Hao, Nima Reyhani, Yongnan Ji, and Amaury Lendasse. Methodology for long-term prediction of time series. 70(16–18):2861–2869, October 2007. ISSN 0925-2312. doi: 10.1016/j.neucom.2006.06.015.

[52] W.H.K. Tsui, Balli Ozer, A. Gilbey, and H. Gow. Forecasting of Hong Kong airport's passenger throughput. *Tourism Management*, 42:62–76, 2014. ISSN 0261-5177. doi: 10.1016/j.tourman.2013.10.008.

[53] W.-C. Wang, K.-W. Chau, D.-M. Xu, and X.-Y. Chen. Improving Forecasting Accuracy of Annual Runoff Time Series Using ARIMA Based on EEMD Decomposition. *Water Resources Management*, 29(8):2655–2675, 2015. ISSN 0920-4741. doi: 10.1007/s11269-015-0962-6.

[54] Larry Wasserman. Bayesian Model Selection and Model Averaging. *Journal of Mathematical Psychology*, 44(1):92–107, March 2000. ISSN 0022-2496. doi: 10.1006/jmps.1999.1278.

[55] Steven Wheelwright, Spyros Makridakis, and Rob J Hyndman. *Forecasting: Methods and Applications*. John Wiley & Sons, 1998.

[56] W. Yan, H. Qiu, and Y. Xue. Gaussian process for long-term time-series forecasting. In *International Joint Conference on Neural Networks, 2009. IJCNN 2009*, pages 3420–3427, June 2009. doi: 10.1109/IJCNN.2009.5178729.

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

[58] Ladislav Zjavka. Wind speed forecast correction models using polynomial neural networks. *Renewable Energy*, 83:998–1006, November 2015. ISSN 09601481. doi: 10.1016/j.renene.2015.04.054.