Wind Turbine Monitoring with Non-Stationary and Multi-Task Gaussian Processes

Steve Hong

Department of Statistical Science

University College London

Supervisors

Prof. Petros Dellaportas & Miss Domna Ladopoulou

A written report for the STAT0035 Project

19th April 2024

Word Count: 14,140

Acknowledgements

Firstly, I would like to thank my first supervisor, Professor Petros Dellaportas, who not only proposed the central idea of this project but also provided invaluable advice and guidance throughout the past seven months. His inspiration and lessons have been pivotal to my research and learning journey and will continue to stay with me in my future career.

My thanks also go to my second supervisor, Miss Domna Ladopoulou, for her tremendous support and amazing advice. Whenever I encountered obstacles, she was always there to help, offering valuable insights that greatly contributed to my progress.

I also want to thank Jeff, whom I could always rely on throughout this very stressful year. And also thanks my Má and Ba for their support and the phone calls that cheer me up. Having heard about this project, Ba is no longer angry that I am not becoming a medical doctor.

Lastly, a big, heartfelt thank you goes to Jeni, Anthony, James, and Melodie without whom my final year at UCL would not have been the same.

Abstract

Gaussian Processes are becoming a popular framework in monitoring wind turbines' condition, especially for early fault detection, thanks to their expressiveness, robustness, and tractability. The performance of this type of model is significantly affected by the selection of kernel functions. Most of the existing research in wind turbine monitoring with Gaussian Processes employs standard stationary kernels, which are not optimal for wind power forecasting where patterns are inherently complex. This project aims to leverage scalable, non-stationary and multi-task kernel designs to apply Gaussian Process regression on a wind farm SCADA dataset and conduct a comparative analysis of their performance against baseline models.

Contents

Chapter 1

Introduction

1.1 Motivation

Wind power is crucial in the global shift towards renewable energy. By 2023, the world had installed about 906 gigawatts of wind energy capacity, growing by 9% from the previous year [World Energy, 2023]. However, its operation and maintenance (O&M) costs are very high, accounting for approximately 25% to 30% of the total costs over a wind farm's lifetime. For a 500 MW offshore wind farm, the O&M costs are between £25 million and £40 million. Without optimally scheduled maintenance, wind turbines can experience abrupt failures and cause financial losses. Condition monitoring is therefore essential reducing O&M costs and helping to speed up the adoption of wind power worldwide.

Machine learning plays an important role in optimising the monitoring process of wind farms [Zhang et al., 2019]. This approach utilises data from Supervisory Control and Data Acquisition (SCADA) systems, which are already in place for performance monitoring in most large-scale wind turbines. This is gaining attention due to its cost-effectiveness and smart use of existing infrastructure, proving to be a promising technique for maintaining wind turbines and preventing faults. The SCADA dataset for this project comes from Kelmarsh Wind Farm, situated near

Haselbech in Northamptonshire, UK. This wind farm features six turbines, each capable of generating 2.05MW of power. These turbines are of the MM92 model. An image of this wind farm can be seen in Figure 1.1. The dataset will be analysed in detail in Chapter 6.

Figure 1.1: Kelmarsh Wind Farm [Helm, 2023]

A key strategy within machine learning-driven condition monitoring is regression-based anomaly detection [Stetco et al., 2019]. This involves modelling the relationship between the operational measurements from the turbine components and the corresponding power outputs. Machine learning models are used to establish this relationship using historical data, and when new data is observed, they are compared against the model's inferences using residual analysis methods. For example, if these new observations fall below the lower confidence interval over several consecutive time intervals, an alarm is triggered and allows for early maintenance of the turbine. This project focuses only on the inference step by exploring methods that can outperform current benchmarks in the literature.

1.2 Related Work

A range of machine learning techniques have been experimented with for wind turbine monitoring [Khan and Byun, 2024]. Examples include support vector machine [Yang et al., 2019], which is a popular kernel method for non-linear regression and classification. Active research has also been conducted on the application of deep neural networks to enhance the prediction of wind power output. In Zhu et al. [2023], a convolutional neural network and a long short-term memory network were implemented to monitor the conditions of wind turbine gearbox bearings. In Jiang et al. [2018], a denoising autoencoder is employed to learn robust representations from noisy sensor data, using a sliding window technique to capture temporal correlations. Deep neural networks are extremely powerful because of their flexibility in approximating relationships, yet they are difficult to work with and require very careful architectural design in order to optimise performance and avoid over-fitting.

Gaussian Process (GP) regression is also recognised as an effective method for non-linear regression, especially useful in wind power prediction [Pandit and Infield, 2018]. Unlike standard point prediction methods, GP is a fully probabilistic model, offering predictive distributions that provide a range of possible outcomes. This aspect is particularly valuable for wind power forecasting, where reliability can be a serious issue. GP stands out when compared to modern deep neural networks techniques due to its flexibility, robustness to over-fitting and well-calibrated uncertainty estimates [Tazi et al., 2023]. Given these advantages, GP is especially suitable for accurate and reliable forecasting tool in this domain.

There is a gap within the GP regression literature for wind turbine monitoring on the choice of the kernel function. In very recent works, such as in [Yadav et al., 2024] and [Pandit and Infield, 2018], standard kernel packages such as the Square-Exponential and Matérn family are used. Despite their speed and simplicity, there is space for improvements on flexibility given the highly complex patterns of wind power data. Some research also explores hand-crafting new kernel by taking products

of multiple standard kernels using domain knowledge, such as [Lee and Sutherland, 2024], to improve performance. However, as climate change and global warming changes underlying patterns of wind power [Ohba, 2019], there will be a need to redesign and recalibrate these hand-crafted kernels every few years. This project proposes implementing very unified, flexible and non-stationary classes of kernel that perform well without the need for hand-crafting them.

1.3 Contributions

This project makes the following contributions:

- Created a novel extension for GPyTorch to include the Non-Stationary Spectral kernel, leveraging the existing framework of the Gibb's kernel as produced by Lalchand et al. [2023]. This addition introduces a previously unavailable kernel choice in GPyTorch, enhancing its utility for modeling non-stationary data across diverse future applications. More details on this are in the Appendix.

- Extensively evaluated key tools in the GP literature, focusing on flexible kernel designs, the application of Multi-Task GP models, and Sparse GP Approximations for efficient and accurate wind power inference. Some of these results are novel to the wind turbine monitoring literature.

- The detail theory behind these methods are also explained explicitly and intuitively between Chapter 2 and 6 so that they are highly comprehensible to second and final year students in the BSc Statistics programme.

Chapter 2

Background

Summary

In this chapter, we will discuss the essential background materials extending those covered in the BSc Statistics programme in order to understand the literature on Gaussian Processes. We will begin by looking at the traditional machine learning paradigm, particularly how models are trained using stochastic optimisation and their ability to generalise. Following this, we will discuss the probabilistic framework to learning and Bayesian Linear Regression as an example. We will finally look at how this idea generalises to Gaussian Processes model and investigate the Gaussian Process framework in detail. Most of the key points in this chapter are drawn from Goodfellow et al [2016] and Rasmussen and Williams [2006], with other sources mentioned as needed.

2.1 Machine Learning

Machine learning can be viewed as an area of applied statistics, but mainly differing in its goals. While traditional statistical approaches aim to understand and estimate unknown parameters and test hypotheses, machine learning focuses on building al-

gorithms to leverage observed data and predict what might happen in new, unseen data. This is revolutionising because quite commonly, predictions are very expensive to be done with rule-based, mathematical, or deterministic approaches. Let's say in weather forecasting, instead of solving physics equations with supercomputers, recent advancements in machine learning research by Google DeepMind [2023] allowed for comparably fast and accurate forecasts to be done at a personal-computer level.

Types of Machine Learning Models

Depending on the data we work with, machine learning models can be generally categorised into two main classes:

- **Supervised Learning**: A model learns from a dataset where the output is already known and labelled. Possible tasks include regression or classification depending on if the outputs are continuous or discrete. Example: A weather forecasting model trained on historical weather data with input temperature and output rainfall.
- **Unsupervised Learning**: A model learns from data without predefined labels, allowing the model to identify patterns and relationships on its own. Example: A customer segmentation tool that groups customers into clusters based on purchasing behavior without any prior labeling.

We only focus on supervised learning in this project, specifically regression, because we are working with labelled continuous data - operational variables of wind turbines labelled with their power output.

2.1.1 Supervised Learning

The objective of supervised learning is to select a model that fits well on seen data, and can generalise at similar performance to unseen data. We can explain this objective as two problems: the **training problem** and the **generalisation problem**. They can be depicted in Figure 2.1.

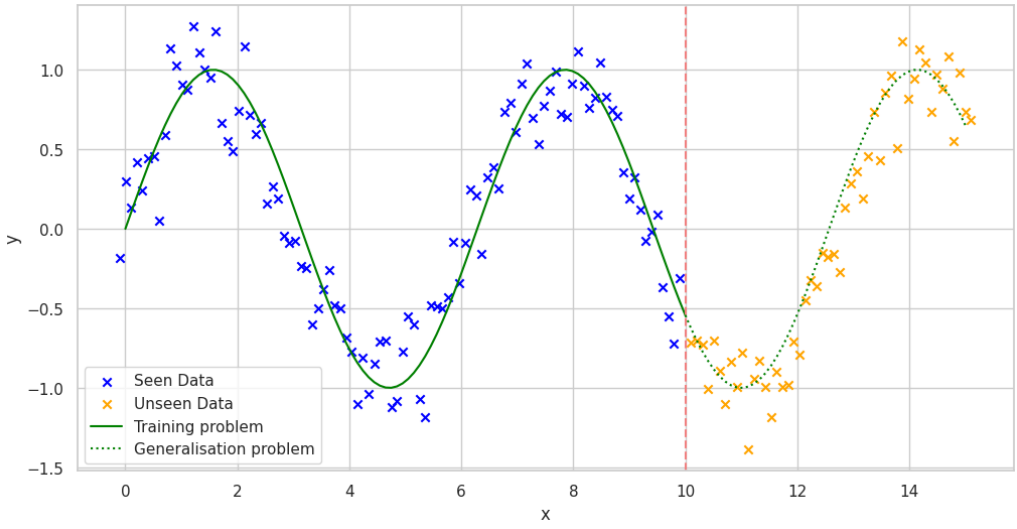


Figure 2.1: Training and generalisation in supervised learning

Let's say we have a dataset $\mathcal{D}$ which consists of N input-output pairs (x_i, y_i) where $i = 1, 2, \ldots, N$. If x_i has m features (or indepdendent variables), then it is a vector in $\mathbb{R}^m$. When we consider all N data points, these feature vectors can be stacked to form a design matrix X, represented in $\mathbb{R}^{N \times m}$ and a vector $\mathbf{y}$ in $\mathbb{R}^N$.

The training problem is concerned with learning a mapping $g : X \to \mathbf{y}$. The space of possible candidates for g, called the **hypothesis space**, is essentially very large - we are allowed to choose any models to learn this mapping. The traditional machine learning approach is to firstly restrict the size of this space significantly into a specific class of models f, and parameterise them to be $f(\boldsymbol{\theta})$. Sometimes, the restriction to f can be easy to recognise from data, for example in Figure 2.1 it is obvious that f is a sine function. But in many applications g is extremely irregular, and we need to choose more flexible f. The error of learning g is called the **loss function**, denoted as $L(\boldsymbol{\theta})$, which is the similarity between the model's predictions $f(X, \boldsymbol{\theta})$ and the actual outputs $\mathbf{y}$. The goal is to iteratively adjust $\boldsymbol{\theta}$ to minimise $L(\boldsymbol{\theta})$. Commonly employed loss functions are the squared loss for regression tasks and the likelihood loss for probabilistic models. The process of adjusting $\boldsymbol{\theta}$ is not

arbitrary but is guided by optimisation algorithms. These optimisation techniques will be explored in detail in Section 2.1.2.

Next, we can use the trained $\boldsymbol{\theta}$ to generalise to unseen data with $f(\boldsymbol{\theta})$, independently of the training data. The generalisation problem depends strongly on the complexity for f that we restrict from the hypothesis space. The complexity of a model is controlled by factors such as the model architecture (etc. linear/non-linear/periodic) or the number of parameters $\boldsymbol{\theta}$. The model will generalise the best when their complexity is appropriate for the true complexity of the underlying pattern in the data, i.e. the mapping from X to $\mathbf{y}$. Failure to choose the right model complexity will lead to problems such as over-fitting and under-fitting, which will be discussed in Section 2.1.3.

It is also important to note that the training and generalisation problem are often not mutually exclusive: poor fit of the model in training can lead to poor prediction performance and vice versa, an overly complex model can lead to an unnecessarily difficult optimisation problem.

A common practice to evaluate the training and prediction performance of models is to split the dataset into a training set and a test set, and benchmark them using metrics such as root-mean-square error (RMSE).

2.1.2 The Training Problem

In order to minimise the loss function $L(\boldsymbol{\theta})$, or equivalently maximise $-L(\boldsymbol{\theta})$, it's not always possible to naively find the gradient of the loss and equate it to zero, $\nabla L(\boldsymbol{\theta}) = 0$, because the equation might not have closed-form solutions. Numerical optimisation methods covered in the BSc Statistics programme include the Newton-Raphson's method. However, this method is computational expensive as it involves an inversion of a Hessian matrix, which is not suitable for large datasets. We therefore introduce Gradient Descent - a class of algorithms that are more efficient when optimising high-dimensional loss functions.

Gradient Descent and Stochastic Gradient Descent

The idea of gradient descent (GD) is to start at a point on $L(\boldsymbol{\theta})$, then keep taking small steps towards the steepest negative gradient (thus gradient "descent") until it reaches convergence. Convergence in theory means the steps becomes close to 0, but in practice, we preset the number of iterations of training to avoid infinite training loops. The detailed algorithm of GD is presented in Algorithm 1. It is worth noting that GD only requires computing the first-order derivative, simplifying computation compared to Newton-Raphson.

Algorithm 1: Gradient Descent for loss function $L(\boldsymbol{\theta})$

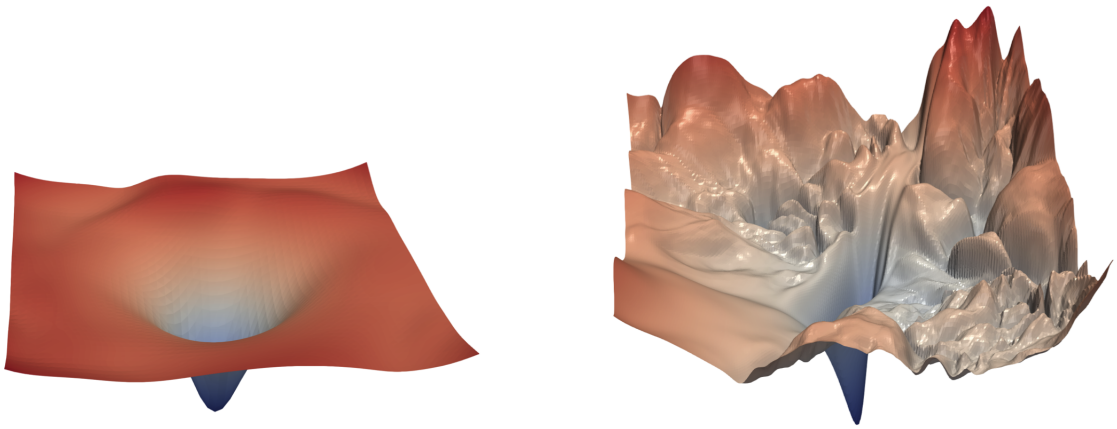
Require: α (Step size)
Require: $\boldsymbol{\theta}_0$ (Initial starting parameter)
Require: $T \leftarrow k$ (Number of iterations)
for *t in range of* T **do**
  $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \alpha \nabla L(\boldsymbol{\theta}_{t-1})$;
end
return θ_T

In GD, the process of computing $\nabla L(\boldsymbol{\theta})$ involves finding the gradient at every single data point and then averaging these to obtain the overall gradient, which is slow if repeated for many iterations. This calculation can be approximated and accelerated by stochastic gradient descent (SGD). Instead of using the entire dataset, SGD use an unbiased estimator, $\nabla L(\boldsymbol{\theta})_{mini}$, of the gradient by using a small, randomly selected subset (or mini-batch) at each iteration. We therefore need to choose the loss function for the mini-batch so that $E[\nabla L(\boldsymbol{\theta})_{mini}] = \nabla L(\boldsymbol{\theta})$.

Adaptive Methods: Momentum + RMSProp = ADAM

There are two shortcomings when optimising the loss function with mini-batch SGD. Firstly, the stochastic selection of mini-batches often leads to high variance in gradient estimations across updates, which causes the directions and sizes of updates to

fluctuate significantly and slows down convergence. Secondly, when we have many parameters or a complex model, there will be numerous saddle points and plateau regions (i.e. being non-convex) in the loss function. This frequently results in the optimiser becoming trapped, hindering its progress towards reaching a global minimum. This is illustrated through a comparison between an easy-to-train and a hard-to-train loss in Figure 2.2, produced by Li et al. [2018].



(a) Convex Loss (b) Highly Non-Convex Loss

Figure 2.2: From Li et al. [2018] (a) Convex Loss: This is a smooth and relatively convex landscape. There are not many local minima and it is easy for optimisation algorithms like SGD to converge. Training a model on a loss landscape like this is generally faster and has a higher chance of converging to the best solution. (b) Highly Non-Convex Loss: This is a very rugged and chaotic landscape with many sharp peaks and valleys. This indicates the presence of numerous local minima and saddle points, SGD can be very slow and may not converge to the best solution.

ADAM [Kingma and Ba, 2015] is one of the most popular algorithm used in training machine learning models. It extends mini-batch SGD to tackle the two issues above by integrating momentum strategy and RMSProp [Ruder, 2016]. The momentum strategy does not rely solely on a single mini-batch gradient for parameter

updates. Instead, it computes a weighted average of past gradients, guiding the descent towards a relevant direction. Concurrently, RMSProp's approach is also adopted where each update step is divided by a similar weighted average of past gradients. This kind of normalisation stabilises the step size, therefore decreasing the step for large gradients to avoid exploding, and increasing the step for small gradients to avoid vanishing.

2.1.3 The Generalisation Problem

Optimising the model to do well on training data does not guarantee that it will do well on test data. Generalisation performance is also subject to choosing the right **model capacity** that is suitable for the dataset. The capacity of a machine learning model refers to its ability to capture and represent the complexity of the underlying data it is trained on, and it's mostly determined by model architecture and the number of parameters. The generalisation ability of a model choice as its capacity increases can be explained in Figure 2.3, produced by Goodfellow et al [2016].

It is therefore very important to understand the complexity of the dataset before selecting models. This is usually done through visualisation techniques, correlation analysis, and engineering input features where necessary. Once having some ideas about how complex our model should be, we can select the architecture and parameters. For architecture, a common strategy is to start with a simple model, let's say linear regression, with a few parameter. We can fit it on the training data, if the training error is larger than the level we need, then we can decide that the model under-fits and explore a group of more complicated models, namely tree-based models and deep neural networks. We can reiterate this process and increase the level of complexity until we get a satisfactory performance. For choosing number of parameters, we can either deploy a similar approach - going from small to large, or to have a model with many parameters and set a preference towards a certain group of parameters. The latter approach is known as **regularisation**, which is done by adding a penalty term to the loss function of the model. Lasso and Ridge regularisation are

two typical examples for this approach:

$$\text{Ridge-regularised loss} = L(\boldsymbol{\theta}) + \lambda \sum_{i=1}^{m} \theta_i^2$$

$$\text{Lasso-regularised loss} = L(\boldsymbol{\theta}) + \lambda \sum_{i=1}^{m} |\theta_i|$$

Adding the penalty term, where the strength of the penalty is controlled by $\lambda \in [0, \infty)$, introduces a trade-off between the model's ability to minimise the original loss function and the complexity of the model. It therefore encourages the optimiser to select a simpler model. For example, Ridge regularisation influences the model selection by dampening the value of some large parameters, while Lasso promotes sparsity and zero out less important parameters.

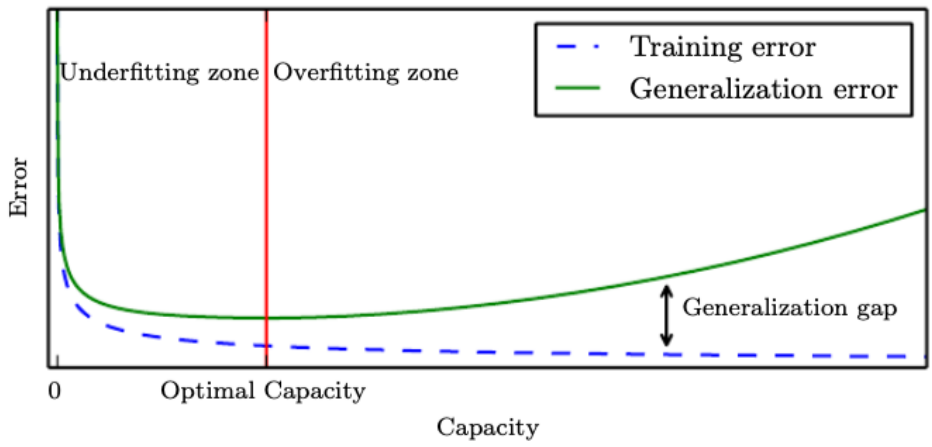


Figure 2.3: Goodfellow et al [2016] describes LHS: If the model's capacity is too low, it's regarded as under-fitting - the training error and the generalisation are high. RHS: If the model's capacity is too large for the data, it's regarded as over-fitting - models can have very desirable training metrics, but when we test it the the error increases significantly. Finding the optimal capacity point is a very challenging optimisation problem, but there are general guidelines how to select one with good capacity.

In summary, the traditional approach to learning involves choosing a parameterised model with suitable capacity then minimise the difference between the data generated from that model and the true data. Learning complicated data patterns therefore requires very careful selection of the model capacity with techniques such as regularisation, otherwise the model will over-fit and not generalise well to testing data. This model also only gives a best guess of the output at each input location, which is not ideal if we are interested in the uncertainty for each prediction made. In the next section, we will explore an alternative approach to learning that predicts a distribution of possible outcomes by leveraging the rules of probability and Bayes rule.

2.2 Probabilistic Approach to Learning

In probabilistic learning, instead of restricting the possible functions to a specific class, we propose a belief about the distribution over all possible functions before observing any data, known as the **prior distribution**. This prior distribution over functions does not have to be specific, we only need to set a preference towards some general properties, e.g. "smooth" or "sharp" functions. After that, we can then use training data to update and reduce our uncertainty about that distribution using Bayes' rule, which is called the **posterior distribution**. The training and generalisation problems essentially becomes similar to a **Bayesian inference** task, which has been well-covered in the BSc Statistics programme.

A question here is: how do we propose a distribution over functions, isn't probability distribution only for random variables? This section shows two ways to achieve this using Bayesian linear regression as an example. One way is to firstly do inference with the distribution over the parameters (weights) then induce the distribution over functions, referred to as **weight-space view**. Another way is to do inference directly with the distribution over functions, whereby we gain computational advantages. This is referred to as **function-space view**.

2.2.1 Weight-space view

In the weight-space view, we do inference on the parameters first then do inference on the model after. We recall the set-up of a linear regression problem:

$$f(X) = X\boldsymbol{\theta}, \quad \mathbf{y} = f(X) + \boldsymbol{\epsilon}, \tag{2.1}$$

Here, X represent the input design matrix, $\mathbf{y}$ is the observed output values. These observations are generated by a function f, which outlines how inputs relate to outputs, and are influenced by a noise term that represents the uncertainty in the data. The noise $\boldsymbol{\epsilon}$ vector is assumed to be iid standard Gaussian distributed i.e., $\epsilon_i \sim \mathcal{N}(0, \sigma_n^2)$. We can follow the three-step procedure to do inference:

Step 1: Choose a Prior Distribution

For conjugacy, we can choose the Gaussian prior for the parameters θ, which can be specified as:

$$\boldsymbol{\theta} \sim \mathcal{N}(0, \Sigma_p). \tag{2.2}$$

where we have a zero prior mean vector, and Σ_p is the covariance matrix of the prior. The tractability properties will be discussed further in Section 2.3.

Step 2: Derive the Posterior Distribution

The **likelihood** is the probability of the data as a function of the parameters. From equation 2.1, we have the likelihood:

$$p(\mathbf{y}|X, \boldsymbol{\theta}) = \mathcal{N}(X\boldsymbol{\theta}, \sigma_n^2\mathbf{I}). \tag{2.3}$$

The posterior is proportional with the product of the likelihood and the prior:

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

We can substitute the likelihood and prior with their Gaussian densities:

$$p(\boldsymbol{\theta}|X, \mathbf{y}) \propto \exp\left(-\frac{1}{2}\sigma_n^{-2}(\mathbf{y} - X^T\boldsymbol{\theta})^T(\mathbf{y} - X^T\boldsymbol{\theta})\right) \exp\left(-\frac{1}{2}\boldsymbol{\theta}^T\Sigma_p^{-1}\boldsymbol{\theta}\right). \tag{2.5}$$

By completing the square and algebraic simplification, we obtain:

$$p(\boldsymbol{\theta}|X,\mathbf{y}) \propto \exp\left(-\frac{1}{2}(\boldsymbol{\theta}-\bar{\boldsymbol{\theta}})^T\left(\sigma_n^{-2}XX^T+\Sigma_p^{-1}\right)(\boldsymbol{\theta}-\bar{\boldsymbol{\theta}})\right), \tag{2.6}$$

where $\bar{\boldsymbol{\theta}} = \sigma_n^{-2}(\sigma_n^{-2}XX^T+\Sigma_p^{-1})^{-1}Xy$. Let $A = \sigma_n^{-2}XX^T+\Sigma_p^{-1}$, we can see that the posterior has a form of a Gaussian with mean $\bar{\boldsymbol{\theta}}$ and the covariance matrix A^{-1}:

$$p(\boldsymbol{\theta}|X,\mathbf{y}) = \mathcal{N}(\sigma_n^{-2}A^{-1}X\mathbf{y}, A^{-1}). \tag{2.7}$$

Step 3: Find Predictive Distribution

As our main goal is to make predictions on unseen data, we need to find the predictive distribution of the function values. In Bayesian inference, we do this by considering the likelihood of unseen data given the parameters, weighted by the posterior probability. Let $\mathbf{x}_*$ be the unseen input vector, $\mathbf{f}_* = f(\mathbf{x}_*)$ is the predicted function values:

$$p(\mathbf{f}_*|\mathbf{x}_*,X,\mathbf{y}) = \int p(\mathbf{f}_*|\mathbf{x}_*,\boldsymbol{\theta})p(\boldsymbol{\theta}|X,\mathbf{y})d\boldsymbol{\theta} \tag{2.8}$$

$$= \int \mathcal{N}(\mathbf{x}_*^T\boldsymbol{\theta},\sigma_n^2)\mathcal{N}\left(\sigma_n^{-2}A^{-1}\mathbf{X}\mathbf{y}, A^{-1}\right)d\boldsymbol{\theta} \tag{2.9}$$

$$= \mathcal{N}\left(\sigma_n^{-2}\mathbf{x}_*^TA^{-1}\mathbf{X}\mathbf{y}, \mathbf{x}_*^TA^{-1}\mathbf{x}_*\right) \tag{2.10}$$

The step between equation 2.9 and 2.10 requires standard results of Gaussian in Section 2.19. It is also worth mentioning the idea of **basis function**. In Bayesian Linear Regression, we simply consider a linear combination of parameters and input values. We can generalise this to non-linear regression by applying $\boldsymbol{\phi}(X)$. Typical examples of basis functions are the polynomial function and the radial basis function. If we apply basis functions to apply our input, equation 2.10 will become:

$$p(f_*|\mathbf{x}_*,\mathbf{X},\mathbf{y}) = \mathcal{N}\left(\sigma_n^{-2}\boldsymbol{\phi}(\mathbf{x}_*)^T\mathbf{A}^{-1}\boldsymbol{\Phi}^T\mathbf{y}, \boldsymbol{\phi}(\mathbf{x}_*)^T\mathbf{A}^{-1}\boldsymbol{\phi}(\mathbf{x}_*)\right), \tag{2.11}$$

where $\boldsymbol{\Phi} = \boldsymbol{\phi}(X)$ and $\mathbf{A} = \sigma_n^{-2}\boldsymbol{\Phi}^T\boldsymbol{\Phi}+\Sigma_p^{-1}$.

We see that the entire process does not require any optimisation, but we integrate. Intuitively, this helps to avoid over-fitting by considering all possibilities of parameters rather than just selecting a best guess. Indeed, if we view this set-up from a traditional approach, having a prior is equivalent to adding a quadratic regularisation term $\boldsymbol{\theta}^T \Sigma_p^{-1} \boldsymbol{\theta}$ to the log likelihood loss, similar to Ridge regularisation in Section 2.1.3. While avoiding over-fitting, we also obtain an entire distribution over the predicted values, which important for some applications that need insights into the reliability of each prediction.

However, remember that in machine learning, our main goal is to make predictions, not understanding the meaning or contribution of each parameters. We notice in this process that, in order to infer $\mathbf{f}$, a series of manipulations of $\boldsymbol{\theta}$ were made, and we eventually just marginalised $\boldsymbol{\theta}$ out. This can be avoided by directly considering the distribution over $\mathbf{f}$.

2.2.2 Function-space view

In the function-space view, we do Bayesian inference directly with the distribution of functions. That means, we start with a prior distribution over functions $p(\mathbf{f})$, then we use training input, training output, and test input to calculate the posterior and predictive distribution $p(\mathbf{f}_*|\mathbf{x}_*, X, \mathbf{y})$ in one go with Bayes' rule, essentially skip all the computations that involve $\boldsymbol{\theta}$. This function-space view not only simplifies the inference process, but also gives better understanding of the distributions over functions. As any other probability distributions, the distribution over functions is characterised by a mean and a variance. But this time it is not a mean vector and covariance matrix, but a **mean function** and **kernel function**. A detailed proof of how equation 2.9 can be transformed and rearranged into a kernel function form can be found in Rasmussen and Williams [2006, p. 12]. For the rest of this project, we will focus on this view for doing inference. A prominent way to set a prior distribution over functions and compute the posterior with the mean and kernel functions is by using Gaussian Processes, which we will investigate next.

2.3 Gaussian Processes

In this section, we introduce Gaussian Processes (GP) as extensions of existing models and show how they can define distributions over functions. We continue to discuss the properties that allow GP to: (1) Model complex, non-linear relationships. (2) Give well-calibrated uncertainty estimates. (3) Do as many of the the computations as possible analytically. Finally, we will derive the steps how the prior distribution can be updated and optimised to make inferences on unseen data.

2.3.1 Definition

The formal definition of GP as a stochastic process is as follow:

Definition 1: *A Gaussian Process is a collection of random variables, any finite number of which have (consistent) joint Gaussian distributions.*

This definition says that if we set a GP prior over function value f_i where $i \in \mathbb{N}$, we assume that each of $f_1, f_2, ...$, and any subset of them, for example (f_1, f_2) or (f_{10}, f_{20}, f_{30}), all follow a univariate and multivariate Gaussian distribution respectively. A GP is fully specified with a mean function $m(x)$ and a kernel function $k(x, x^{'})$, were x and x' are indices of data points. We can write:

$$f \sim \mathcal{GP}(m(x), k(x, x')) \tag{2.12}$$

Intuitively, the mean function controls the base line where the functions are expected to "wiggle" about, while the kernel function determines how each pair of function values "wiggle" against each other. This can be illustrated in Figure 2.4. The mean function is by default set to be 0, while there are numerous design options for the kernel function. We discuss the kernel design problem formally in chapter 3.

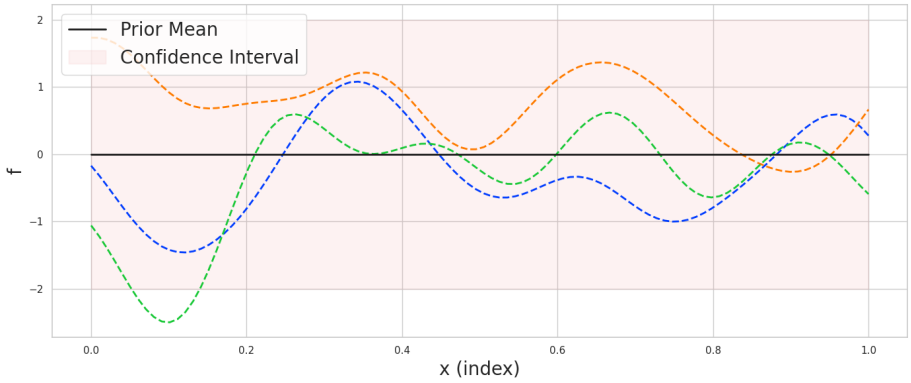


Figure 2.4: This plot describes the GP prior over functions. The realised functions "wiggle" about the prior mean line in bold, while their shapes are generally determined by the kernel function.

To have a clearer understanding of GP, we investigate how it generalises the two ideas in statistics that we know: The multivariate Gaussian distribution and the Bayesian Linear Regression model.

Gaussian Processes as an Infinite-Dimension Multivariate Gaussian

GP can be regarded as a generalisation of the Gaussian distribution, from mean and covariance being a vector and matrix, to have mean function and kernel function. This is equivalent to a Gaussian distribution with an infinite-length mean vector and an infinite-dimension covariance matrix, allowing us to theoretically model datasets of any sizes. However, in practice, we only have a finite N number of observations in a dataset. That means we only need to work with a N-dimensional Gaussian distribution:

$$\mathbf{f} \sim \mathcal{N}(\boldsymbol{\mu}, K), \tag{2.13}$$

where $\mu_i = m(x_i)$ and $K_{ij} = k(x_i, x_j)$ for $i, j = 1, ..., N$. It's helpful to think of GP as a way to visualise this N-dimensional Gaussian, compared to using typical contours plot in the bivariate Gaussians, thereby construction distributions over complex curves, as illustrated in Figure 2.5. It should be highlighted that $k(x_i, x_j)$

can be any non-linear function, not restricted to the linear function $k(x_i, x_j) = cov(x_i, x_j)$ in the usual Gaussian distribution. The only condition is that the resulting covariance matrix has to be positive semi-definite. A popular kernel function is the square-exponential: $k_{\text{SE}}(x, x') = \sigma^2 \exp\left(-\frac{\|x-x'\|^2}{2l^2}\right)$.

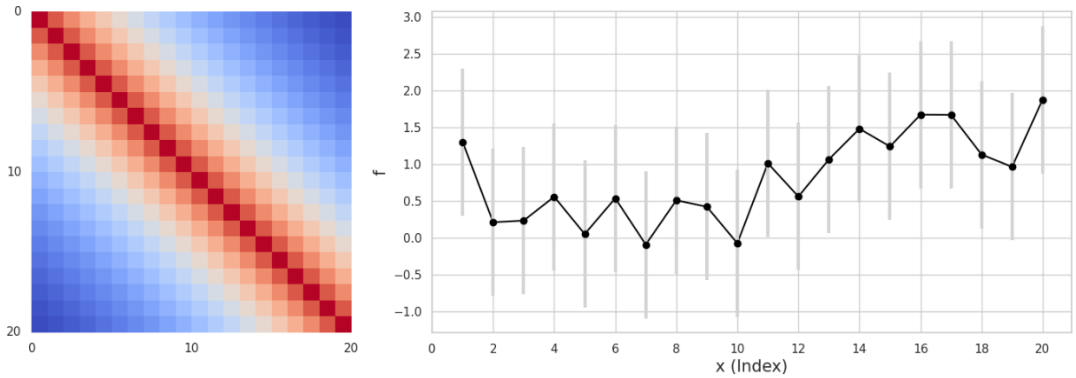


Figure 2.5: LSH: this shows a heatmap of K for $N = 20$, with hotter colors indicating high $k(x, x')$ values for nearby index values of x and x', and colder colors for distant ones. RHS: This describes corresponding function values, where nearby points "wiggle" strongly due to high $k(x, x')$ values. Each function point is also Gaussian-distributed with its own error bar. We can see that, by controlling the matrix K rather than through parameters, we have access to many functional forms for our prior distribution. If we take N to infinity, we obtain Figure 2.4.

By modeling the function values as Gaussian distributed, we can leverage standard results of Gaussians that allow us to do calculations analytically. These standard results include:

1) The marginal densities of the multivariate Gaussian are also Gaussians

Given a joint Gaussian distribution:

$$p(\mathbf{f}, \mathbf{g}) = \mathcal{N}\left(\begin{bmatrix}\mathbf{f}\\ \mathbf{g}\end{bmatrix}; \begin{bmatrix}\mathbf{a}\\ \mathbf{b}\end{bmatrix}, \begin{bmatrix}A & C\\ C^T & B\end{bmatrix}\right), \tag{2.14}$$

then the marginal $\mathbf{f}$ is also Gaussian:

$$p(\mathbf{f}) = \int p(\mathbf{f}, \mathbf{g}) d\mathbf{g} = \mathcal{N}(\mathbf{a}, A). \tag{2.15}$$

2) The conditional densities are also Gaussian

$$p(\mathbf{f}|\mathbf{g}) = \mathcal{N}(\mathbf{a} + CB^{-1}(\mathbf{g} - \mathbf{b}), A - CB^{-1}C^T) \tag{2.16}$$

3) The product of two Gaussians

The product of two Gaussians wrt $\mathbf{x}$ is an unnormalised Gaussian with respect to $\mathbf{x}$:

$$\mathcal{N}(\mathbf{x}|\mathbf{a}, A)\mathcal{N}(\mathbf{x}|\mathbf{b}, B) = \mathcal{N}(\mathbf{a}|\mathbf{b}, A + B)\mathcal{N}(\mathbf{x}|\mathbf{c}, C), \tag{2.17}$$

$$\mathbf{c} = C(A^{-1}\mathbf{a} + B^{-1}\mathbf{b}), \tag{2.18}$$

$$C = (A^{-1} + B^{-1})^{-1}. \tag{2.19}$$

Gaussian Processes as a Nonparametric Regression Model

GPs extend Bayesian Linear Regression by eliminating the need for predetermined basis functions and parameter vectors, but instead employing a kernel to implicitly define an infinite-dimensional space of parameters. For this reason, GPs is called a nonparametric model, not because they do not have any parameters, but because they have an infinite number of parameters, represented as functions.

Zoubin Ghahramani [2013] gives an effective analogy of the transition from parametric to nonparametric model that explains why it is useful: we can view all models as an information channel from training data to predictions. In parametric models, we add a bottleneck θ in this information channel, whereby we only train this bottleneck and all the future prediction will be made through this bottleneck, independently of the training data. This makes the information channel of parametric models not flexible when training data grows to be more complex. Meanwhile, in

nonparametric models, this bottleneck is infinitely large - meaning we can make inferences directly from data. This allows for the complexity of the model only grows as the amount of the data grows, making nonparametrics more flexible yet robust to over-fitting.

2.3.2 Learning and inference

Instead of following the 2-step approach to learn and infer unseen quantities as in the weight-space view, GP follows the function-space view and does this in one step: computing the **predictive posterior**. Let $\mathbf{f}$ be the seen function values of the training inputs X, and let $\mathbf{f}_*$ be the unseen set of function values for the test set inputs, X_*. We are trying to infer $\mathbf{f}_*$, using $\mathbf{f}$, X_* and X. After setting the prior as in equation 2.13, we can firstly set up the joint distribution between $\mathbf{f}$ and $\mathbf{f}_*$ similarly to equation 2.14:

$$\begin{bmatrix} \mathbf{f} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu} \\ \boldsymbol{\mu_*} \end{bmatrix}, \begin{bmatrix} K & K_* \\ K_*^T & K_{**} \end{bmatrix} \right). \tag{2.20}$$

If we have n training points and n_* test points, K_* is then the kernel matrix between training and test inputs with dimension $n \times n_*$. The same applies to the other K and K_{**}. In order to account for random noise in training data, we can add a noise term σ_n^2 to the diagonal of the training kernel matrix and obtain the joint between $\mathbf{y}$ and $\mathbf{f}_*$:

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu} \\ \boldsymbol{\mu_*} \end{bmatrix}, \begin{bmatrix} K + \sigma_n^2 I & K_* \\ K_*^T & K_{**} \end{bmatrix} \right). \tag{2.21}$$

To get the posterior distribution, we need to compute the conditional distribution of $\mathbf{f}_*$ given $\mathbf{f}$, X_* and X. Fortunately, we only to apply standard results of Gaussian

distribution from equation 2.16:

$$\begin{aligned} \mathbf{f}_*|\mathbf{y}, X, X_* &\sim \mathcal{N}(\boldsymbol{\mu}_{\text{post}}, K_{\text{post}}), \\ \text{where } \boldsymbol{\mu}_{\text{post}} &= \boldsymbol{\mu}_* + K_*^T(K + \sigma_n^2 I)^{-1}(\mathbf{y} - \boldsymbol{\mu}), \\ \text{and } K_{\text{post}} &= K_{**} - K_*^T(K + \sigma_n^2 I)^{-1}K_*. \end{aligned} \tag{2.22}$$

From this Gaussian posterior, we can sample new function values with unseen inputs and make inferences. We can see that the posterior variance K_{post} is equal to the prior variance within unseen data subtracts a positive quadratic term that relies on training inputs X. Therefore, the posterior variance has to be smaller than the prior variance as we are exposed to more information. This is an interesting property because it allows GP to give well-calibrated uncertainty estimates. GP "knows when it does not know": where it is given useful (high correlation) information, it will have less uncertainty and vice versa. Another result of this mechanism is that if we we want to infer an unseen point that is far away from the training set, the uncertainty will be high due to low correlation. The posterior can be illustrated in Figure 2.6.

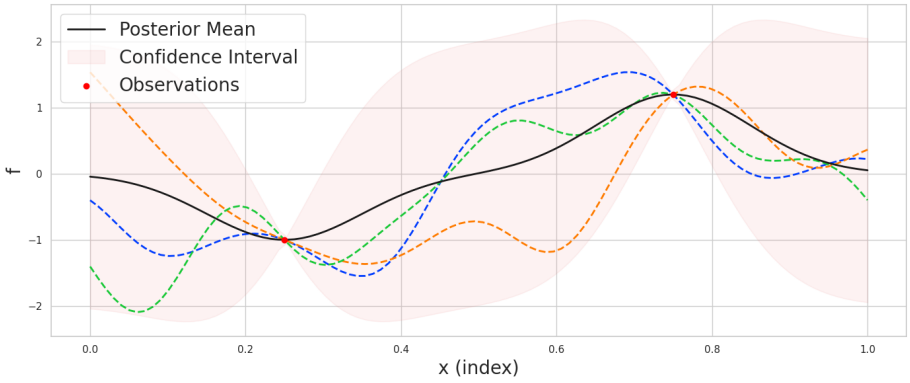


Figure 2.6: This plot describes the posterior distribution over functions after updating the prior from Figure 2.4 with 2 training data points. We can see that after updating, the posterior mean move towards the values of training data. The posterior variance at these updates are also reduced significantly.

2.3.3 Hyperparameter optimisation

The inference process above requires us to select the right prior over functions in order to achieve good performance. However, selecting priors includes two choices. First, it is the discrete choice of the functional forms for mean and kernel functions. Second, these functions can be equipped with **hyperparameters $\boldsymbol{\theta}$**, and we need to adapt them to our data. Note that these hyperparameters are not the same as the parameters in parametrics models. Hyperparameters carry very general properties of the functions, such as smoothness, periodicity and stationarity, instead of specific input covariate information. This adaptation process is often called **Empirical Bayes**, which requires optimisation techniques from section 2.1.2 and the loss function is the **marginal likelihood**.

First, we want to derive the marginal likelihood of the targets, given inputs and the hyperparameter. *Marginal* means marginalising over function values $\mathbf{f}$:

$$\begin{aligned} p(\mathbf{y}|X,\boldsymbol{\theta}) &= \int p(\mathbf{y}|\mathbf{f},X)p(\mathbf{f}|X,\boldsymbol{\theta})d\mathbf{f} \\ &= \int \mathcal{N}(\mathbf{f},\sigma_n^2 I)\mathcal{N}(\boldsymbol{\mu}_\theta,K_\theta)d\mathbf{f} \\ &= \mathcal{N}(\boldsymbol{\mu}_\theta,K_\theta+\sigma_n^2 I), \end{aligned}$$

where the final step used standard results of the Gaussian distribution. Next, we want to take log of the marginal likelihood for mathematical convenience and numerical stability. Log of a Gaussian density gives us:

$$L(\boldsymbol{\theta}) = \underbrace{-\frac{1}{2}\log|K_\theta+\sigma_n^2 I|}_{\text{complexity penalty}} \underbrace{-\frac{1}{2}(\mathbf{y}-\boldsymbol{\mu}_\theta)^T(K_\theta+\sigma_n^2 I)^{-1}(\mathbf{y}-\boldsymbol{\mu}_\theta)}_{\text{model fit}} - \frac{n}{2}\log(2\pi), \quad (2.23)$$

and we can find the **maximum marginal likelihood estimate** for $\boldsymbol{\theta}$, using numerical optimisation methods such as ADAM to maximise L.

We see that in Equation 2.23, the marginal likelihood is composed of three terms: a determinant term - for complexity penalty, a matrix inversion term - for model fit and a constant term. Intuitively, the marginal likelihood loss helps ensure a good fit by balancing the complexity of the model against its fit to the data, thereby preventing over-fitting while accounting for the underlying data pattern.

Cholesky Decomposition

Because we only focus on kernel matrices, which are positive semi-definite, we want a numerically stable and efficient way to calculate the determinant and matrix inversion. Cholesky decomposition is a common choice. Given a symmetric positive semi-definite matrix K, the Cholesky decomposition can be expressed as:

$$K = LL^T$$

where L is a lower triangular matrix with positive diagonal elements, and L^T is its transpose.

The inversion of matrix K using Cholesky decomposition involves solving two triangular systems. The inverse K^{-1} can be computed as:

$$K^{-1} = (LL^T)^{-1} = (L^T)^{-1}L^{-1},$$

The determinant of matrix K can be easily calculated from its Cholesky decomposition. Since the determinant of a triangular matrix is the product of its diagonal entries, the determinant of K is given by:

$$\det(K) = (\det(L))^2,$$

where $\det(L)$ is the product of the diagonal elements of the triangular L. The specific algorithm for Cholesky can be found at Bishop [2006]. The time complexity of finding determinant and matrix inversion are both $\mathcal{O}(n^3)$ with Cholesky, where n is the number of observations.

Performance Evaluation

RMSE (Root Mean Squared Error) and NLPD (Negative Log Predictive Density) are the two main metrics that are used to evaluate the out-of-sample inference of GP regression models. RMSE quantifies the overall deviation of the predictions from the true values, with lower values indicating better predictive performance:

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}$$

Meanwhile, NLPD assesses both the mean and variance of prediction: penalising overconfident and underconfident predictions. Lower NLPD means more appropriately confident prediction capability of the GP model. The NLPD is calculated by taking the negative logarithm of the predictive probability density of the new observations given the model:

$$\text{NLPD} = -\frac{1}{N}\sum_{i=1}^{N}\log(p(y_i|\mathbf{x}_i, \mu, \sigma_i^2))$$

2.3.4 Discussion

We have so far covered the definition and important operations of GPs. In summary, they are a powerful class of probabilistic, nonparametric models with the following advantages:

1. **Flexibility**: GPs do not require specifying any functional form for the mapping we aim to learn. Instead, we give very high-level properties of the mapping via the mean and kernel functions. This approach allows GPs to adapt flexibly to the data.

2. **Robustness to Over-fitting**: The complexity of the mapping that GPs learn scales with the size of the dataset. As the dataset grows, so does the kernel ma-

trix. Additionally, the optimisation of hyperparameters inherently incorporates regularisation effects, further mitigating the risk of over-fitting.

3. **Well-Calibrated Uncertainty Estimates**: GPs provide low uncertainty estimates where the given data are informative, and low certainty when data is not informative and if we go further from training data.

Meanwhile, we can see that there are three challenges when applying GPs:

1. **Kernel Design**: The choice of the kernel function is crucial in GP but finding an optimal kernel that improves on default kernels such as square-exponential can be challenging. Common practice is to compare models with metrics such as NLPD and RMSE. This makes the process of kernel selection and tuning a significant bottleneck in GP modeling.

2. **Scalability**: GPs suffer from poor scalability to large datasets. The computational complexity of training a GP model is $\mathcal{O}(n^3)$ that arises from the inversion of the kernel matrix, making the computation infeasible for large datasets.

3. **Multi-Task Regression**: GPs can be extended to multi-task regression scenarios, where the goal is to predict multiple tasks simultaneously. This extension involves modeling the covariance not only between inputs (as in standard GPs) but also between different tasks, i.e. different turbines. By leveraging covariance across tasks, multi-task GPs can improve prediction accuracy, particularly in cases where some tasks have sparse data.

In the next three chapters, we will discuss the modern literature on tackling these three problems. In Chapter 3, we discuss basic and advanced kernel functions that can optimally account for complex patterns in data. Chapter 4 will delve into techniques for effective multi-task GP modeling, including the construction of cross-task kernels and methods to address computational challenges. In Chapter 5, we explore sparse approximation methods that trade off some model accuracy for speed-ups in training.

Chapter 3

Kernel Design

Summary

Selecting the appropriate kernel is crucial for setting an optimal prior in GP models, enabling effective inferences about unseen data. However, not any arbitrary function of $\mathbf{x}$ and $\mathbf{x}'$ can be a kernel functions, but it is subject to mathematical constraints. There are some common valid kernel functions with unique properties that can be used on GP. This chapter will firstly explore the mathematical requirements for kernel functions and examine these commonly used kernels in GP applications. After that, we will discuss the cutting-edge developments in kernel design, the Spectral Mixture Kernel and its non-stationary extensions, to increase model expressiveness and automate pattern discoveries.

3.1 Making New Kernels from Old

3.1.1 Property of Kernels

First, we will formally define a kernel function. A kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ for a Gaussian Process, where $\mathcal{X}$ is the index set corresponding to the input space, is a

function that computes the covariance between any two points $x, x' \in \mathcal{X}$. The kernel function has to be **positive semi-definite (PSD)** to be valid. This means: for any finite set of points $x_1, x_2, \ldots, x_n \in \mathcal{X}$ and any set of real numbers $a_1, a_2, \ldots, a_n$, the matrix formed by applying the kernel function to every pair of points $K_{ij} = k(x_i, x_j)$ must satisfy the condition:

$$\sum_{i=1}^{n} \sum_{j=1}^{n} a_i a_j k(x_i, x_j) \geq 0$$

To check if a kernel matrix is PSD, we verify that all its eigenvalues are non-negative. This can be done by computing the eigenvalues of the matrix with methods such as Singular Value Decomposition and ensuring none of them are less than zero.

3.1.2 Standard Kernels

There are a few simple, out-of-the-box kernels that can be efficiently implemented on regression problems. They are easy and fast to train, but might not be the optimal choice for a given dataset. However, these are good baseline models for making comparisons with more sophisticated kernels. We make choices between these kernel functional forms based on their smoothness and periodicity, presented in Figure 3.1. Other properties include stationarity and monotony, which will be discussed in detailed in the next section.

Squared-Exponential Kernel

The Squared-Exponential (SE) kernel, also known as the Gaussian or Radial Besis Function (RBF) kernel, is characterised by its smoothness and infinite differentiability. This kernel is widely used for modeling functions that are smooth across their domain. The formula for the SE kernel between two data points at index x and x' is given by:

$$k_{\text{SE}}(x, x') = \sigma^2 \exp\left(-\frac{\|x - x'\|^2}{2l^2}\right), \tag{3.1}$$

where σ^2 is the variance parameter that controls the overall variance of the function and l is the length scale parameter that determines the smoothness of the function. A larger l leads to smoother functions. The SE kernel is sensitive to the choice of l, influencing the correlation between points and thus the smoothness of the function.

Matérn Kernel

The Matérn kernel provides a more flexible approach to modeling functions with varying degrees of smoothness. It is parameterised by a smoothness parameter ν and a length scale l. The general form of the Matérn kernel is given by:

$$k_{\text{Matérn}}(x, x') = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu}\|x - x'\|}{l} \right)^{\nu} K_{\nu} \left(\frac{\sqrt{2\nu}\|x - x'\|}{l} \right), \tag{3.2}$$

where Γ is the gamma function, K_{ν} is the modified Bessel function of the second kind, and $\|x - x'\|$ is the Euclidean distance between the points at index x and x'. The length scale parameter l affects the correlation between points in a manner similar to the SE kernel. The parameter $\nu > 0$ controls the smoothness of the function; larger values of ν result in smoother functions. By substituting $\nu = 0.5$ and $\nu = 1.5$, we obtain:

$$k_{\text{Matérn 0.5}}(x, x') = \exp\left(-\frac{\|x - x'\|}{l} \right)$$

$$k_{\text{Matérn 1.5}}(x, x') = \left(1 + \frac{\sqrt{3}\|x - x'\|}{l} \right) \exp\left(-\frac{\sqrt{3}\|x - x'\|}{l} \right),$$

where Matérn 0.5 corresponds to functions that are once differentiable and Matérn 1.5 corresponds to functions that are twice differentiable.

Periodic Kernel

The Periodic kernel is specifically designed to model functions with periodic behavior. The kernel is defined as:

$$k_{\text{Periodic}}(x, x') = \sigma^2 \exp\left(-\frac{2\sin^2(\pi|x - x'|/p)}{l^2}\right), \tag{3.3}$$

where σ^2 is the variance parameter, l is the length scale, and p represents the period of the function. The kernel captures periodic patterns within the data, making it ideal for tasks involving time series or any phenomena with regular intervals. The hyperparameter p directly influences the modeled period, allowing for the adaptation to the specific periodicity of the data.

Automatic Relevance Determination

Automatic Relevance Determination (ARD) is a technique used to automatically identify and give different weights to the inputs based on their relevance to the output. This is achieved by extending the parameterisation of kernel functions to allow a separate length scale hyperparameter for each input dimension.

In kernels such as the SE or the Matérn kernel, ARD can be implemented by modifying the distance metric in the kernel function to include a distinct length scale for each input feature. For example, the SE kernel with ARD is given by:

$$k_{\text{SE-ARD}}(x, x') = \sigma^2 \exp\left(-\frac{1}{2}\sum_{i=1}^{D}\frac{(x_i - x_i')^2}{l_i^2}\right), \tag{3.4}$$

where D is the number of input dimensions, x_i and x_i' are the i-th dimensions of input vectors x and x', respectively, σ^2 is the variance parameter, and l_i is the length scale associated with the i-th input dimension.

The ARD mechanism allows the GP model to adjust the influence of each input dimension on the prediction by learning a separate length scale for each dimension. This makes ARD particularly useful for feature selection and improving model generalisation by preventing over-fitting.

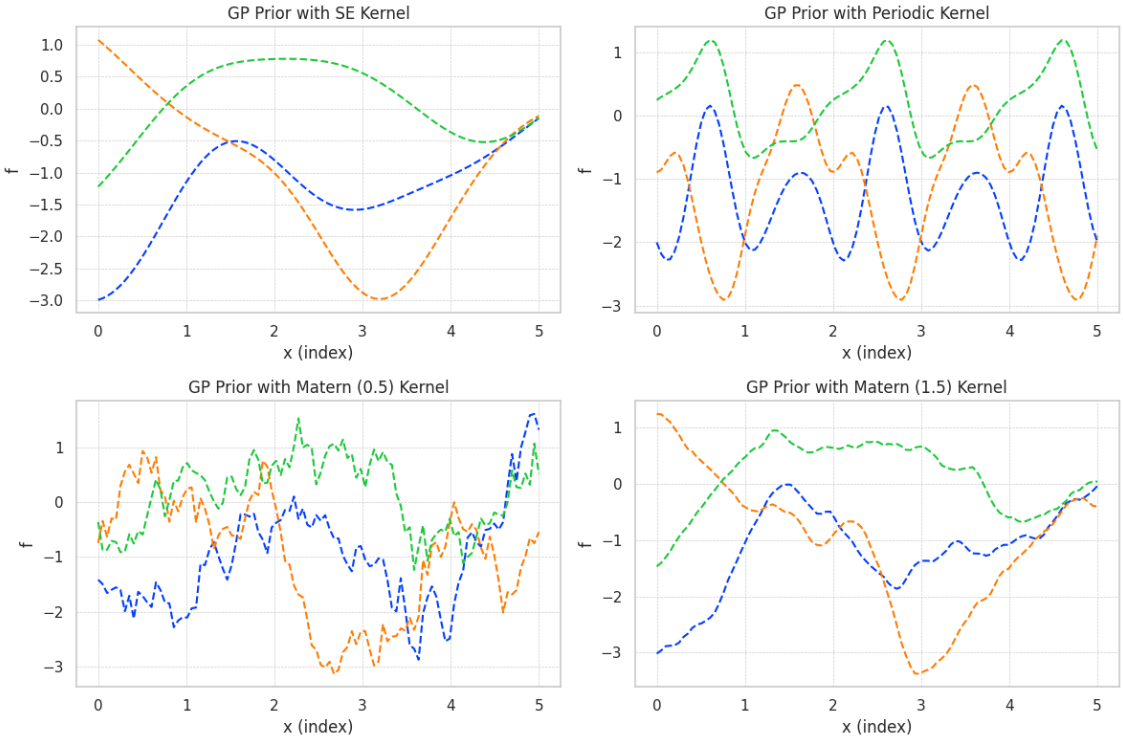


Figure 3.1: **Top-left**: Smooth and continuous functions, highlighting the SE kernel's assumption of high smoothness. **Top-right**: Wavelike patterns, showcasing the Periodic kernel's capacity for capturing regular cycles. **Bottom-left**: Jagged fluctuations, illustrating the Matern kernel (0.5) modeling rough, uneven behavior. **Bottom-right**: Balanced smoothness and roughness, depicted by the Matern kernel (1.5) for moderate smoothness.

3.1.3 Numerical Instability

When working with GPs in practice, numerical challenges frequently arise given the PSD requirement. One such challenge is the presence of an ill-conditioned kernel matrix, which happens when there's a high ratio of the largest to the smallest eigenvalue. Another issue is a near-singular matrix, characterised by having one or more eigenvalues that are very close to zero or exactly zero. This suggests that the matrix

is close to losing its capacity to be inverted. To mitigate these issues, a common solution involves adding a small value (between 1×10^{-6} to 1×10^{-4}), known as jitter, to the diagonal elements of the kernel matrix while optimising the hyperparameters, or making inferences. It helps to increase the minimum eigenvalue and improve stability.

3.1.4 Combining Kernels

The properties of kernel functions in the previous sections can not only be used for checking whether a kernel is valid, but also to compose rules for combining simple kernels to achieve more complex ones as long as the PSD property is preserved. We call this the **closure property** of the kernel under such operations. Examples of such operations are **sums** and **products** of multiple PSD kernel functions:

$$k_{\text{sum}}(x, x') = k_1(x, x') + k_2(x, x'),$$
$$k_{\text{product}}(x, x') = k_1(x, x') \times k_2(x, x').$$

Sum of kernels

We can add a SE kernel and a Periodic kernel to capture both smooth variations and periodic patterns:

$$k_{\text{sum}}(x, x') = k_{\text{SE}}(x, x') + k_{\text{Periodic}}(x, x'), \tag{3.5}$$

with hyperparameters σ^2_{SE}, l_{SE}, $\sigma^2_{\text{Periodic}}$, l_{Periodic}, and p.

Product of kernels

Consider a dataset where a linear trend is present, but the amplitude of this trend varies smoothly and increasingly across the domain. To model such data, we can multiply a Linear kernel $k_{\text{Linear}}(x, x') = \sigma_b^2 + \sigma_v^2(x - c)^\top(x' - c)$, which captures the linear trends, with a SE kernel, which captures smooth variations.

$$k_{\text{product}}(x, x') = k_{\text{Linear}}(x, x') \times k_{\text{SE}}(x, x'), \tag{3.6}$$

with hyperparameters for the Linear kernel: bias variance σ_b^2, variance of the linear part σ_v^2, and intercept c, and for the SE kernel: σ_{SE}^2 and l_{SE}.

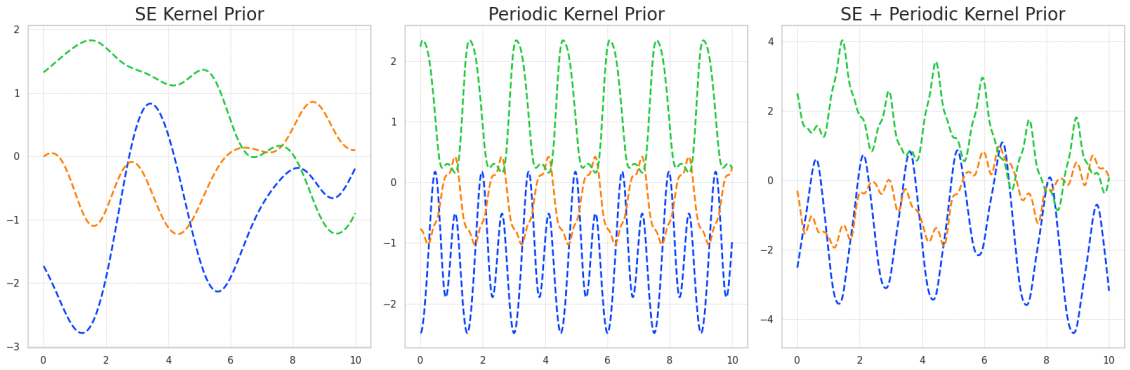


Figure 3.2: Sampled functions from GP priors using an SE kernel (left) and a Periodic kernel (middle), and the diverse patterns obtained by summing these kernels (right)

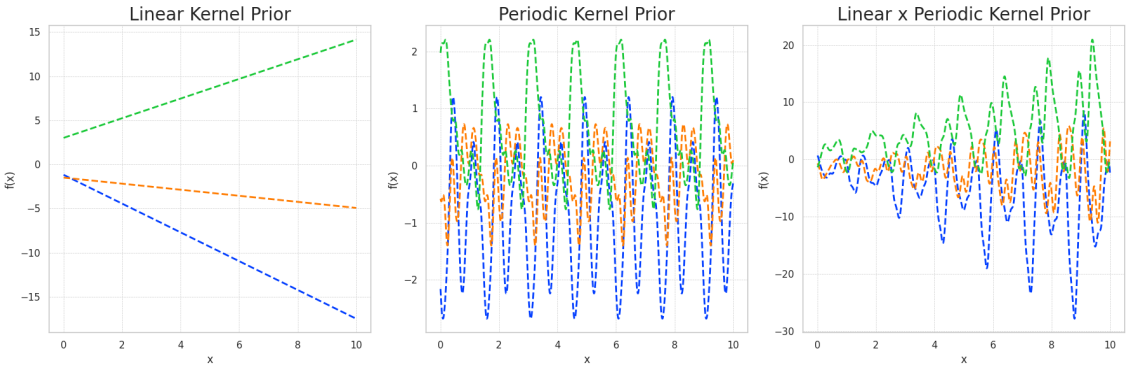


Figure 3.3: Sampled functions from GP priors using an Linear kernel (left) and a Periodic kernel (middle), and the patterns obtained by taking product of these kernels (right)

Besides summation and product, there are other operations to compose new kernels: multiplying with scalars, exponentiation or even composition with other functions, which can be found in Bishop [2006]. As long as the resulting kernel function

is PSD, we can hand-craft a new kernel function based on our own understanding of data patterns. However, if the patterns are sophisticated, it's challenging to impose an effective inductive bias with such a hand-crafting process. In the next section, we will discuss an expressive kernel construction that can automatically recover a range of different kernels, without the need for manual composition.

3.2 Spectral Mixture Kernel

From all the common kernels above, we note that they can all be written as a function of relative location between data points: $k(\tau) = k(x - x')$. This class of kernel is called **stationary kernels**. In general, parameterising kernel functions directly can be inconvenient because we need to ensure they are always PSD. If we are only working with stationary kernels, we can alternatively parameterise the **spectral density**, which is easier to handle as it requires to have non-negative and symmetric properties. The spectral density quantifies how different frequencies contribute to shaping the correlation structure, influencing how smooth or wavy the process appears. We use a principle called Bochner's Theorem [Bochner, 1959] to obtain these spectral densities:

Theorem 1. *Let $k : \mathbb{R}^P \to \mathbb{C}$ be a function that represents the covariance of a stationary process on $\mathbb{R}^P$ if and only if it can be expressed in the form:*

$$k(\tau) = \int_{\mathbb{R}^P} e^{2\pi i s^\top \tau} \, d\psi(s),$$

where $\tau = x - x'$ and ψ denotes a positive and finite measure.

If $S(s)$ is the density of ψ, then S is called the spectral density of k. Intuitively, the theorem says that as long as the kernel we have is stationary, we can decompose it into spectral densities. The functions $k(\tau)$ and $S(s)$ are known to be Fourier duals

[Chatfield, 2003], satisfying the following relationships:

$$k(\tau) = \int S(s) e^{2\pi i s^\top \tau} ds, \tag{3.7}$$

$$S(s) = \int k(\tau) e^{-2\pi i s^\top \tau} d\tau. \tag{3.8}$$

Equation (3.8) is called Inverse Fourier Transform (IVF). The Fourier duals resemble a bridge between the kernel function and its spectral density using the Fourier transform. If Fourier transform is applied to a kernel, we obtain its spectral density. And we can do the reverse as well by taking IVF on the spectral density to get the kernel. For example, applying (3.8) to the SE kernel will yield:

$$S_{\text{SE}}(s) = \left(2\pi\ell^2\right)^{P/2} \exp\left(-2\pi^2\ell^2 s^2\right).$$

The Fourier duals can be leveraged to construct a new kernels through parameterising spectral densities: if we can design a valid spectral density that can represent a wide range of different other spectral densities, we can apply IVF to get a stationary kernel with the same characteristics. Wilson and Adams [2013] proposed using a mixture of Gaussians spectral densities, which is effectively a weighted sum of Gaussian densities. This was motivated by two reasons: (1) The spectral density of SE kernel is a Gaussian centered on the origin. If we use Gaussian densities with non-zero means, we can generalise to a more flexible kernel (2) Mixture of Gaussians allows us to recover many other continuous distributions [Kostantinos N., 2017]. It was found that standard kernels can be recover with fewer than 10 mixtures. This this called the Spectral Mixture (SM) kernel. Its derivation straightforward. First, we consider a fundamental case of 1-D Gaussian spectral density:

$$\phi(s; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(s-\mu)^2}{2\sigma^2}\right),$$

and establish a mixture of spectral densities:

$$S(s) = \frac{\phi(s) + \phi(-s)}{2},$$

acknowledging that spectral densities of symmetric kernels are reflected across $s = 0$. By inserting $S(s)$ into Equation (3.8), we derive:

$$k(\tau) = \cos(2\pi\tau\mu) \exp\left(-2\pi^2\tau^2\sigma^2\right).$$

When $\phi(s)$ is alternatively a composition of Q Gaussians in $\mathbb{R}^P$, with each q-th Gaussian characterised by a mean vector $\mu_q = (\mu_q^{(1)}, \ldots, \mu_q^{(P)})$ and a diagonal covariance matrix $V = \text{diag}(v_q^{(1)}, \ldots, v_q^{(P)})$, where w_q signify the mixture weights, and τ_p represents the p-th entry of the P-dimensional distance vector $\tau = x - x'$, the kernel is expressed as:

$$k(\tau) = \sum_{q=1}^{Q} w_q \cos(2\pi \boldsymbol{\mu}_q^\top \tau) \prod_{p=1}^{P} \exp\left(-2\pi^2\tau_p^2 v_q^{(p)}\right). \tag{3.9}$$

This kernel satisfies the PSD condition thanks to Fourier Duals. We interpret the hyperparameters of the kernel: the weights w_q are relative contribution of each component within the mixture. The reciprocal of μ_q, which is $\frac{1}{\mu_q}$, represents the periodicity of each component. Also, $\frac{1}{\sqrt{v_q}}$ represents the length scales, which effectively describe the rate at which a component's value changes in relation to the inputs x.

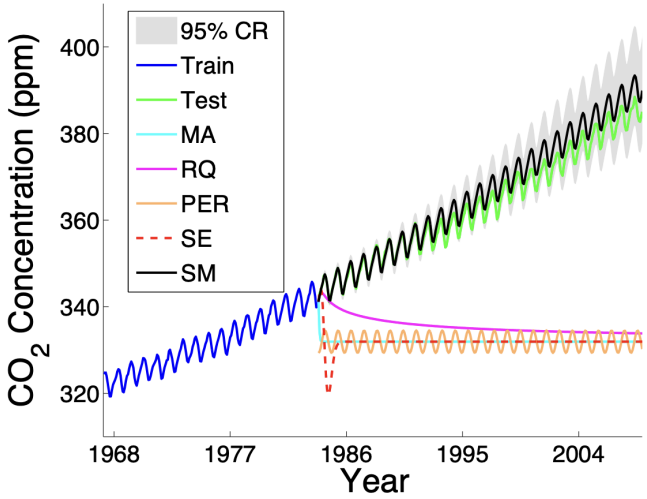


Figure 3.4: Comparative analysis of GP with different stationary kernels extrapolating CO2 concentration trends from 1968 to 2004 [Wilson and Adams, 2013]

The strength of the SM kernel lies in its extrapolation capability. In the original research paper, a range of stationary kernels Matérn (MA), Rational Quadratic (RQ), Periodic (PER), Square-Exponential (SE) and Spectral Mixture (SM) were tested on extrapolating the CO2 dataset [Keeling and Whorf, 2004]. The SM kernel, shown in black, was able to capture both the periodic and upward trends in the test data, while other kernels could only capture either the periodic pattern or no pattern. The alternative to match the performance of Spectral Mixture kernel in this context is to have a product between a linear and periodic kernel, but as we generalise to more complex patterns, manual composition becomes infeasible.

The tradeoff for increased flexibility in the Spectral Mixture kernel is an increased number of hyperparameters. We have one set of three hyperparameters for each mixture, leading to a $Q \times 3$ hyperparameter matrix. This practically poses more risks of over-fitting to the regression model, and maken the optimisation process more sensitive to initialisation (i.e. local minima) compared to standard kernels with fewer hyperparameters. One approach to alleviate this is to train the GP with multiple initialisations and pick the best results, so that we can explore potentially better optima solutions.

3.3 Non-Stationary Spectral Kernel

Stationary kernels have a strong record to perform well in time series prediction scenarios, but there is space for improvements in those where the properties of the true function vary with the absolute position of input, not just the relative distance. This is a common phenomena in temporal or spatiotemporal regression problems in climate sciences, such as rainfall prediction [Lalchand et al., 2023] and wind power prediction [Stetco et al., 2019], where the smoothness or periodicity of the function can become more rough in some seasons when environmental conditions becomes more extreme. Using stationary kernel prior in these cases can leads to under-fitting in multiple perriods of extrapolation. This problem motivates the development of

non-stationary methods for GP models. Attempts to do this include transforming inputs to control stationary [Snoek et al., 2014], doing local approximation [Gramacy and Lee, 2009], and designing input dependent kernels. We will focus on the last approach in this work.

Remes et al. [2017] proposed a non-stationary extension of the SM kernel, called the Generalised Spectral Mixture (GSM) kernel. The GSM kernel allows each hyperparameter of the SM kernel w_i, ν_i, μ_i to be a function of input index $w_i(x), \ell_i(x)$ and $\mu_i(x)$, where $\ell_i(x) = \frac{1}{\sqrt{\nu_i(x)}}$ for notation convenience. Each of the hyperparameter can be given a GP prior with 0 mean function and RBF kernel:

$$\begin{aligned}\log w_i(x) &\sim \mathcal{GP}(0, k_w(x, x')),\\ \log l_i(x) &\sim \mathcal{GP}(0, k_l(x, x')),\\ \operatorname{logit} \mu_i(x) &\sim \mathcal{GP}(0, k_\mu(x, x')).\end{aligned}$$

The hyperparamers are transformed before assigning the GP prior so that their range is restricted to be non-negative for $w_i(x)$ and $\ell_i(x)$, and between 0 and Nyquist frequency for $\mu_i(x)$. Next, we can substitute these into the SM kernel formula in equation (3.9), which gives:

$$k_{\text{GSM}}(x, x') = \sum_{i=1}^{Q} w_i(x) w_i(x') k_{\text{Gibbs},i}(x, x') \cos\left(2\pi(\mu_i(x)x - \mu_i(x')x')\right), \tag{3.10}$$

where the Gibbs kernel follows:

$$k_{Gibbs,i}(x, x') = \sqrt{\frac{2 l_i(x) l_i(x')}{l_i(x)^2 + l_i(x')^2}} \exp(-\frac{(x - x')^2}{l_i(x)^2 + l_i(x')^2})$$

Recall that the exponential component in the SM kernel because stems from the Gaussian spectral density. The Gibbs kernel [Gibbs, 1997] is the non-stationary counterpart of that component with input-dependent lengthscales. The GSM kernel set up above is for one-dimensional inputs. We can extend to P-dimension using:

$$k_{\text{GSM}}(x, x'|\theta) = \prod_{p=1}^{P} k_{\text{GSM}}(x_p, x'_p|\theta_p), \tag{3.11}$$

where $x, x' \in \mathbb{R}^P$, $\theta = (\theta_1, \ldots, \theta_P)$ are the input-dependent hyperparameter for each mixture $\theta_p = (w_{ip}, l_{ip}, \mu_{ip})_{i=1}^{Q}$ of the N-dimensional realisations $w_{ip}, l_{ip}, \mu_{ip} \in \mathbb{R}^N$ per dimension p.

Validity of GSM Kernel

The setup of the GSM kernel contradicts the constraints of the Fourier Duals, as we are now working with non-stationary kernels. In the derivation by Remes et al. [2017], a generalised version of Fourier transform was used:

$$k(x, x') = \int_{\mathbb{R}} \int_{\mathbb{R}} e^{2\pi i(xs - x's')} \mu_S(ds, ds'),$$

where μ_S is the Lebesgue-Stieltjes measure. This allowed for applying Fourier transform to go from the mixture of multivariate Gaussian spectral densities to the GSM kernel, but the result of the Inverse Fourier Transform no longer holds. The full technical details of this proof is not discussed here and can be viewed in the original paper. However, despite the theoretical incompleteness, we can still guarantee that this is a PSD kernel because equation (3.10) is a product between three PSD terms, thus can be applied in GP.

Hyperparameter Optimisation

Due to the complex hierarchical structure of GPs, we can no longer conduct maximum marginal likelihood inference because in order to obtain the marginal likelihood loss, we need to marginalise out not only f, but also all the hyperparameter functions. This causes the integral to be intractable. An alternative is to conduct MAP inference, where we maximise log of the posterior $\log p(\theta|y) \propto \log p(y|\theta)p(\theta)$, where only f was integrated out, using numerical optimisers like ADAM:

$$\mathcal{L}(\theta) = \log \left(\mathcal{N}(y|0, K_\theta + \sigma_n^2 I) \prod_{i,p=1}^{Q,P} \mathcal{N}(w_{ip}|0, K_{wp}) \mathcal{N}(\mu_{ip}|0, K_{\mu_p}) \mathcal{N}(\ell_{ip}|0, K_{\ell_p}) \right),$$

Similar to the SM kernel, using the GSM poses two tradeoffs of over-fitting and local optima solutions due to having many parameters involved. It is therefore useful to experiment with multiple number of mixtures and optimise with multiple initialisations to avoid local minima.

3.4 Discussion

In Chapter 3, we have covered the kernel design toolkit that can be experimented with in regression problems. However, in applications such as wind power prediction, often times we can leverage inputs of multiple wind turbines at the same time to produce enhance the model. This is an active research area in GP, which we will discuss in the next chapter.

Chapter 4

Multi-Task Gaussian Processes

Summary

Performance of a GP model can not only be improved with flexible kernels. Wind farm data often come with multiple different turbines, and we can leverage information transfer across turbines to enhance prediction performance, as opposed to training each turbine independently. A way to do this is to use multi-task Gaussian Processes (MTGP). This chapter explores the general paradigm of multi-task learning, then specifically in the GP context. We then explore methods to model cross-task covariances, and modern literature in MTGP. The main ideas in below are summarised from Alvarez et al. [2012] and Dai et al. [2024] unless stated otherwise.

4.1 Multi-Task Learning

In scalar supervised learning, we typically deal with datasets consisting of paired inputs and outputs, denoted as $G = \{(x_1, y_1), \ldots, (x_N, y_N)\}$, where each x_i corresponds to a single output y_i. When we have multiple different class of outputs (e.g. predict power output of multiple wind turbines), we have two options. One is to build multiple independent models to predict separately for each turbine. The other

is conducting multi-task or multi-output learning, depending on the data at hand. In multi-task learning, referred to as **heterotopic**, each task, like predicting the output of different wind turbines, has its unique input dataset. On the other hand, multi-output learning, also known as **isotopic**, would use a shared input dataset to predict all the outputs. These concepts originated from the geo-statistics literature [Wackernagel, 2003].

Our interest lies in multi-task learning, not multi-output learning, because we are looking at a scenario where each wind turbine has its own set of input data that is distinct from the inputs of others. This means for each wind turbine, or task $d = 1, \ldots, D$, there is a separate training set $G_d = \{X, Y\} = \{(X_d, \mathbf{y}_d), \ldots, (X_D, \mathbf{y}_D)\}$ with N data points for each task $X_d = \{\mathbf{x}_{d,n}\}_{n=1}^{N}$. Each input X_d is exclusive to its output $\mathbf{y}_d$.

4.2 Multi-Task Gaussian Processes

GPs can extend naturally from just functions to **vector-valued functions** in the multi-task framework. In the simpler single-task case, a GP prior can be given by a mean function and a kernel function. When we transition to multi-task GP, the concept scales up and the prior can be given by:

$$\mathbf{f}(X) \sim \mathcal{GP}(\mathbf{m}(X), K(X, X)), \tag{4.1}$$

where $\mathbf{m}(X)$ concatenates mean functions for each task and $K(X, X)$ is a block-partitioned matrix including kernel functions for each pair of task [Bonilla et al., 2007]. $K(X, X)$ is an $ND \times ND$ matrix with $(K(x_i, x_j))_{d,d'}$ being entry kernel functions, for $i, j = 1, \ldots, N$ and $d, d' = 1, \ldots, D$. The diagonal entries are the kernel matrices of each task, and the off-diagonal matrices are the cross-task covariance

matrices. In explicit form:

$$K(X,X;\boldsymbol{\theta}) = \begin{bmatrix} (K(X_1,X_1;\boldsymbol{\theta}_{11}))_{11} & \dots & (K(X_1,X_D;\boldsymbol{\theta}_{1D}))_{1D} \\ (K(X_2,X_1;\boldsymbol{\theta}_{21}))_{21} & \dots & (K(X_2,X_D;\boldsymbol{\theta}_{2D}))_{2D} \\ \vdots & \ddots & \vdots \\ (K(X_D,X_1;\boldsymbol{\theta}_{D1}))_{D1} & \dots & (K(X_D,X_D;\boldsymbol{\theta}_{DD}))_{DD} \end{bmatrix}$$

Inference

In finite dimensions, the likelihood can be written as:

$$p(Y|\mathbf{f},X,\Sigma) = \mathcal{N}(\mathbf{f}(X),\Sigma\otimes I_N), \tag{4.2}$$

Let $\boldsymbol{\Sigma} = \Sigma\otimes I_N$ and omit $\boldsymbol{\theta}$ for brevity, predictive posterior will be:

$$p(\mathbf{f}_*|x_*,X,Y,\boldsymbol{\theta}_M) = \mathcal{N}(\bar{\mathbf{f}}_*,V_*) \tag{4.3}$$

where: $\begin{cases} \bar{\mathbf{f}}_* = K(\mathbf{x}_*,X)(K(X,X)+\boldsymbol{\Sigma})^{-1}Y \\ V_* = K(\mathbf{x}_*,\mathbf{x}_*) - K(\mathbf{x}_*,X)(K(X,X)+\boldsymbol{\Sigma})^{-1}K(X,\mathbf{x}_*). \end{cases}$

Hyperparamter Optimisation

Similar to single-task GP, we can optimise the prior with past data by minimising the marginal log likelihood:

$$\mathcal{L}_{\{\theta,\boldsymbol{\Sigma}\}} \propto -\tilde{Y}^T(K(X,X;\boldsymbol{\theta})+\boldsymbol{\Sigma})^{-1}\tilde{Y} + \log|K(X,X;\boldsymbol{\theta})+\boldsymbol{\Sigma}|, \tag{4.4}$$

where $\tilde{Y} = Y - \mathbf{m}(X)$, and $\boldsymbol{\Sigma} = \Sigma\otimes I_N$. Note that this loss function also inherits from the set up of single-task GP where there is an automatic trade-off between model fit and complexity penalty. From the set-up above, one important remark is that the kernel $K(X,X;\boldsymbol{\theta})$ needs to be parameterised in a certain way in order to be PSD and compatible with the multi-task learning objective. The next four sections will discuss the main frameworks for designing such kernels.

4.2.1 Separable Kernel

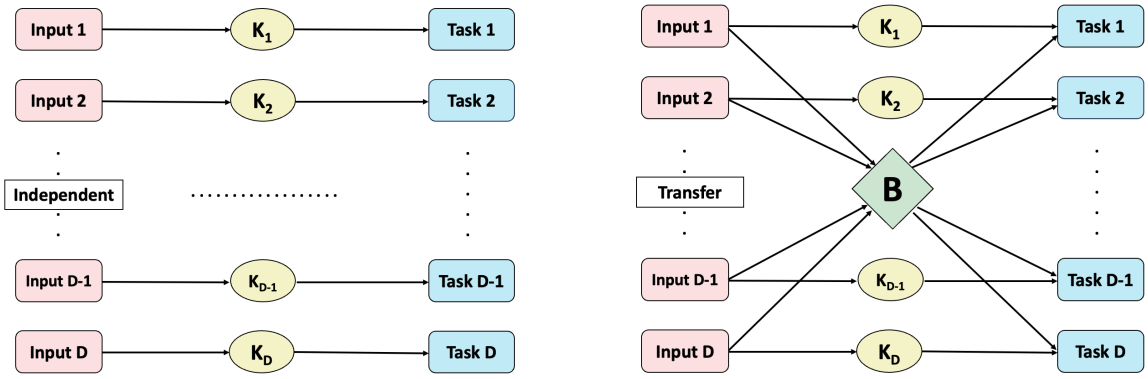
A simple class of MTGP kernel that is easy to work with is the separable, or sum of separable, kernel(s). Let B be a $D \times D$ PSD matrix, where D is the number of tasks, the separable kernel matrix can be written as:

$$K(X, X) = B \otimes k(x, x'), \tag{4.5}$$

and the sum of separable kernel can be written as:

$$K(X, X) = \sum_{q=1}^{Q} B_q \otimes k_q(x, x'), \tag{4.6}$$

The specific meaning of Q will be discussed in the next section. For now, we should focus on the fact that the matrix B, known as the **coregionalisation matrix**, will model the cross-task covariance and the kernel $k(x, x')$ can be shared among all the tasks, or we also have the choice to model them separately for each task. Therefore, the coregionalisation matrix, together with the per-task kernel function, is something to be learnt during the hyperparameter optimisation process. The role of B with the kernel functions can be illustrated in Figure 4.1.



(a) Independent GP per Task

(b) MTGP with coregionalisation matrix

Figure 4.1: Comparison of inference with indepdendent GPs and MTGP

4.2.2 Linear Model of Coregionalisation

The Linear Model of Coregionalisation (LMC) [Journel and Huijbregts, 1978] is a framework for constructing a multi-task kernel by linearly combining multiple single-task kernels, each weighted by $B_q = \boldsymbol{\theta}_q^{(r)}(\boldsymbol{\theta}_q^{(r)})^T$, which is an outer product. The kernel for the multi-task GP in LMC is constructed as follows:

$$K_{LMC}(X, X) = \sum_{q=1}^{Q} \sum_{r=1}^{R_q} \boldsymbol{\theta}_q^{(r)}(\boldsymbol{\theta}_q^{(r)})^T \otimes k_q(x, x'),$$

where:

- Q is the number of groups of GPs.
- R_q is the number of tasks in the q-th group.
- $\boldsymbol{\theta}_q^{(r)}$ is the vector of weights for the r-th GP in the q-th group.
- $k_q(x, x')$ is the kernel function for the q-th group.
- $\otimes$ denotes the Kronecker product. Note that in the case where we have distinct output and input for each task, we can use the Hadamard product $A \odot B$

This is the most general class of linear coregionalisation as we are able to control both the number of groups of task as well as the number of task in each group. This could be useful in modelling the linear relationship between the kernel functions of multiple wind turbines. For example, if we have 10 wind turbines, 2 of them sit on one farm, another 5 sit on a hill and the rest sit by a river. It would be intuitive to group them into three groups, each with different number of turbines, and each group can be modelled by different kernels according to their geographical environment.

4.2.3 Intrinsic Model of Coregionalisation

The Intrinsic Model of Coregionalisation (IMC) [Goovaerts, 1997] assumes a single shared GP across all tasks, but with different weightings for each task. The kernel in

the IMC is a special case of the LMC with $Q = 1$, meaning there is only one group of tasks and all of them share the same kernel:

$$K_{IMC}(X, X) = \sum_{r=1}^{R} \boldsymbol{\theta}^{(r)} (\boldsymbol{\theta}^{(r)})^T \otimes k(x, x'),$$

where:

- R is the number of tasks in the single group.
- $\boldsymbol{\theta}^{(r)}$ is the weight vector for the r-th member.
- $k(x, x')$ is the shared kernel function across all tasks.

The IMC model is more limiting than the LMC because it treats every basic covariance between points as equally important when building autocovariances and cross covariances for outputs. However, thanks to the characteristics of the Kronecker product, the calculations needed for inference is simplified significantly. The IMC method will be useful if we believe all the members should share the same kernel, for example if we have 6 wind turbines lying close to each other on the same farm. This will be our method of choice later in the experiment.

4.2.4 Semiparametric Latent Factor Model

The Semiparametric Latent Factor Model (SLFM) [Teh et al., 2005] represents each task as a combination of latent functions, where each latent function has its own kernel and task-specific weighting:

$$K_{SLFM}(X, X) = \sum_{q=1}^{Q} \boldsymbol{\theta}_q \boldsymbol{\theta}_q^T \otimes k_q(x, x'),$$

where:

- Q is the number of groups.
- $\boldsymbol{\theta}_q$ is the weight vector of the q-th group.

- $k_q(x, x')$ is the kernel function for the q-th latent function.

The kernel in the SLFM is a special case of the LMC with $R = 1$. Meaning each task is its own group and inherit a different kernel. When imagine the application of this model in wind farms, it is not quite feasible because the wind turbines would be likely to be very separated from each other and need to be modelled by different kernels.

4.3 Spectral and Non-Stationary extensions

We can see that from LMC, ICM and SLFM, the cross covariances are modelled to be linear combination of the covariance of a subgroup of tasks. There is a more complex approach that leverages the idea of the Spectral Mixture kernel from 3.9, called the Multi-Output Spectral Mixture kernel [Parra and Tobar, 2017]. This introduces complex-valued cross-spectral densities, allowing it to capture delays and phase differences between channels more expressively. A non-stationary extension is also available in Altamirano and Tobar [2022], where the Multi-Output Harmonizable Spectral Mixture (MOHSM) kernel is introduced, extending the model's applicability to both stationary and non-stationary processes by leveraging harmonisable kernels for enhanced flexibility and expressiveness.

4.4 Discussion

This chapter discussed an effective framework to leverage the covariance structure across multiple task to improve performance of individual GP models. The main short-coming approach, as well as complex single-task kernels, is the trade-off of heavy computational load. Multi-task GP scales more quickly with $\mathcal{O}(n^3 \times m^3)$ for m tasks and n observations, as opposed to $m \times \mathcal{O}(n^3)$ if we train each ask independently. In the next chapter, we discuss sparse approximation methods that trade-off some model accuracy for a speed-up in training time.

Chapter 5

Sparse Gaussian Process Approximations

Summary

This chapter explores Sparse Gaussian Process Approximations methods to address the cubic scaling limitation of GPs with large datasets. Generally, we try to avoid inverting and calculating the determinant of the full kernel matrix. There are two ways to do this: First is by model approximation - replace the prior kernel with a smaller one and secondly, by approximate inference - obtaining the full predictive posterior then approximate it during hyperparameter optimisation. We start with the naive Subset of Data method, then progresses to more sophisticated methods such as the Fully Independent Training Conditional approximation and the Variational Free Energy approximation.

5.1 Subset of Data

The Subset of Data (SoD) method is a simple and naive approach to approximate a GP by selecting a smaller subset D_{sod} with size $M < N$ from the full training set

D, as illustrated in Figure 5.1. This method maintains efficient GP inference with reduced computational time, as it only requires calculations on a matrix K_{mm} related to the M selected data points.

Selecting a subset of data means that we completely change the original objective, and a great performance in the approximated model does not directly mean it will perform well in the full GP model. This subset selection can potentially inadequately select the unimportant data points and missing out informative data, for example extreme events. Empirical results from Hayashi et al. [2019] suggests that SoD can give a good predictive mean, yet it is very limited in estimating the prediction's uncertainty - the confidence intervals produced by SoD are notably wide compared to the full model. More advanced approximation methods are developed to calibrate these confidence intervals, which will be discussed next.

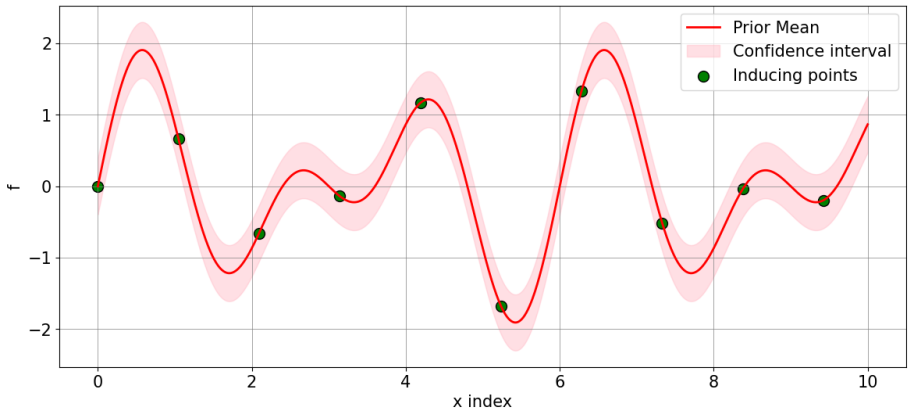


Figure 5.1: This plot shows a GP prior approximation using subset of data points. The green dots, placed at peaks and valleys, act as inducing points to efficiently summarise the dataset for the GP model.

5.2 Fully Independent Training Conditional

The Fully Independent Training Conditional (FITC) approximation [Snelson and Ghahramani, 2006], is a method of using pseudo data points to create an approxi-

mate model to the original GP. Pseudo data means that we use **inducing points**, which essentially are the subset of $M < N$ points initialised from the full dataset (function values u and location Z), to augment the dataset by constructing the joint distribution:

$$p\left(\begin{bmatrix}\mathbf{y}\\ \mathbf{u}\end{bmatrix}\middle| X, Z\right) = \mathcal{N}\left(\begin{bmatrix}\mathbf{y}\\ \mathbf{u}\end{bmatrix}\middle|\begin{bmatrix}0\\ 0\end{bmatrix}, \begin{bmatrix}K_{ff}+\sigma^2 I & K_{fu}\\ K_{fu}^T & K_{uu}\end{bmatrix}\right)$$

where $K_{ff} = K(X, X)$, $K_{fu} = K(X, Z)$ and $K_{uu} = K(Z, Z)$. This joint distribution allows us to derive the marginal likelihood of y by integrating out u:

$$p(\mathbf{y}|X) = \int p(\mathbf{y}, \mathbf{u}|X, Z)\, d\mathbf{u}$$

But we know that using the chain rule:

$$p(\mathbf{y}, \mathbf{u}|X, Z) = p(\mathbf{y}|\mathbf{u}, X, Z)p(\mathbf{u}|Z)$$

And we know that with the conditional Gaussian form of the joint distribution above:

$$p(\mathbf{y} \mid \mathbf{u}, X, Z) = \mathcal{N}\left(\mathbf{y} \mid K_{fu}K_{uu}^{-1}u, K_{ff} - K_{fu}K_{uu}^{-1}K_{fu}^T + \sigma^2\mathbf{I}\right).$$

$$p(\mathbf{u} \mid Z) = \mathcal{N}(\mathbf{u} \mid 0, K_{uu}).$$

The FITC approximation specifically assumes that:

$$\tilde{p}(\mathbf{y} \mid \mathbf{u}, X, Z) = \mathcal{N}(\mathbf{y} \mid K_{fu}K_{uu}^{-1}\mathbf{u}, \Lambda + \sigma^2\mathbf{I}),$$

where $\Lambda = \text{diag}[K_{ff} - K_{fu}K_{uu}^{-1}K_{fu}^T]$. Intuitively, FITC introduces sparsity by allowing the observations in the original dataset to be conditionally independent given the inducing points. We can now marignalise u out and call $Q_{ff} = K_{fu}K_{uu}^{-1}K_{fu}^T$, obtain:

$$\tilde{p}(\mathbf{y}|X) = \mathcal{N}(\mathbf{y}|0, Q_{ff} + \text{diag}[K_{ff} - Q_{ff}] + \sigma^2 I) \tag{5.1}$$

Note that the diagonal term acts as a confidence interval calibrator, where we correct the approximated covariance matrix by subtracting the error between the

original and the approximated matrices. We can parameterise this marginal likelihood to optimise the hyperparameters for the GP prior. Also, we can optionally optimise the location of the inducing points, or strategically initialise them using clustering methods such as K-Means [Bishop, 2006]. Note that the only matrix inversion that we need to do here is on K_{uu}, which is of size $N \times N$. The overall complexity therefore reduces to $\mathcal{O}(n \times m^2)$.

One remark from Bui et al. [2017] is that FITC challenges the philosophy of probabilistic model: as the dataset grows, we want to select a larger set of inducing points to approximate the prior better, which undermines the separation between modeling and inference. Also, similar to SoD, good performance in the approximated objective might not lead to a good performance in the full objective. Such problems motivates an alternative approach, which is to approximate at inference time.

5.3 Variational Free Energy

Variational Free Energy (VFE) [Titsias, 2009] enhances GP training efficiency by approximating the posterior distribution while maintaining the full model. It does this by introducing a simpler distribution $q(f)$ to approximate the posterior, and then minimising the difference between these distributions. We are no longer working with the usual marginal likelihood, but derive a quantity called **Free Energy**, which is the lower bound of the marginal likelihood. By Jensen's inequality:

$$\begin{aligned}
\log p(\mathbf{y}|\boldsymbol{\theta}) &= \log \int p(\mathbf{y}, \mathbf{f}|\boldsymbol{\theta}) d\mathbf{f} \\
&\geq \int q(\mathbf{f}) \log \frac{p(\mathbf{y}, \mathbf{f}|\boldsymbol{\theta})}{q(\mathbf{f})} d\mathbf{f} \\
&= \mathbb{E}_{q(\mathbf{f})} \left[\log \frac{p(\mathbf{y}, \mathbf{f}|\boldsymbol{\theta})}{q(\mathbf{f})} \right] \\
&= \mathcal{F}(q, \boldsymbol{\theta}).
\end{aligned}$$

$\mathcal{F}(q, \boldsymbol{\theta})$ can be expressed as the difference between the log-marginal likelihood of the model and the Kullback-Leibler (KL) divergence from the variational distribution

to the true posterior. Intuitively, KL divergence is the information loss when using one distribution to approximate another. The gernal formula for KL divergence is $\mathrm{KL}[P||Q] = \sum_{x \in X} P(x) \log\left(P(x)/Q(x)\right)$. Free Energy therefore becomes:

$$\mathcal{F}(q, \boldsymbol{\theta}) = \log p(\mathbf{y}|\boldsymbol{\theta}) - \mathrm{KL}[q(\mathbf{f}) \| p(\mathbf{f}|\mathbf{y}, \boldsymbol{\theta})]. \tag{5.2}$$

This bound is maximised when $q(\mathbf{f})$ equals $p(\mathbf{f}|\mathbf{y}, \boldsymbol{\theta})$. However, optimising equation 5.2 does not bring any scaling benefits because we still work fully with the original data. A trick by Titsias is that the function space can be partitioned into pseudo-data and the rest as $\mathbf{f} = \{\mathbf{u}, \mathbf{f}_{\neq \mathbf{u}}\}$, and the approximated posterior is rewritten as:

$$\begin{aligned} q(\mathbf{f}) &= q(\mathbf{u}, \mathbf{f}_{\neq \mathbf{u}}|\boldsymbol{\theta}) \\ &= p(\mathbf{f}_{\neq \mathbf{u}}|\mathbf{y}, \mathbf{u}, \boldsymbol{\theta}) p(\mathbf{u}|\mathbf{y}, \boldsymbol{\theta}) \\ &\approx p(\mathbf{f}_{\neq \mathbf{u}}|\mathbf{u}, \boldsymbol{\theta}) q(\mathbf{u}). \end{aligned} \tag{5.3}$$

Approximation happens because the posterior no longer directly depends on the data $\mathbf{y}$, but only on $\mathbf{u}$, which is of smaller size and summarises the prior for the rest of the function space. This allows for simplification in the Free Energy term:

$$\begin{aligned} \mathcal{F}(q, \boldsymbol{\theta}) &= \mathbb{E}_{q(\mathbf{f}|\boldsymbol{\theta})} \left[\log \frac{p(\mathbf{y}|\boldsymbol{\theta}) p(\mathbf{f}_{\neq \mathbf{u}}|\mathbf{u}, \boldsymbol{\theta}) p(\mathbf{u}|\boldsymbol{\theta})}{p(\mathbf{f}_{\neq \mathbf{u}}|\mathbf{u}, \boldsymbol{\theta}) q(\mathbf{u})} \right] \\ &= \mathbb{E}_{q(\mathbf{f}|\boldsymbol{\theta})} \left[\log \frac{p(\mathbf{y}|\boldsymbol{\theta}) p(\mathbf{u}|\boldsymbol{\theta})}{q(\mathbf{u})} \right] \\ &= \mathbb{E}_{q(\mathbf{f}|\boldsymbol{\theta})} \left[\log p(\mathbf{y}|\mathbf{f}, \boldsymbol{\theta}) \right] - \mathrm{KL}\left[q(\mathbf{u}) \| p(\mathbf{u}|\boldsymbol{\theta}) \right]. \end{aligned} \tag{5.4}$$

We can further assume that the observations are conditionally independent, given the underlying function and hyperparameters:

$$p(\mathbf{y}|\mathbf{f}, \boldsymbol{\theta}) = \prod_{i=1}^{N} p(y_i|f_i, \boldsymbol{\theta}). \tag{5.5}$$

This allows us to arrive at the final form for Free Energy:

$$\mathcal{F}(q, \boldsymbol{\theta}) = \sum_{i=1}^{N} \mathbb{E}_{q(f_i|\boldsymbol{\theta})} \left[\log p(y_i|f_i, \boldsymbol{\theta}) \right] - \mathrm{KL}\left[q(\mathbf{u}) \| p(\mathbf{u}|\boldsymbol{\theta}) \right] \tag{5.6}$$

We can parameterise the location of the inducing points $\mathbf{u}$, for example by restricting them to be Gaussian with mean and covariance parameters [Hensman et al., 2013]. These are called **variational parameters**. Such a restriction conveniently allows for the Free Energy to be compatible with mini-batch optimisation, thereby speeding up the optimisation process. We can therefore optimise:

$$q^*(\mathbf{u}) = \arg\max_{q(\mathbf{u}),\boldsymbol{\theta}} \mathcal{F}(q, \boldsymbol{\theta}) \tag{5.7}$$

This will return the optimal location of the inducing points as well as the optimal set of hyperparameters. In order to obtain the approximated posterior, we plug the result of Equation 5.7 into Equation 5.3 and obtain a Gaussian distribution. The complexity of this approach is the same as FITC, at $\mathcal{O}(n \times m^2)$ where $m < n$ is the size of inducing point set.

It is important to note that there is no need for additional augmentation since the method directly targets the posterior over functions, which includes the inducing variables. The treatment of pseudo-input locations is different from methods like FITC. Here, they are purely variational parameters, not involved in modifying the generative model. This does not interfere with the probabilistic modelling philosophy and provide a coherent way to increase the number of inducing points as training data grow.

5.4 FITC vs VFE: Which one is better?

The objective functions of FITC and VFE share many similarities. In order to obtain them: for FITC, we just take log of the density in equation 5.1 and for VFE, we plug the optimal $q^*(\mathbf{u})$ into equation 5.3. Both can be summarised in the following set up [Bauer et al., 2016]:

$$\text{Loss} = \frac{N}{2}\log(2\pi) + \underbrace{\frac{1}{2}\log|\mathbf{Q}_{ff} + \mathbf{G}|}_{\text{complexity penalty}} + \underbrace{\frac{1}{2}\mathbf{y}^\top(\mathbf{Q}_{ff} + \mathbf{G})^{-1}\mathbf{y}}_{\text{data fit}} + \underbrace{\frac{1}{2\sigma_n^2}\text{tr}(\mathbf{T})}_{\text{trace term}}, \tag{5.8}$$

where:

$$\begin{aligned}\mathbf{G}_{\text{FITC}} &= \text{diag}[\mathbf{K}_{ff} - \mathbf{Q}_{ff}] + \sigma_n^2\mathbf{I} \\ \mathbf{T}_{\text{FITC}} &= 0 \\ \mathbf{G}_{\text{VFE}} &= \sigma_n^2\mathbf{I} \\ \mathbf{T}_{\text{VFE}} &= \mathbf{K}_{ff} - \mathbf{Q}_{ff}.\end{aligned}$$

The common objective function consists of three components: data fit, complexity penalty, and a trace term. From this we can see clearly the difference between the way FITC and VFE calibrate the confidence interval during learning the hyperparamters. For FITC, it does so by adding $\text{diag}[\mathbf{K}_{ff} - \mathbf{Q}_{ff}]$ to the complexity penalty, whereas for VFE, we have an extra trace term. The trace term imposes a regularising effect on the sum of conditional variances of training input conditioned on inducing input.

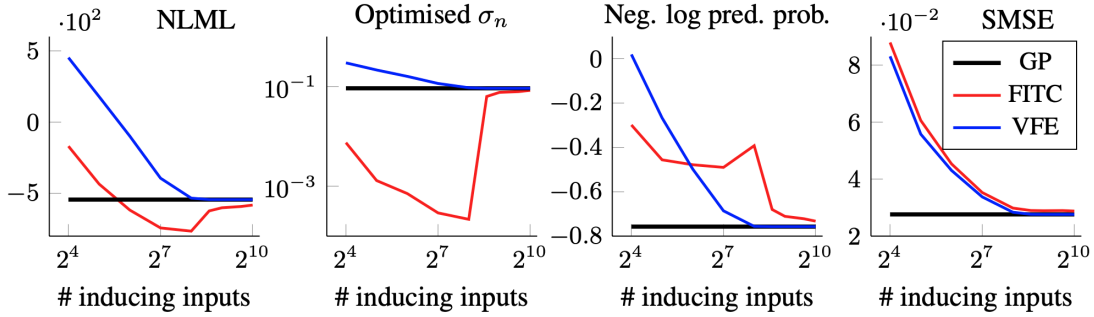


Figure 5.2: Bauer et al. compare the optimisation performance of VFE and FITC with a varying number of inducing inputs to the full Gaussian Process (GP). This includes analysing the objective function (negative log marginal likelihood), optimised noise level, negative log predictive probability, and standardised mean squared error.

Also, Bauer et al.'s empirical analysis suggests that VFE demonstrates stronger theoretical guarantees compared to FITC, for example VFE improves reliably across many aspects with more inducing point while FITC might not necessarily. This is illustrated in Figure 5.2. By rule of thumb, VFE yields more better mean prediction,

while FITC provides better error bars. It is also noted, however, that VFE is prone to giving under-fitting solutions, which is a common problem for variational methods. Yet it is concluded that this is rather an opimisation problem and can be resolved with careful initialisations, for example with clustering techniques such as K-Means or initialising with the solution of FITC.

5.5 VFE in Non-Stationary Gaussian Processes

It is not possible to directly use VFE to approximate the posterior of non-stationary and hierarchical kernel designs such as the GSM kernel from Section 3.10. The Free Energy function for the multi-latent GP is defined as:

$$\begin{aligned}\mathcal{F}(q) = \int \log p(\mathbf{y}|\mathbf{w}, \boldsymbol{\ell}, \boldsymbol{\mu}) q(\mathbf{w}, \boldsymbol{\ell}, \boldsymbol{\mu}) d\mathbf{w} d\boldsymbol{\ell} d\boldsymbol{\mu} \\ - KL[q(\mathbf{u}_w, \mathbf{u}_\ell, \mathbf{u}_\mu)||p(\mathbf{u}_w)p(\mathbf{u}_\ell)p(\mathbf{u}_\mu)].\end{aligned} \tag{5.9}$$

The likelihood in this case, which lies in the intergral above, becomes multi-latent and cannot be factorised in the way that is mentioned in Equation 5.5. This prevents the overall Free Energy loss from being compatible with stochastic optimisation, leading to no computational reduction if we were to evaluate this integral with methods such as Monte Carlo and quadrature. We can say that this integral remains intractable.

A proposed solution to this problem by Paun et al. [2023] is by using "trajectory segmentation". By assuming that the data can be divided into L segments and imposing independence among these segments, the approximate likelihood can be expressed as the product of the likelihoods of the individual segments:

$$p(\mathbf{y}|\mathbf{w}, \boldsymbol{\ell}, \boldsymbol{\mu}) = \prod_{i=1}^{L} p(\mathbf{y}_i|\mathbf{w}_i, \boldsymbol{\ell}_i, \boldsymbol{\mu}_i).$$

Consequently, the intractable integral transforms into a summation of expectations:

$$\sum_{i=1}^{L} \mathbb{E}_{q(\mathbf{w}_i, \boldsymbol{\ell}_i, \boldsymbol{\mu}_i)}[\log p(\mathbf{y}_i|\mathbf{w}_i, \boldsymbol{\ell}_i, \boldsymbol{\mu}_i)].$$

whereby Monte Carlo sampling can be used to calculate each of the L low-dimensional expectation. This approach, in summary, adds another layer of approximation on top of VFE to accelerate inference and learning speed.

5.6 Discussion

This chapter discussed into two prominent methods, FITC and VFE, aimed at overcoming the computational challenges associated with GP models for large datasets. By introducing pseudo-data, both methods effectively reduce the computational complexity from cubic to quadratic relative to the number of inducing points, thus enabling the application of GP models to larger datasets. While FITC focuses on modifying the GP model, VFE approximates the posterior distribution directly, offering a more theoretically grounded approach with better scalability and potential accuracy improvements. In the experiment work, we will conduct comparative analysis on the performance of FITC and VFE on the large wind turbine dataset.

Chapter 6

Experiments and Contributions

Summary

In this chapter, we will implement the methods discussed in the previous 4 chapters on the SCADA dataset and benchmark them against the baseline RBF GP model. We firstly investigate the covariates and correlation structure within the SCADA dataset to select important features, then clarify the processing done before modeling. Next, we compare the Spectral Mixture kernel, the Generalised Spectral Mixture kernel and the multi-task GP against the baseline model, then we finally experiment with the approximation methods.

6.1 Exploratory Data Analysis

6.1.1 SCADA Data Overview

The SCADA dataset originally comes with more than 100 input covariates and over 1 million observations across six wind turbines between 2017 and 2021. The target of each observation is the Power Output, measured kW. The input covariates cover operational metrics such as wind speed, temperature, acceleration, angle of many

components. Figure 6.1 illustrates the type of components that these covariates may cover. Each observation has a time-stamp and is recorded in 10-minute intervals. Following the previous work of my supervisors, we are able to focus on the most relevant 32 input covariates.

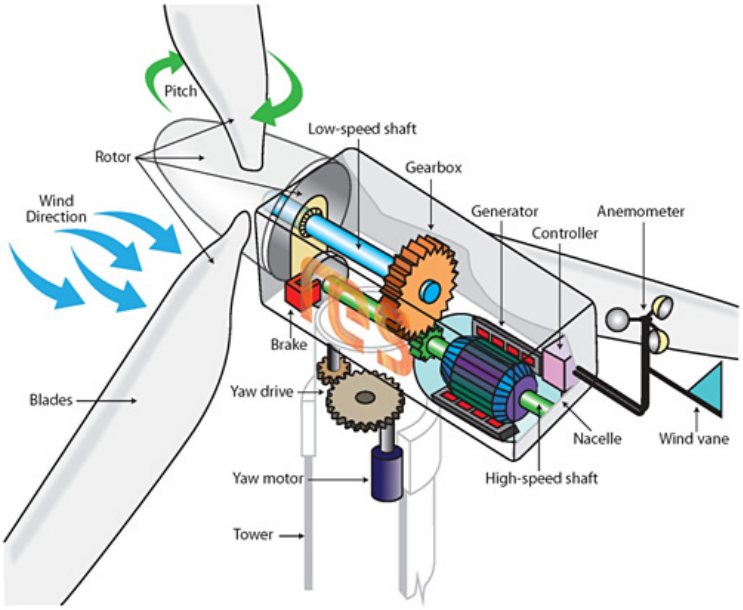


Figure 6.1: Illustration by [Beaufort Court, 2017] for the operational variables within a wind turbine that a SCADA system can record.

6.1.2 Feature Selection

We aim to select the number of parameters that strikes a balance between maximum computational capacity available and optimising model performance. Through our experiments with Single-Task GP with exact inference, the maximum number of parameters was 5 and for multi-task GP, the number was 3. Therefore, we need to select the most important 5 features that represent the overall dataset and take a subset of that for multi-task GP. In the wind turbine monitoring literature, such as in Rogers et al. [2019], Wind Speed is always the most important covariate with strong correlation with Power Output, thus will be included. For the other covariates, we

aim to select those with strong correlation with Power Output yet reasonable multicolinearity with other variables, as well as well-represent the component category. The selected variables are Nacelle Temperature, CPU Temperature, Yaw Drive Acceleration and Power Acceleration. It can be also noted an important information from the blades are embedded in the Wind Speed variable, so they do not need to be included separately. The selected variables can be visualised in Figure 6.2.

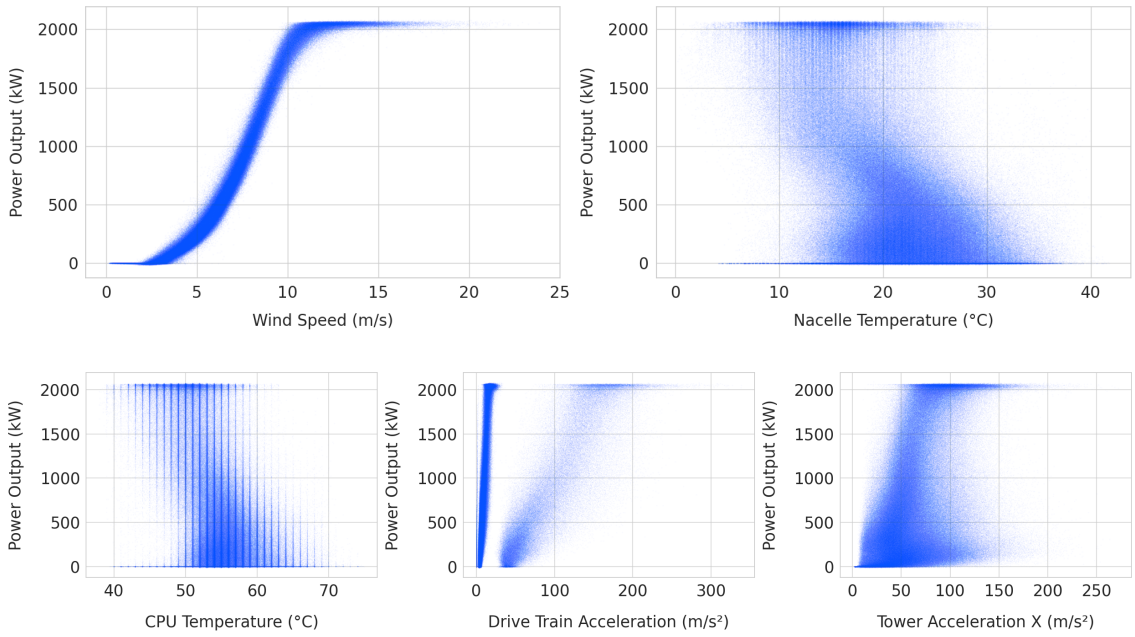


Figure 6.2: Scatter plots of Power Output against 5 selected input covariates

6.1.3 Further Data Preprocessing

To prepare our dataset for modeling, we carry out three preprocessing steps. First, we identify and remove any negative values of Power Output, considering them outliers. Next, we standardise the dataset for all six wind turbines to [-1,1] range to avoid numerical overflow when optimising the GP prior. This involves calculating and storing the mean and standard deviation for each turbine's data, which will be essential for accurately interpreting predictions later on. Lastly, to apply the MTGP

model effectively, we synchronise the timestamps across all six turbines since the original dataset had mismatched timestamps. This results in a dataset dedicated solely for MTGP and its comparison with a baseline model. For other analyses not involving MTGP, we use the original, unsynchronised dataset, as direct comparisons with the MTGP findings would not be appropriate.

6.2 Single-Task Kernel Comparison

6.2.1 Spectral Mixture Kernel

This section will analyse and compare the performance between the Spectral Mixture (SM) kernel and the baseline RBF kernel. We will compare them with three different dataset sizes from the 100,000 observations available for Turbine 1:

Table 6.1: Training and Testing Set Sizes for Different Scenarios

Scenario	Training Size	Testing Size
1	1,000	2,000
2	5,000	10,000
3	7,000	10,000

Each of these will include the 5 important input covariates that are obtained from the previous section. The purpose of choosing such irregularly large test set is that we want to stretch the extrapolation capability of the SM kernel that was claimed in theory. We will conduct hyperparameter optimisation for 10 initialisations for each model in each dataset and record the RMSE and NLPD on the test set. The configuration of each model is presented in Table 6.2:

Table 6.2: GP Prior Configuration

	RBF	SM
ARD	Yes	Yes
No. of Mixtures	No	3
No. of Parameters	7	34

The comparison in RMSE over all 10 initialisations can be presented in Figure 6.3. For the train size of 1,000, the RBF kernel generally performs worse than the SM kernel, with more variability in its RMSE. As the training size increases to 5,000 and 7,000, the RBF kernel's performance becomes more stable, but still remains worse in RMSE compared to the SM kernel. The SM kernel consistently exhibits lower RMSE across all initialisations and training sizes, indicating it has better performance in predicting point estimates with the predictive posterior. It can also noted that the larger the training data set, the better the performance of both kernels, yet the SM kernel experience a larger improvement overall. The result is similar when comparing the NLPD between the the two kernels on three datasets, given by Figure 6.4. This means that even when accounting for the uncertainty estimation, the SM kernel provide better performance than the RBF kernel.

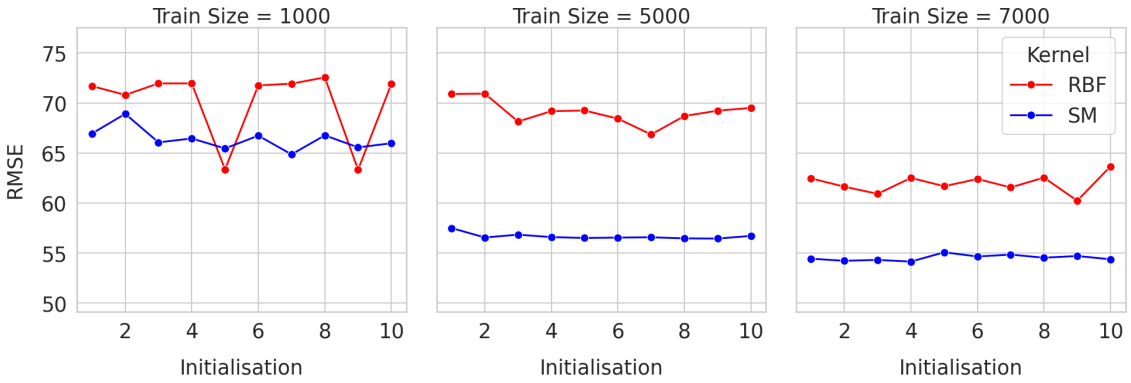


Figure 6.3: Comparative RMSE performance of RBF and SM kernels over varying 3 training set sizes of Turbine 1 and 10 initialisations

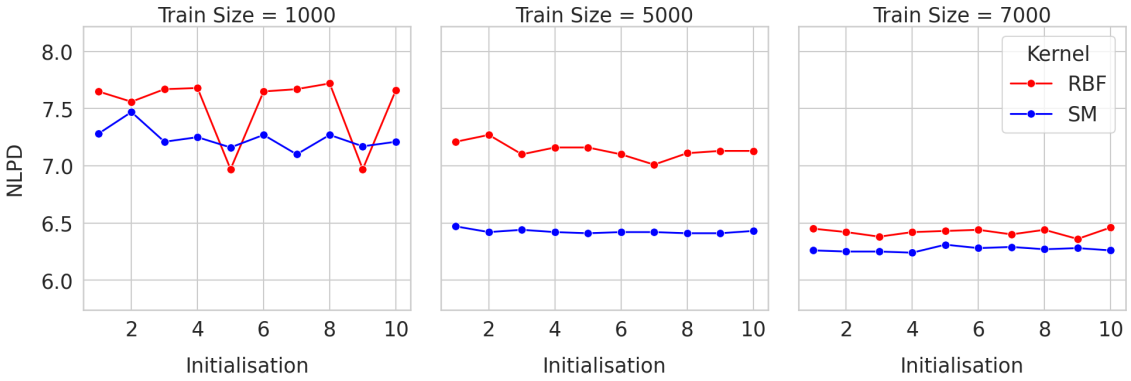


Figure 6.4: Comparative NLPD performance of RBF and SM kernels over 3 training set sizes of Turbine 1 and 10 initialisations

We can also take the average of the metrics across 10 initalisations and compare, as presented in Table 6.3. The SM kernel consistently exhibits lower RMSE and NNLPD averages than the RBF kernel, across all sample sizes. This indicates a higher predictive accuracy and a better fit to the data when using the SM kernel. The improvements are also strongly on average as the training size increases, suggesting that the SM kernel may efficiently leverage larger datasets to enhance model performance.

Table 6.3: Average RMSE and NLPD for RBF and SM kernels

Training Size	**RBF**		**SM**	
	RMSE	**NLPD**	**RMSE**	**NLPD**
1,000	70.12	7.52	**66.37**	**7.24**
5,000	69.11	7.14	**56.68**	**6.43**
7,000	61.96	6.42	**54.54**	**6.27**

We can also test the two SM and RBF models that were trained on 7,000 observations on multiple different time horizons to see how they perform in short, medium and long term. This is illustrated in Figure 6.5. For short-term predictions, as indicated by the 1-day ahead forecast, both the RBF and SM kernels exhibit

comparable performance with the SM kernel having a slight edge. As the forecasting horizon extends to mid-term, indicated by the 7 to 28-day predictions, the SM kernel consistently demonstrates better performance with lower RMSE and NLPD values compared to the RBF kernel. In the long-term prediction range, from 35 days ahead and beyond, the SM kernel maintains its lead in predictive accuracy. Overall, we can see an deterioration in performance in the two GP models as we move further away from the training set, which aligns with the theory of GP. However, the rate of that in the SM kernel is significantly lower than the RBF kernel, meaning it is more reliable for long-term prediction. This also aligns with the conclusions of Wilson and Adams [2013] in the original paper.

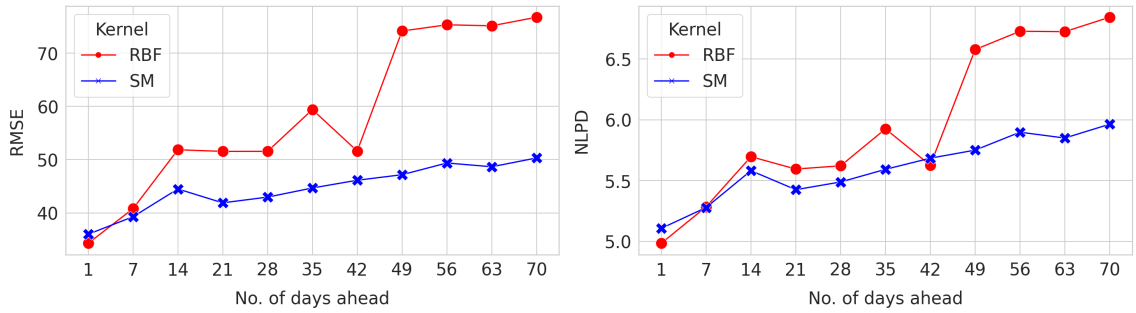


Figure 6.5: Comparative RMSE and NLPD performance of RBF and SM kernels between 1 and 70 day-ahead predictions for Turbine 1

Finally, we can assess the Power Output time series at the 35-day ahead prediction where the difference in performance would be the clearest based on Figure 6.5. We will have the general view of the entire prediction first, roughly 5,000 data points ahead, then we will zoon in the beginning and ending sections to see the difference in mean prediction and the confidence interval.

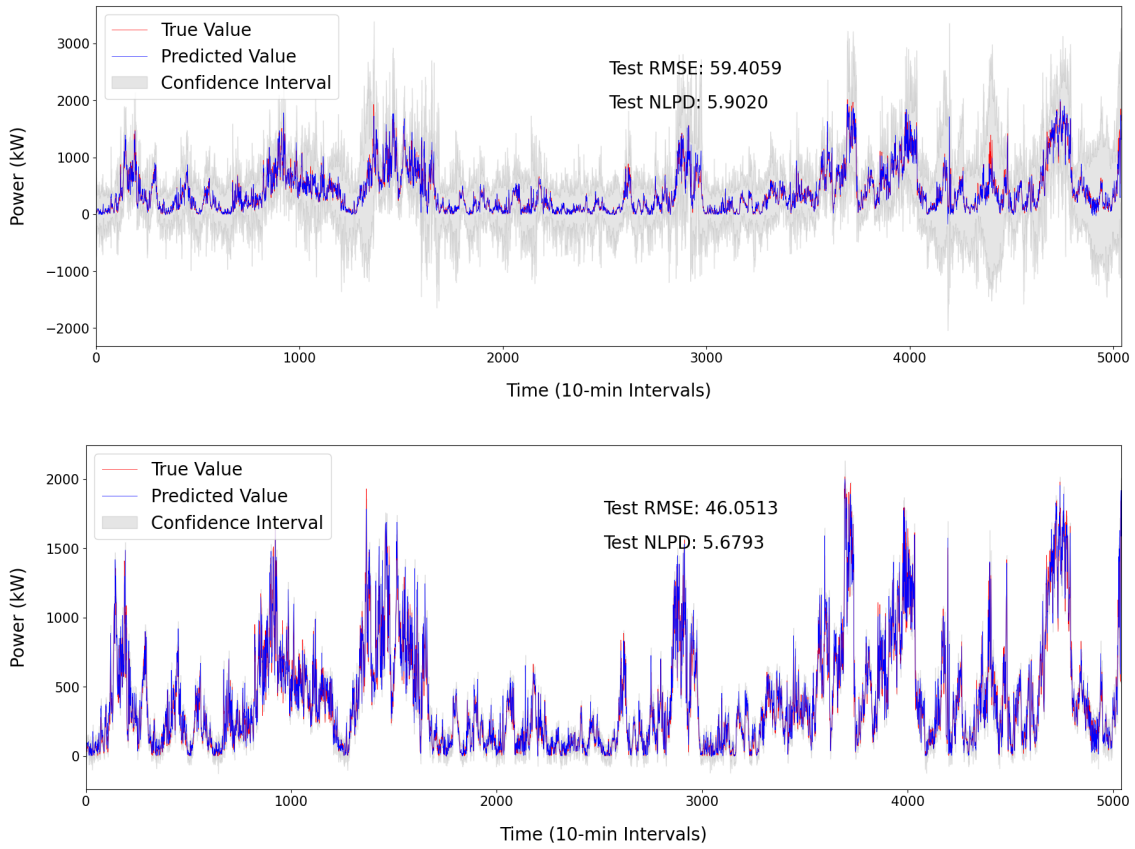


Figure 6.6: **Top**: A 35-day forecast with 95% Confidence Interval using the RBF kernel. **Bottom**: A 35-day forecast with 95% Confidence Interval using the SM kernel. Both are tested on Turbine 1.

From Figure 6.6, it is notable that the SM kernel produce an overall better mean prediction and shows tighter confidence intervals, whereas the RBF kernel produce relatively good prediction but wider confidence interval. These visual observations align with the RMSE and NLPD metric, where the SM kernel outperforms the RBF kernel. One remark of the prediction is that sometimes the GP using both kernels produce negative values of confidence intervals, which is invalid. Further work is needed on this, which will be discussed in Chapter 7.

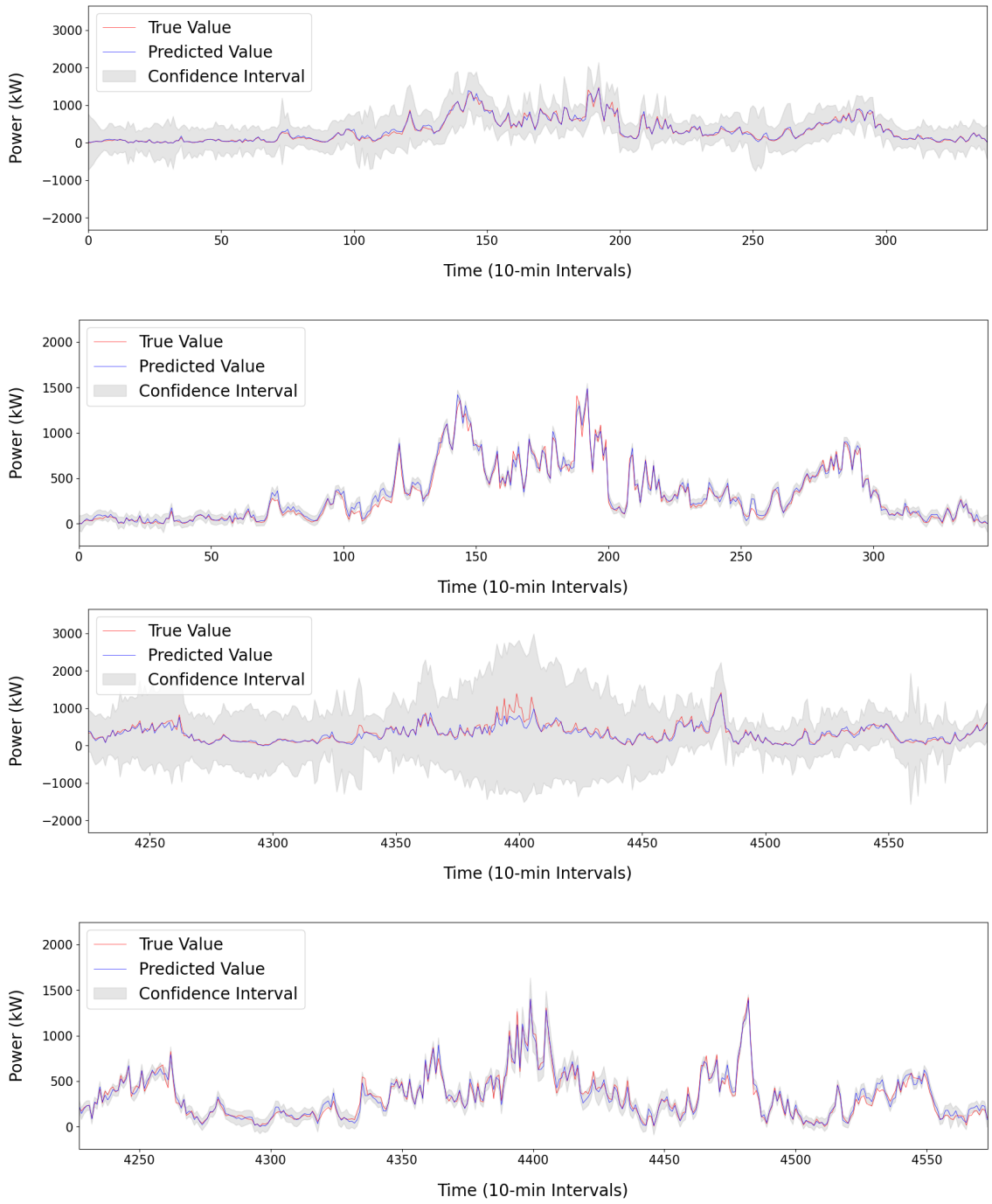


Figure 6.7: Kernel Mean Prediction with 95% Confidence Interval Comparison. Top to bottom: (1) Start-of-period prediction using RBF kernel, (2) Start-of-period prediction using SM kernel, (3) End-of-period prediction using RBF kernel, (4) End-of-period prediction using SM kernel. Start-of-period predictions are closer to training data, whereas end-of-period predictions are farther from training data - Turbine 1.

The performance pattern is consistent when we zoom at at the start and end of the forecast period, illustrated in Figure 6.7. The confidence interval produced by the RBF kernel is generally wider, and both the mean prediction and the overall accuracy significantly decline as we move further from the training data. In contrast, the performance under the SM kernel only experiences a minor deterioration.

In summary, we can claim that with a dataset with 5 covariates, the SM kernel perform similarly to the RBF for short-term prediction, but will outperform the RBF for long range extrapolation and produce more confident predictions overall.

6.2.2 Non-Stationary Spectral Kernel

This section compares the performance of the Generalised Spectral Mixture (GSM) kernel with the SM and the baseline RBF kernels. We will select the the training size similar to Table 6.1. However, due to limits in our implementation and hardware, we adjusted the comparison to focus on one input variable, Wind Speed, instead of five. For similar reasons, the GSM model uses two mixtures, unlike the SM model, which uses three. The settings for the three models are detailed in a Table 6.4.

Table 6.4: GP Prior Configuration

	RBF	**SM**	**GSM**
No. of Mixtures	No	3	2
No. of Parameters	3	10	13

We can first compare the RMSE performance between the RBF, SM and GSM kernels on 3 different training set sizes. The GSM kernel perform the best with the smallest data set, achieving the lowest RMSE at a train size of 1,000. As the training size grows, the RBF and SM kernels display very similar performances in RMSE values, particularly in the 5,000 training size, and the GSM is worse than these two. At the largest train size of 7,000, the GSM's RMSE advantage diminishes, and it generally falls behind the RBF and SM kernel. It is notable that when we have

only 1 covariate in this context, the SM kernel tend not to outperform the RBF kernel.

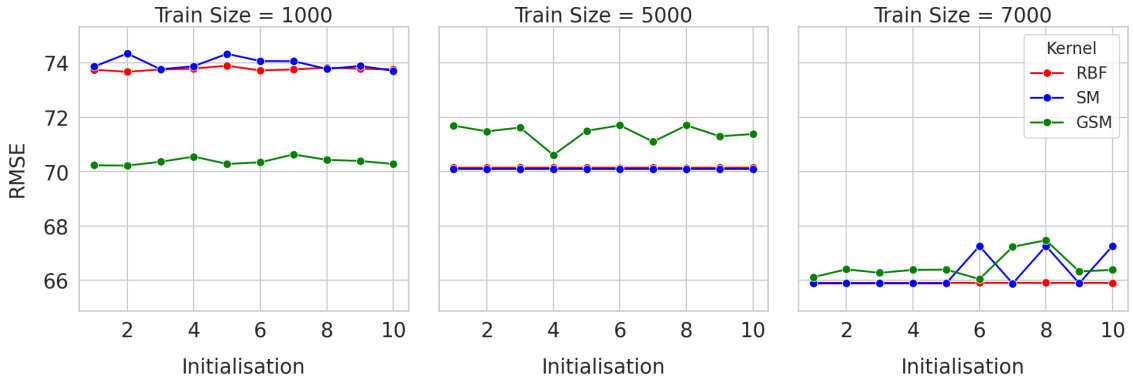


Figure 6.8: Comparative RMSE performance of GSM, SM and RBF kernels over 3 training set sizes of Turbine 1 an 10 initialisations

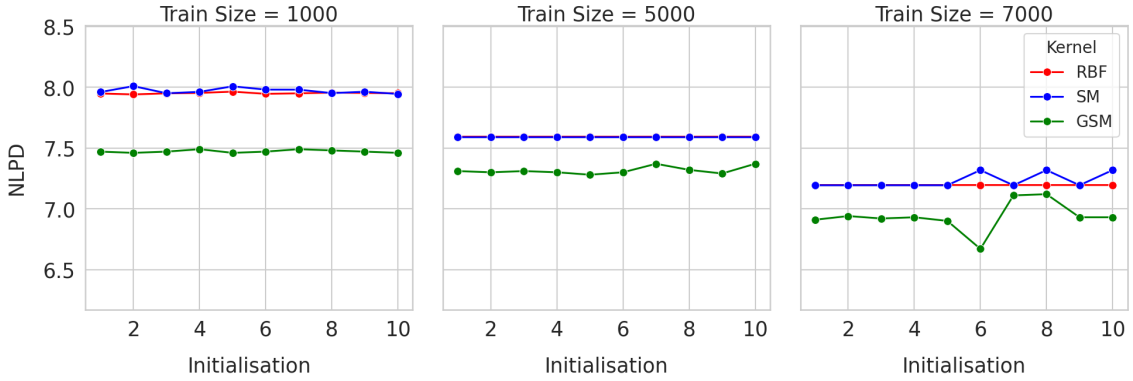


Figure 6.9: Comparative NLPD performance of GSM, SM and RBF kernels over 3 training set sizes of Turbine 1 an 10 initialisations

A different pattern can be seen in the NLPD performance. The GSM kernel consistently outperforms RBF and SM kernels in all training scenarios, demonstrating superior prediction accuracy through the lowest NLPD values across different training sizes. While both RBF and SM kernels show slight improvements in NLPD as the training size increases, indicating better predictive performance, they remain similar

to each other without a significant distinction. Even with larger training sizes, the GSM kernel maintains its advantage, meaning it is better when we acocunt for both mean prediction and the confidence interval.

When we consider the average metric across 10 initialisations: the GSM kernel consistently outperforms RBF and SM in terms of NLPD, meaning greater confidence in the predictive distributions it generates. However, while RBF and SM occasionally yield better RMSE scores in some cases, the GSM kernel demonstrates the strongest overall performance, particularly with the smallest training set where it leads in both RMSE and NLPD.

Table 6.5: Average RMSE and NLPD for RBF, SM, and GSM kernels.

Training Size	**RBF**		**SM**		**GSM**	
	RMSE	**NLPD**	**RMSE**	**NLPD**	**RMSE**	**NLPD**
1,000	73.77	7.95	73.97	7.95	**70.37**	**7.47**
5,000	70.14	7.59	**70.10**	7.59	71.41	**7.32**
7,000	**65.90**	7.20	66.30	7.23	66.50	**6.94**

In terms of comparing performance in multiple time horizons, for RMSE, the three kernel functions—RBF, SM, and GSM—display closely aligned results, with minimal variation across the forecasting period. However, when evaluating NLPD, the GSM kernel clearly demonstrates superior performance, indicating it provides more reliable uncertainty estimates in GP regression than the RBF and SM kernels.

In summary, given the experiment in this section, we can claim that the GSM kernel generally provides better NLPD metrics yet similar or worse RMSE performance compared to the SM and RBF kernels. However, there are some contradictions in the results of this section: the SM kernel sometimes does not outperform the RBF kernel. This is likely to stem from the fact that we are only able to use one covariate instead of five. There are two reasons why five covariates, or even more than one, was not possible when this experiment was conducted: (1) The implementation for the GSM kernel was developed on the code base the Gibb's kernel, which was optimised

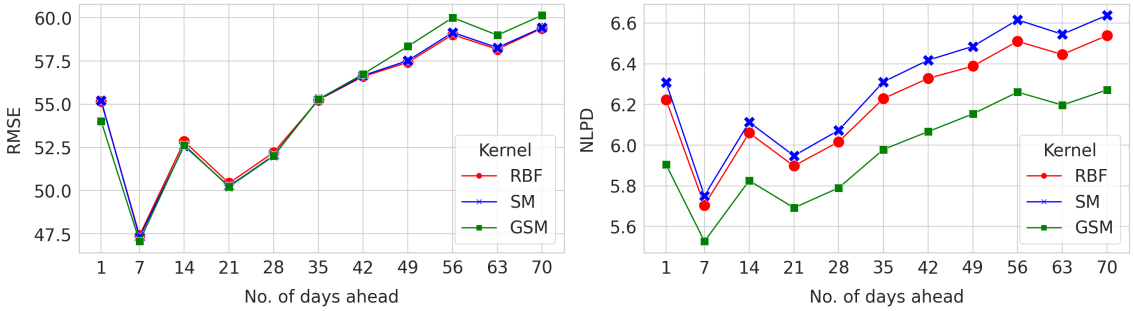


Figure 6.10: Comparative NLPD and RMSE performance of RBF, SM and GSM kernels over increasing time horizons of predictions for Turbine 1

for only one input covariate. Re-optimising the utility functions for numerical instability management for multiple input covariates is challenging and extends beyond the current project's scope (2) The project was limited by computational resources, which restricted the feasibility of processing higher dimensions. We will look more into these issues and what they mean for our study in Chapter 7, which will help guide the next steps for future work.

6.3 Multi-Task Kernel Comparison

6.3.1 Model Choice and Setting

In this section, we will compare a MTGP model, where we make predictions for six different wind turbines at once, with using six independent GP with RBF kernel as previous baseline models. The specific MTGP model that we use is a simple case of the Intrinsic Model of Coregionalisation, where the coregionalisation matrix will have the following form with $\boldsymbol{\theta}$ being a length 6 vector and $\mathbf{v}$ being an 6×6 parameter matrix to model variance:

$$\mathrm{B} = (\boldsymbol{\theta}\boldsymbol{\theta}^T + \mathrm{diag}(\mathbf{v}))$$

and the MTGP kernel as the following form using Hadamard product:

$$K_{ICM} = B \odot K_{RBF}$$

The dataset used will be the synchronised version of the SCADA dataset, where we have 18,000 observations for train set and 6,000 observations for test set with 3 covariates across six turbines. The training process involves two models in Table 6.6:

Table 6.6: GP Prior Configuration

	MTGP	**6 × RBF**
ARD	Yes	Yes
No. of Parameters	72	5 each

6.3.2 Results

The comparative result is presented in Table 6.7. Training is carried out over 10 iterations and the average results is presented. Overall, we can see that using MTGP improves in RMSE only in Turbine 6, and in NLPD in Turbine 2, Turbine 3 and Turbine 4.

Table 6.7: Average RMSE and NLPD for MTGP and Independent GPs by Turbine

Turbine	**Average RMSE**		**Average NLPD**	
	MTGP	**Independent**	**MTGP**	**Independent**
1	66.17	**64.18**	6.85	**6.83**
2	60.71	**58.14**	**6.21**	6.37
3	62.73	**61.04**	**6.58**	6.59
4	58.63	**57.01**	**6.26**	6.29
5	59.91	**58.28**	6.57	**6.38**
6	**55.35**	56.02	6.36	**6.22**

We can go further and pick out the GP models with the best improvement in RMSE and most reduction in RMSE to compare how they compare over multiple time horizon prediction period. Turbine 6 received the best improvement while Turbine 1 received the worst reduction in performance.

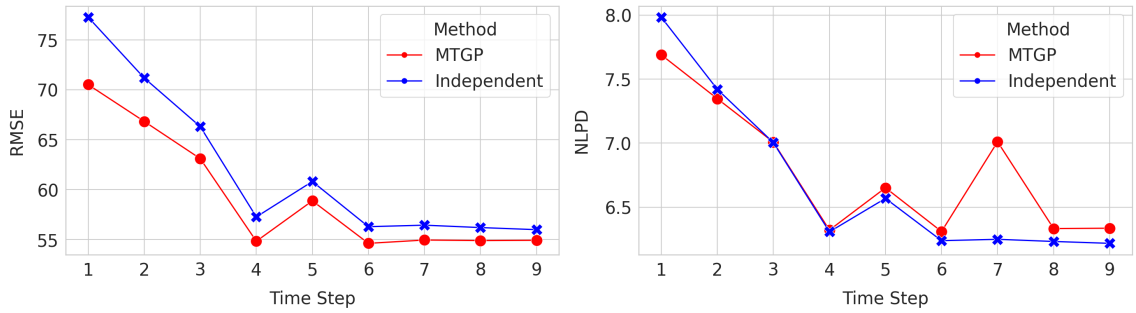


Figure 6.11: Comparative RMSE and NLPD performance of MTGP and Independently-trained GP model over increasing time horizons of predictions (9 uniform intervals in 7 days) for Turbine 6

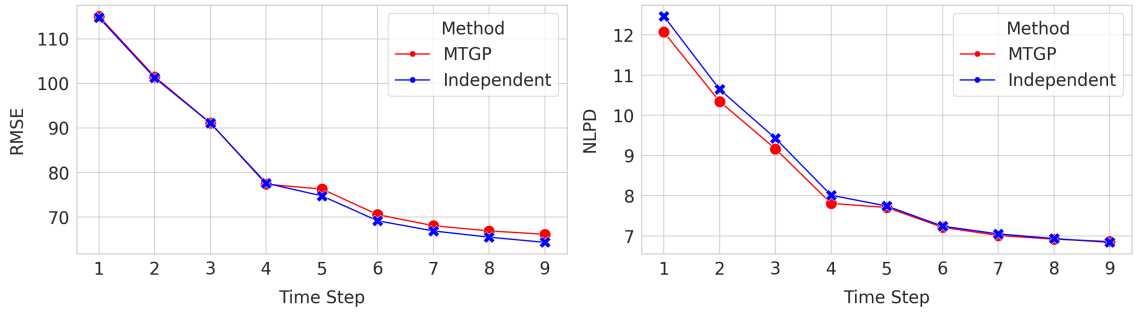


Figure 6.12: Comparative RMSE and NLPD performance of MTGP and Independently-trained GP model over increasing time horizons of predictions (9 uniform intervals in 7 days) for Turbine 1

In figure 6.11, it can be noted that in short term prediction, within 1 week, the prediction performance increases as we predict more days ahead. The MTGP (focusing on Turbine 6) can be seen to improve in RMSE more quickly than when we

train the GP independently. The improvement in NLPD appears to be inconsistent, and tends to get worse after the 4th interval. Meanwhile, in figure 6.12 and for a similar prediction period, we can see that the MTGP and Independently-trained GP models perform quite similarly, even though the MTGP is worse at RMSE after interval 4 and better at NLPD before interval 4. The overall metrics of models trained for Turbine 1 is high, proving that the synchronised dataset might have been more challenging than other turbines.

Overall, it can be claimed that the MTGP improves inference performance in one wind turbine in the short term. Yet the results are inconsistently different in other wind turbines. This could pontentially be hindered by the choice of the coregionalisation method to be ICM. Future work can experiment with more generalised methods such as the LCM, for example, we can divide 6 turbines into 3 groups and each group can be modelled by a separate kernel function.

6.4 VFE and FITC Comparison

6.4.1 Comparison Method

In this section, we will compare the two sparse approximation methods FITC and VFE. The goal is to see how close the RMSE and NLPD performance of each approximated GP are compared to the full GP model, as well as how much training time is accelerated by. We do not compare the absolute performance of each approximated GP both much better or worse performance relative to the full GP model mean unreliability in the approximation method.

Regarding configuration, the full GP model to be approximated will use the SM kernel in the previous section, with 5 input covariates and 7,000 observations for Turbine 1, with ARD activated. We use 500 inducing points with 2 initialisation methods. The first way is to uniformly select the inducing points from the 7,000 observations, i.e. 1 every 14, and the second way is with K-Means clustering, where we select 500 centroids after training the clustering model. The reason for this is to

assess the initialisation problems that VFE is expected to have in Bauer et al. [2016]. All training will be initialised 10 times and the average metric will be taken.

6.4.2 Results

The result is presented in Table 6.8. Two observations from the results: Firstly, VFE is indeed sensitive to initialisation method, while FITC is not. The performance of VFE using uniform initialisation caused the model to under-fit and perform poorly in the test set, yet when carefully initialised with K-Means clustering, the performance is enhanced significantly. Overall, it is notable that the performance with VFE approximation is closer to the full GP model compared to FITC, especially in RMSE performance. The difference is not as noticeable in the NLPD metric.

Table 6.8: Comparison of Average RMSE and NLPD between Uniform and K-Means initialisation for VFE and FITC approximations

	Average RMSE		**Average NLPD**	
	Uniform	**K-Means**	**Uniform**	**K-Means**
FITC GP	54.20	54.20	6.24	6.24
VFE GP	57.19	**54.47**	6.47	**6.26**
Full GP	54.54		6.27	

We can also compare how these models perform in extrapolating to multiple time horizons. From Figure 6.13, it is notable that the performance of VFE approximated model with K-Means initialised inducing points recovers the performance a lot better across all time periods.

Meanwhile, initialisation method does not make a strong difference in performance of the FITC approximation method. The FITC method performs well, similarly to the VFE, even though there is a noticeably worse performance in short term prediction, in 1 day period.

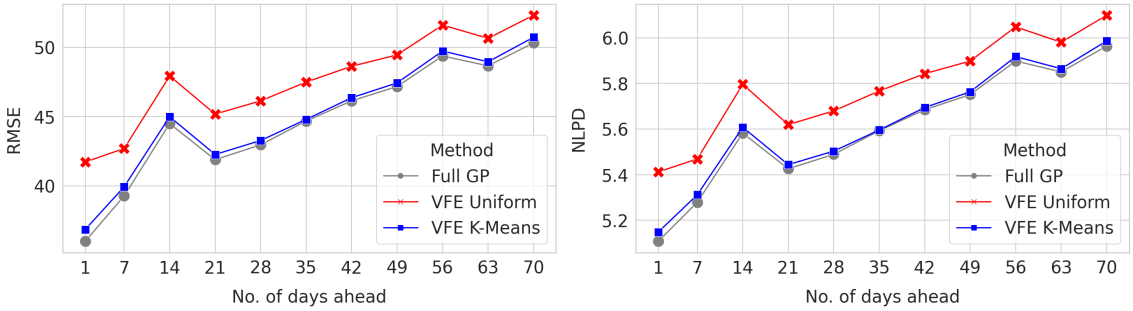


Figure 6.13: Comparative RMSE and NLPD performance of VFE approximated GP compared to the exact GP increasing time horizons of predictions for Turbine 1

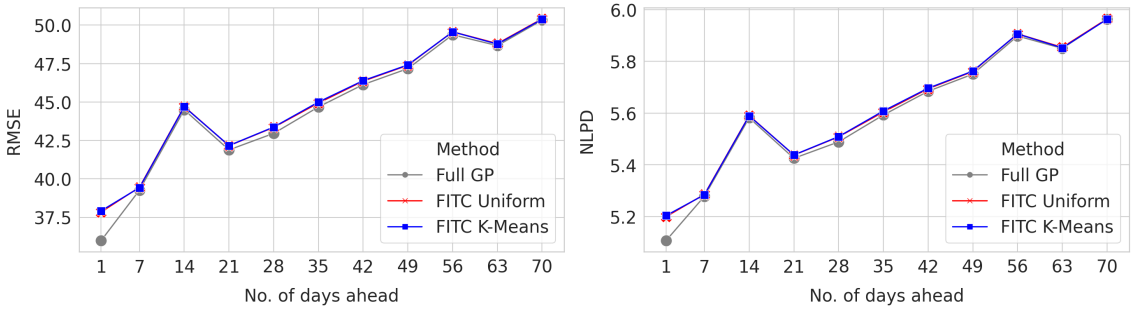


Figure 6.14: Comparative RMSE and NLPD performance of FITC approximated GP compared to the full GP increasing time horizons of predictions for Turbine 1

Overall, we can claim that the results in this section strongly aligns with the observations of Bauer et al. [2016]. The VFE method performs better with careful initialisation, and it is also more reliable in both short term and long extrapolation when it comes to recovering the full GP model. Without careful initialisation, VFE overestimate both the posterior mean and the confidence interval in the context of wind power inference. Meanwhile, the FITC is not sensitive to initialisation method, but it is slightly less reliable when compared to VFE, particularly in short term prediction.

6.4.3 Time Improvement

By using the sparse approximations methods, the hyperparameter optimisation time (1,000 iterations/intialisation) is reduced by roughly 10 times, given in Table 6.9.

Table 6.9: Training Time Comparison

	Full GP	**VFE**	**FITC**
Training Time	11.5 hours	1.5 hour	1 hour

Overall, we can claim that in the context of wind turbine monitoring, VFE approximation perform slightly better and more reliable than FITC given that we give it careful initalisation of inducing point. However, this difference as not been significant on average, potentially because the overall dataset is not large enough. Also, the reason why VFE is 0.5 hour slower than FITC is because it involves optimising the inducing point location on top of the SM kernel hyperparameters, thus having more parameters overall. We did not optimise these locaiton in FITC, even though we theoretically can, is because there is no implementation out-of-the-box to do so. This can be left for future work so that the comparison between the two methods can be fairer.

Chapter 7

Conclusions

7.1 How to predict wind power now?

In this project, we evaluated various tools within GP literature to improve wind power prediction. Our analysis, grounded in theory and experiments using the SCADA dataset, led to four insights. Firstly, there is strong evidence to show that the SM kernel outperforms the RBF kernel across multiple prediction time horizons, especially with five input covariates and ARD enabled. Secondly, the GSM kernel is good at providing accurate uncertainty estimates, though its mean prediction accuracy does not convincingly surpass that of SM and RBF kernels. Thirdly, MTGP enhances performance in one four out of six turbines across two metrics, with slight improvements. Lastly, VFE is found to be more reliable than FITC for GP model approximation given good inducing point initialisation, and both of which offer up to a tenfold reduction in training time. These findings encourage the adoption of sophisticated kernels like SM or MTGP, integrated with VFE, to achieve superior performance over the conventional RBF kernel in future GP applications for wind turbine monitoring.

7.2 Limitations and Future Work

The analysis of this project is of course not entirely exhaustive. There are three important shortcomings to improve on:

- The prior of all the GP defined in this work can take both positive and negative values, leading to forecasts that often include negative values for wind power, which are invalid. A potential solution is to apply a log-transformation to the prior and then convert back to the original scale using the log-normal distribution formula for predictions.
- The GSM kernel is implemented based on the Gibb's kernel code base, which lacks optimised numerical stability management methods for multi-covariate applications. This limitation prevents the use of more than one covariate when benchmarking the GSM. Updating the utility functions in the sophisticated code base is an extensive task that exceeds the project's timeframe.
- The final experiment does not optimise the location of inducing points in the FITC method. The VFE approach has been the standard, and there is no available code base for optimising inducing point locations. This aspect is recommended for future research if FITC is still of interest in the wind turbine monitoring literature.

Further work can also involve testing the spectral and non-stationary extensions for MTGP such as Altamirano and Tobar [2022] to see it outperforms all the current methods that we studied. There are also extension of VFE that can be applied to GSM and MTGP, such as in Paun et al. [2023] and Zhao and Sun [2016] that will speed up these complex kernels.

Appendix A

Programming details and declarations

This project's code is published under **this link to Google Drive folder**, which ensures anonymity. The software and packages mentioned below have made significant contributions to the development of Chapter 6:

1. **climate-kernel-learning** by Lalchand et al. [2023]: This code base provides a fundamental framework for a hierarchical Gibb's kernel. I expanded it into the full GSM kernel with two and three mixtures. The implementation include registering parameters, modifying the parameter constraints, rewrite the kernel composition to that of Remes et al. [2017] and update the inference and parameter whitening code.

2. **GPyTorch** by Gardner et al. [2018]: Deployed for learning and inference of RBF, SM and MTGP kernels. It is also used for VFE and FITC approximation, as well as for tensor manipulation and optimisation routines such as using ADAM.

3. **Python Programming Language** and commonly used packages including Matplotlib, NumPy, and Pandas.

The training of GP models are quite computational heavy, with the highest computing memory needed being roughly 50GB of RAM. In order to reproduce the

results, the user would need to load the correct datat set specified in each Python Notebook and the experiment details in Chapter 6. They also need to load the corresponding model using the guidance in the notebook. The detailed recordings of the metrics are also presented in the Spreadsheets files in the Google Drive folder.

Note that some of the results presented in the report may not be precisely reproduced when running the scripts, as they were generated through multiple executions with different initialisation. However, they should be highly identical.

Bibliography

Altamirano, M. and Tobar, F. [2022], Nonstationary multi-output gaussian processes via harmonizable spectral mixtures, *in* 'Proceedings of Machine Learning Research', Vol. 151, PMLR.

Alvarez, M., Rossaco, L. and Lawrence, N. [2012], 'Kernels for vector-valued functions: A review', *Foundations and Trends in Machine Learning* .

Bauer, M. S., van der Wilk, M. and Rasmussen, C. E. [2016], Understanding probabilistic sparse gaussian process approximations, *in* 'Advances in Neural Information Processing Systems'.

Beaufort Court [2017], 'Renewable energy sources: Wind'. Article URL.

Bishop, C. M. [2006], *Pattern Recogniton and Machine Learning*, Springer.

Bochner, S. [1959], *Lectures on Fourier Integrals.(AM- 42)*, Vol. 42, Princeton University Press.

Bonilla, E. V., Kian, C. and Williams, C. [2007], Multi-task gaussian process prediction, *in* 'Advances in Neural Information Processing Systems'.

Bui, T. D., Yan, J. and Turner, R. E. [2017], 'A unifying framework for gaussian process pseudo-point approximations using power expectation propagation', *Journal of Machine Learning Research* **18**, 1–72.

Chatfield, C. [2003], *The Analysis of Time Series: An Introduction*, 6 edn, Chapman and Hall/CRC, New York.

Dai, Y., Yan, W. and Yin, F. [2024], Graphical multioutput gaussian process with attention, *in* 'International Conference on Learning Representations', The Chinese University of Hong Kong, Shenzhen.

Gardner, J. R., Pleiss, G., Bindel, D., Weinberger, K. Q. and Wilson, A. G. [2018], Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration, *in* 'Advances in Neural Information Processing Systems'.

Gibbs, M. [1997], Bayesian Gaussian Processes for Regression and Classification, Phd thesis, University of Cambridge.

Goodfellow et al [2016], *Deep Learning*, MIT Press. Deep Learning Book URL.

Google DeepMind [2023], 'Learning skillful medium-range global weather forecasting', *Science* pp. 1416–1421. Article URL.

Goovaerts, P. [1997], *Geostatistics for Natural Resources Evaluation*, Oxford University Press, USA.

Gramacy, R. B. and Lee, H. K. H. [2009], 'Bayesian treed gaussian process models with an application to computer modeling'.

Hayashi, K., Imaizumi, M. and Yoshida, Y. [2019], 'On random subsampling of gaussian process regression: A graphon-based analysis'.

Helm, T. [2023], 'Onshore wind projects in england stall as no new applications are received', *The Guardian* .

Hensman, J., Fusi, N. and Lawrence, N. D. [2013], Gaussian processes for big data, *in* '29th Conference on Uncertainty in Artificial Intelligence', pp. 282–290.

Jiang, G., Xie, P., He, H. and Yan, J. [2018], 'Wind turbine fault detection using a denoising autoencoder with temporal information', *IEEE/ASME Transactions on Mechatronics* **23**(1), 89–100.

Journel, A. G. and Huijbregts, C. J. [1978], *Mining Geostatistics*, Academic Press, London.

Keeling, C. D. and Whorf, T. P. [2004], 'Atmospheric CO2 records from sites in the SIO air sampling network.', *Trends: A Compendium of Data on Global Change. Carbon Dioxide Information Analysis Center.* .

Khan, P. W. and Byun, Y. C. [2024], 'A review of machine learning techniques for wind turbine's fault detection, diagnosis, and prognosis', *International Journal of Green Energy* **21**(4), 771–786.

Kingma, D. and Ba, J. [2015], Adam: A method for stochastic optimization, *in* 'International Conference on Learning Representations (ICLR)', San Diega, CA, USA.

Kostantinos N., D. H. [2017], 'Gaussian Mixtures and Their Applications to Signal Processing', pp. 89–124.

Lalchand, V., Tazi, K., Cheema, T., Turner, R. and Hosking, S. [2023], 'Kernel learning for explainable climate science'.

Lee, W. and Sutherland, J. [2024], 'Time to failure prediction of rotating machinery using dynamic feature extraction and gaussian process regression', *International Journal of Advanced Manufacturing Technology* **130**, 2939–2955.

Li, H., Xu, Z., Taylor, G., Studer, C. and Goldstein, T. [2018], Visualizing the Loss Landscape of Neural Nets, *in* 'Advances in Neural Information Processing Systems'. Project Repository.

Ohba, M. [2019], 'The impact of global warming on wind energy resources and ramp events in japan', *Atmosphere* **10**(5), 265. Link to Article.

Pandit, R. K. and Infield, D. [2018], 'Scada-based wind turbine anomaly detection using gaussian process models for wind turbine condition monitoring purposes', *IET Renewable Power Generation* **12**(11), 1249–1255. .

Parra, G. and Tobar, F. [2017], Spectral mixture kernels for multi-output gaussian processes, *in* 'Advances in Neural Information Processing Systems', pp. 6684–6693.

Paun, I., Husmeier, D. and Torney, C. J. [2023], 'Stochastic variational inference for scalable non-stationary gaussian process regression', *Statistics and Computing* **33**, 2–21.

Rasmussen, C. E. and Williams, C. K. I. [2006], *Gaussian processes for machine learning*, MIT Press.

Remes, S., Heinonen, M. and Kaski, S. [2017], Non-stationary spectral kernels, *in* '31st Conference on Neural Information Processing Systems'.

Rogers, T., Gardner, P., Dervilis, K., Maguire, A., Papatheou, E. and Cross, E. [2019], 'Probabilistic modelling of wind turbine power curves with application of heteroscedastic gaussian process regression', *Journal of Wind Energy* .

Ruder, S. [2016], 'An overview of gradient descent optimization algorithms'. Link to article.

Snelson, E. and Ghahramani, Z. [2006], Sparse gaussian processes using pseudo-inputs, *in* 'Neural Information Processing Systems', Vol. 18.

Snoek, J., Swersky, K., Zemel, R. and Adams, R. P. [2014], 'Input warping for bayesian optimization of non-stationary functions', *JMLR* .

Stetco, A., Dinmohammadi, F., Zhao, X., Robu, V., Flynn, D., Barnes, M., Keane, J. and Nenadic, G. [2019], 'Machine learning methods for wind turbine condition monitoring.', *Renewable Energy* .

Tazi, K., Lin, J. A., Viljoen, R., Gardner, A., John, S., Ge, H. and Turner, R. E. [2023], 'Beyond Intuition, a Framework for Applying GPs to Real-World Data'. Link to Arxiv.

Teh, Y. W., Seeger, M. and Jordan, M. I. [2005], Semiparametric latent factor models, *in* 'AISTATS', pp. 333–340.

Titsias, M. [2009], Variational learning of inducing variables in sparse gaussian processes, *in* 'Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics', Vol. 5, PMLR, pp. 567–574.

Wackernagel, H. [2003], *Multivariate Geostatistics*, Springer-Verlag, New York.

Wilson, A. and Adams, R. [2013], Gaussian process kernels for pattern discovery and extrapolation, *in* 'Proceedings of the 30th International Conference on Machine Learning', Vol. 28, PMLR, pp. 1067–1075.

World Energy [2023], Wind Power Total Installed Global Wind Energy Capacity Grew to 906 GW. This Represents Year-On-Year Growth of 9%, Technical report, GWEC.

Yadav, A., Bareth, R., Kochar, M., Pazoki, M. and Sehiemy, R. [2024], 'Gaussian process regression-based load forecasting model', *IET Generation, Transmission & Distribution* **18**, 899–910.

Yang, C., Liu, J., Zeng, Y. and Xie, G. [2019], 'Real-time condition monitoring and fault detection of components based on machine-learning reconstruction model', *Renewable Energy* **133**, 433–441.

Zhang, Y., Li, M., Dong, Z. Y. and Meng, K. [2019], 'A probabilistic anomaly detection approach for data-driven wind turbine condition monitoring', *CSEE Journal of Power and Energy Systems* .

Zhao, J. and Sun, S. [2016], 'Variational dependent multi-output gaussian process dynamical systems', *Journal of Machine Learning Research* **17**, 1–36.

Zhu, A., Zhao, Q., Yang, T., Zhou, L. and Zeng, B. [2023], 'Condition monitoring of wind turbine based on deep learning networks and kernel principal component analysis', *Computers and Electrical Engineering* **105**.

Zoubin Ghahramani [2013], 'Bayesian non-parametrics and the probabilistic approach to modelling'.