Université Fédérale
Toulouse Midi-Pyrénées

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Institut Supérieur de l'Aéronautique et de l'Espace (ISAE)*

Présentée et soutenue le *07/12/2017* **par :**

ANKIT CHIPLUNKAR

Incorporating Prior Information from Engineering Design into Gaussian Process Regression Models

---

JURY

NICOLAS GAYTON	IFMA, Clermont-Ferrand	Rapporteur
ALEXANDER FORRESTER	University of Southampton	Rapporteur
RODOLPHE LE RICHE	CNRS, Saint-Étienne	Examinateur
NATHALIE BARTOLI	ONERA, Toulouse	Examinateur
JOSEPH MORLIER	ISAE-Supaero, Toulouse	Directeur de thèse
EMMANUEL RACHELSON	ISAE-Supaero, Toulouse	Co-directeur de thèse

---

École doctorale et spécialité :
AA : Aéronautique et Astronautique

Unité de Recherche :
Institut Clément Ader

Directeur(s) de Thèse :
Prof. Joseph MORLIER (directeur de thèse), Dr. Emmanuel RACHELSON (co-directeur de thèse) et *M. Michele COLOMBO (superviseur industriel)*

Rapporteurs :
Prof. Nicolas GAYTON et *Prof. Alexander FORRESTER*

Contents

List of Figures

List of Matlab Codes

Part I

Introduction

Chapter 1

Context

Machine learning

In the past decade due to the boom in Information Technology (IT) companies, more investment has gone into developing both computational infrastructure and methods. One of the ubiquitous methods developed due to this investment is that of Machine Learning. Learning algorithms look for patterns in data to learn from them and make decisions. They are used in web search, optimization, spam filtering, ad placement, stock trading, healthcare, manufacturing, space exploration, particle physics, security, and lot more. The speed and adaptability of learning methods are changing everything around us one algorithm at a time [Domingos 2015]. The World Economic Forum, Davos 2016 [Schwab 2016] has dubbed this as the fourth industrial revolution; first was steam powered, second was electrically powered, third was IT powered, fourth will be powered by artificial intelligence algorithms.

Aircraft design

In comparison aircraft design is almost a century old field, the first successful flight being by the Wright brothers in 1903 [Wright 1934]. Currently, the design of an aircraft is a highly-iterative optimization process. Based on the initial target objectives and general trends, aircraft designers define the objectives for domain specific departments (eg. aerodynamics design or structural design). These objectives further trickle down to more dedicated teams and tasks (eg airfoil design or fuselage design). This gives rise to a huge Multi-Disciplinary Optimization (MDO) problem. Teams come up with individual constraints, they iterate around their domains, interact with neighboring domains and negotiate overlapping constraints. Teams negotiate and solidify individual objectives and constraints to find the optimized design for an aircraft. An optimized design as close to the initial target objectives, an optimized design taking into account the sparse infrastructure, human and economic limitations. This is a massive MDO problem spanning almost a decade, costing billions and involving thousands of people.

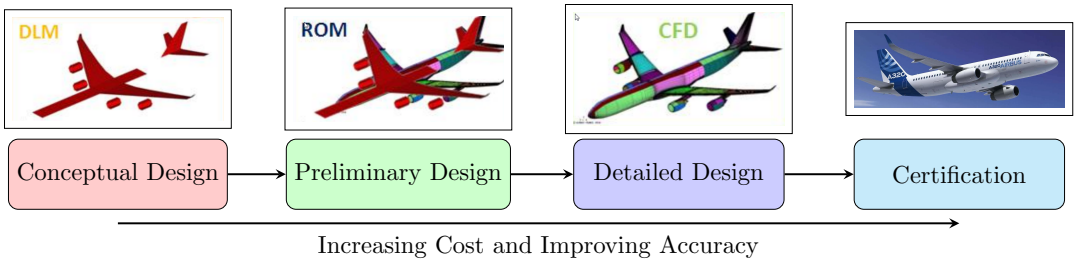


Figure 1.1: Phases of an aircraft design cycle

1.1 Aircraft design cycle

Design phases

To simplify this process an aircraft design is broken down into several design phases (figure 1). Each phase requires an ever increasing amount of predictions and fidelity. Preliminary design phase requires a few low-fidelity design trade-offs between major disciplines. Whereas, during detailed design phase, intensive intra-disciplinary and inter-disciplinary optimization's take place. Finally, during the flight-test and certification phase capability to predict real-time can provide significant gains in reducing the flight-test phase. These analyses cover large parts of flight envelope and require high-fidelity predictions. Hence the capability to accurately and quickly predict is an integral part of an aircraft design cycle [Raymer 2012].

Need for Speed

In the last decade high-fidelity, physics based, mathematical simulations have become central to designing an aircraft. However, high-fidelity simulations are computationally expensive, this is the case for several Computational Fluid Dynamics (CFD) and Finite Element Method (FEM) based solvers. Due to this high cost high-fidelity simulations are launched only for a few carefully chosen design configurations. This results in inefficient exploration of the the design space and thus a non-optimal design. A common strategy to speed up simulations is by reducing the physical complexity of the model to make quick predictions. As an example linear potential flows (simpler aerodynamic model), or coarser FEM meshes (simpler structural model) are regularly used during the preliminary design phase [Cummings 2015]. While this is an acceptable practice during the preliminary design phase, during the detailed design phase physical complexity is needed to find a robust optimum design point [Raymer 2012].

Need for accuracy

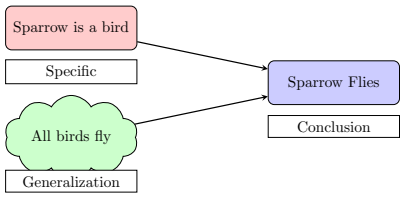
Instead of approximating physical complexity, surrogate models[1] simplify mathematical complexity [Verveld 2016]. Surrogate models learn patterns between the input and output dataset and then are used to make predictions on the desired point. This prop-

[1]Surrogate models, learning algorithms and machine learning models will be used interchangeably throughout this manuscript

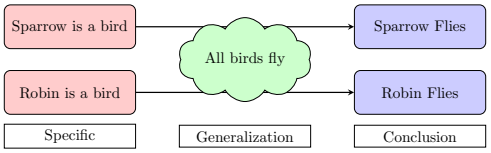
erty is very useful in quickly exploring the design space and finding a robust design point [Forrester 2008]. Moreover, surrogate models are commonly passed across disciplines to perform inter-disciplinary optimizations. For example, a loads department would prefer running a quick surrogate model over the costly CFD model while performing the load's loop.

Deduction vs Induction

The main difference between the engineering design and surrogate modeling can be explained by the difference between deduction and induction [Domingos 2012] (figure 1.1). Engineering design is deduction: where a very general formula is applied to a particular case (figure 1.2(a)). The basics of Newtonian physics, when applied to a particular aircraft geometry, give inertial loads. The basics of aerodynamics, when applied to a particular set of aircraft geometry and aircraft states, give out aerodynamic pressures. Engineering design takes global rules and applies them to local configurations. Whereas, surrogate modeling is induction: it looks at local features and data, tries to find similarity measures between them and gives a global formula for the process (figure 1.2(b)). For example, an algorithm to detect faces in images will look at several images with and without faces, learn a facial pattern and make predictions on new images [Marszalek 2007]. We here see a possible complementary relationship between engineering design and machine learning; where engineering design needs models to generate data, machine learning needs data to generate models.



(a) Engineering design - Deduction: The figure shows an application of a general rule to a particular case.



(b) Surrogate modelling - Induction: The figure shows how multiple examples can be used to infer underlying rules or patterns that govern the system.

Figure 1.2: Induction vs Deduction

Developing faster models

Since models are an integral part of any engineering design, model building in aircraft engineering was traditionally outsourced to research institutes. Researchers perform iterative experiments in a controlled environment and discover patterns between the physics of the system. For example, it took 200 years to iteratively develop the Gas law [2], Boyle's law in 1600's found the relation between Pressure and Volume, Charle's Law in 1787 discovered the relationship between Volume and Temperature, while Gay-Lussac's Law in 1809 discovered the relationship between Pressure and Temperature [Clapeyron 1834]. This is a rigorous and time-consuming method of developing models. Machine learning is a much

[2] $Pressure \times Volume \propto numberOfMoles \times Temperature$

more elegant method of building models. Using data and few basic assumptions, automatic models can be built between desired inputs and outputs. For example, while the first model of a neural network was proposed in 1950's [Kleene 1951], neural networks are today used daily for tasks such as tagging cat photos on facebook and converting speech to text[3]. In this thesis, we wish to automatically build models for aircraft design tasks primarily to be used during the detailed design phase and certification phase.

1.2 Machine Learning

Components of learning

The core objective of learning algorithms is to find a transformation function between the inputs and outputs. There are three main components in a learning algorithm:

1. **Representation**: A learning algorithm starts with a family of functions. For example, a linear model is a family of linear functions, a trigonometric model defines a family of trigonometric functions. If an algorithm is not able to represent the actual function in its family of functions, it will find the closest function in its hypothesis space[4].

2. **Evaluation**: Some measure is needed to distinguish a good function from a bad function in the chosen hypothesis space. This measure is termed as evaluation; one example is the least squares error commonly used in many learning algorithms.

3. **Optimization**: Finally, the algorithm iteratively searches in its hypothesis space to find the best possible function explaining the data. The choice of optimizer defines the speed of learning and is also important if there are multiple minima in the evaluation criteria.

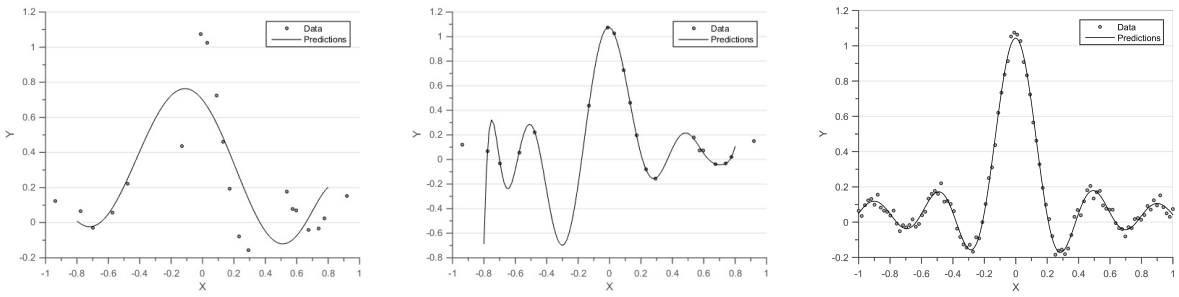
Bias vs Variance

Surrogate models suffer from the bias vs variance trade-off (figure 1.3), formalized by 'Wolpert' in his famous "no free lunch theorem" [Wolpert 1997]. The constituent functions in a hypothesis space represent the bias or assumptions of the learning algorithm (eg. linear functions for linear regression). In the absence of sufficient assumptions, the family of functions in the search space becomes very large which leads to high variance or over-fitting in the surrogate model (figure 1.3(b)). On the contrary, wrong bias means that the true transformation function (f) does not exist in the hypothesis space. In this case, the learning algorithm finds the function closest to f in its hypothesis space and leads to under-fitting (figure 1.3(a)).

Soft and hard constraints

[3] I wrote a fourth of this thesis using a text to speech software

[4] The term family of functions, hypothesis space and representation will be used interchangeably throughout this manuscript



(a) High-bias and low variance (b) Low bias and High variance (c) Bias replaced with lots of data

Figure 1.3: Bias vs Variance trade-off

One method to overcome this trade-off is by using lots and lots of data. Data acts as a hard constraint for learning algorithm, imagine a family of all possible continuous 2-dimensional functions in the range of $x \in [-1, 1]$ and $y \in [-1, 1]$. What will happen, given an observation $(x = 0, y = 0)$. All the functions that do not pass through this point will be eliminated, the new data point has basically reduced the possible family of functions. Whereas, bias acts as soft constraint, we can use the bias of linearity or smoothness to reduce our hypothesis space. Therefore, both bias and data help in reducing the hypothesis space[5]. Given access to more and more data, we can progressively reduce the bias in learning models thereby relying more on true evidence. This is also the main concept behind deep learning, where several layers of neural networks define a very large hypothesis space [Goodfellow 2016, LeCun 2015].

Unfortunately in aircraft design, generating a huge amount of accurate data is a costly exercise, for example a high fidelity CFD simulation runs for weeks [Murthy 2014, Jameson 2012] and a flight-test campaign costs millions of euros [Fox 2004]. On another hand due to centuries of research and tinkering, we have a treasure trove of prior information about the physical systems. We propose to build better machine learning models by integrating the time-tested prior knowledge of physical systems with experimental data.

Contribution

The main contribution of this thesis is to provide a framework on how to combine the prior knowledge from a physical system and add it to a learning algorithm. A model generated from merging of the two methodologies will be both consistent with the physics of the system and be quicker to evaluate. We integrate three types of prior knowledge by answering the following questions:

1. **Pattern**: How to add *apriori* information of a pattern in a learning algorithm?

[5]Bias can also be looked upon as distilled knowledge or patterns gained after interpreting huge amounts data

For example, given that shock is a discontinuous change in pressure, how to predict the position of shock on an airfoil (part II).

2. **Relationships**: How to add *apriori* information of relationships between measurements? For example given $Loads = \int Pressures$, how to make a robust loads model when we measure both pressures and loads (chapter **??**).

3. **Simulation model**: How to merge *apriori* information of simulations with experiments? For example, given a simulation model and experimental data how to perform extrapolations on experimental data(chapter **??**).

To integrate the prior information we propose to use a Bayesian inference for model building. Bayesian inference is a method of statistical inference in which Bayes theorem is used to update an initial probability (prior) as more and more evidence becomes available the final probability (posterior) gives our estimate. More specifically, we will use the Gaussian Processes (GP) Regression framework which is a subset of the Bayesian Inference algorithms to define prior information of physical systems. But, before deep diving into the details of GP let us have a look at a simple Bayesian Linear Regression algorithm.

1.3 Bayesian Linear Regression

Suppose we have access to a training set of observations (or outputs) $Y = (y(x_1); \ldots; y(x_i); \ldots; y(x_N))$, evaluated at a set of known inputs $X = (x_1; \ldots; x_i; \ldots; x_N)$, and we wish to predict $y(x_*)$ at a test input x_*. The input and output can be multi-dimensional; $x_i \in \mathbb{R}^{D_{inputs}}$ and $y(x_i) \in \mathbb{R}^{D_{outputs}}$. The process of learning the transformation function f to make prediction at a new point is called as regression. In the following section we follow formulation for Bayesian Linear Regression provided by [MacKay 2003].

Basis functions

A simple method to perform regression is by assuming a form of the function f, then minimizing the error between the true measurements and assumed form to estimate the parameters of f. The function is written in terms of basis functions $\phi(x)$. For example when $\phi(x) = \{1, x\}^T$ we are performing linear regression, when $\phi(x) = \{1, x, x^2, \ldots, x^L\}^T$ we are performing L^{th} order polynomial regression. We will focus on linear regression in this section and hence $\phi(x) = \{1, x\}^T$.

$$f(x_i) = \begin{Bmatrix} 1 & x \end{Bmatrix} \begin{Bmatrix} w_0 \\ w_1 \end{Bmatrix} \qquad y(x_i) = f(x_i) + \epsilon \tag{1.1}$$

Likelihood

Here, w are the parameters of the function, such that w_0 denotes the intercept and w_1 denotes the slope of the regression model. The measurements are corrupted by independent

white noise ϵ, such that the noise is a random variable sampled from a white noise Gaussian with variance σ_n^2[6]. The above equations 1.1 can be combined to result in the likelihood $\Pr[Y \mid X, w]$

$$\begin{aligned}\Pr[Y \mid X, w] &= \prod \Pr[y_i \mid x_i, w] \\ &= \prod \mathcal{N}[[1, x_i]w, \sigma_n^2] \\ &= \mathcal{N}(X^T w, \sigma_n^2 I)\end{aligned} \tag{1.2}$$

The notation $\Pr[Y \mid X, w]$ symbolizes probability distribution of observations Y at the inputs X given the parameter w. The notation $\mathcal{N}[\mu, \Sigma]$ symbolizes a multi-variate Gaussian distribution for mean vector μ and covariance matrix Σ.

While performing Bayesian inference we specify a prior distribution to encode our assumptions on the parameters before we look at the observations. For this case, we put a zero mean Gaussian prior on our weights.

Prior

$$\Pr[w] = \mathcal{N}(0, \Sigma_{Prior}) \tag{1.3}$$

The prior distribution on w induces a prior distribution over functions parametrized by w, effectively we are defining family of functions ($\Pr[f(x_i)] = \mathcal{N}(0, x_i^T \Sigma_{Prior} x_i)$) by placing a prior distribution over w. Once we have a prior distribution encoding our beliefs, we use Bayes rule to look at the observations and get a posterior distribution of parameters.

$$posterior = \frac{likelihood \times prior}{marginal \quad likelihood}$$

$$\Pr[w \mid Y, X] = \frac{\Pr[Y \mid X, w] \times \Pr[w]}{\Pr[Y \mid X]} \tag{1.4}$$

The *marginal likelihood* is a normalization constant, for more details please refer to section 2.3. After, using the equation 1.2, 1.3 and 1.4 we can get the posterior distribution of weights as:

Marginal Likelihood

$$\Pr[w \mid Y, X] = \mathcal{N}\left(\frac{1}{\sigma_n^2} A^{-1} XY, A^{-1}\right) \tag{1.5}$$

Here, $A = \sigma_n^{-2} XX^T + \Sigma_{Prior}^{-2}$. Thus the posterior distribution for function f at test

[6] $\Pr[\epsilon] = \mathcal{N}(0, \sigma_n = \frac{1}{\sqrt{2\pi\sigma_n^2}} \exp^{-\frac{\epsilon^2}{2\sigma_n^2}}$

point x_* becomes:

$$\Pr[f \mid x_*, X, y] = \mathcal{N}\left(\frac{1}{\sigma_n^2} x_* A^{-1} XY, x_*^T A^{-1} x_*\right) \tag{1.6}$$

Posterior

The mean $\frac{1}{\sigma_n^2} x_* A^{-1} XY$ can be used as a prediction at the test point x_*, while the variance is a measure of uncertainty for this prediction. We can thus obtain the prediction $f(x_*)$, using a prior set of beliefs (equation 1.3) and updating those beliefs using observations.

Experiment

Suppose we have a toy dataset $\mathcal{D}_1 = \{X = [-0.5, 0.33, 0.66], Y = [0, 0.5, 0.5]\}$ and want to find a function that fits this dataset using Bayesian Linear Regression. If we assume a prior distribution of parameter w as defined by equation 1.7, then the probability density of w will look like figure 1.4(a).

$$\Pr\left[\begin{pmatrix} w_0 \\ w_1 \end{pmatrix}\right] = \mathcal{N}\left[\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \Sigma_P = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right] \tag{1.7}$$

Using equations 1.7 and 1.6 we can calculate the posterior distribution for the parameters. For a noise model of $\Pr[\epsilon] = \mathcal{N}(0, (\sigma_n = 0.1)^2)$ the posterior distribution comes out as the figure 1.4(b)

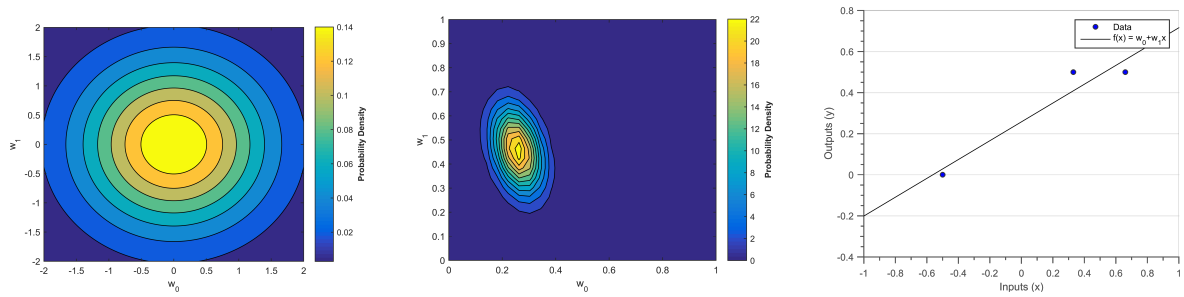
$$\Pr\left[\begin{pmatrix} w_0 \\ w_1 \end{pmatrix} \mid \mathcal{D}_1, \Sigma_P, \sigma_n\right] = \mathcal{N}\left[\begin{pmatrix} 0.2576 \\ 0.4584 \end{pmatrix}, \begin{bmatrix} 0.0037 & -0.0022 \\ -0.0022 & 0.0138 \end{bmatrix}\right] \tag{1.8}$$

This simple example demonstrates how to estimate the parameters w_0 and w_1 using the prior distribution on parameters w and dataset $\mathcal{D}_1$. The estimate of the intercept is $w_0 = 0.2576$ and slope is $w_1 = 0.4584$ (equation 1.8), we can use this estimate to plot the value of linear function $f(x)$ (figure 1.4(c)).

While the Bayesian Linear Regression framework provides an opportunity to encode prior assumptions in terms of distributions of the parameters. A much more elegant and expressive method is by using Gaussian Processes to perform regression. Gaussian Processes are a distribution over functions and hence enable us to encode prior knowledge directly in the functional space.

Non-parametric models

Learning algorithms are mainly divided into two main types. The first is defined by parametric models which can only represent a limited hypothesis space. They use parameters to describe the function between input and output domain. For example the weight parameters w in Bayesian Linear Regression. The second are non-parametric models whose hypothesis space grows with the size of data [Ghahramani 2013]. Non-parametric models use data to represent functions, Gaussian Process Regression is a type of non-parametric



(a) Contours of probability density for a prior on parameters w as defined by equation 1.7

(b) Contours of probability density for posterior distribution of parameters w using equations 1.7 and 1.5. The predicted intercept is 0.2576 and the predicted slope is 0.4584

(c) Linear Prediction based on the posterior, given the Dataset $\mathcal{D}_1$ prior $\Sigma_P$and noise σ_n. The predicted linear line does not pass through the data points but between them.

Figure 1.4: Prior, Posterior and Prediction in Bayesian Linear Regression

model. One can imagine a non-parametric model like a stretched rubber sheet: whenever it sees data it deforms accordingly to compensate for the new data point. Hence the more data it sees the more it starts mimicking the actual function.

Gaussian Process

Gaussian Process or Kriging was first used in the context of Geo-statistics research by Daniel Krige [Krige 1951], this was later formalized by Matheron in his seminal work "Principals of Geo-statistics" [Matheron 1963]. Recently, interest in the Gaussian Process (GP) grew in the machine learning community from neural networks research. It was shown that a Bayesian neural network becomes a Gaussian process as the number of neurons tends to infinity [Neal 2012]. Gaussian processes are probabilistic distributions over functions, which provide a Bayesian non-parametric approach to smoothing and interpolation. A Gaussian Process can be fully parameterized by its mean and covariance function. More generally, a Gaussian Process is a method to probabilistically define a family of functions, chapter 2 expands GP in more detail.

1.4 Outline

This thesis is divided into three main parts, each part is then divided into individual chapters and their constituent sections. This part sets up the prerequisites required to understand the concepts introduced in the next two parts. The first chapter demonstrates the need for performing regression in aircraft design tasks and describes a very basic Bayesian Linear Regression scheme.

Chapter 2

Chapter 2 shows the key processes involved in a GP regression framework. GPs as distributions over functions have a rich history in geo-statistics and machine learning.

The second chapter heavily draws ideas from [Krige 1951, Matheron 1963] of the geo-statistics community and [Stein 1999, Kennedy 2000, Rasmussen 2005, MacKay 2003] of the machine-learning community, showing a process flow of how to perform regression using GPs. The remaining chapter unfolds as follows, section 2.1 describes the key constituents of a GP and how to draw random functions from a GP. Section 2.2 describes how to perform prediction in presence and absence of measurement noise. Section 2.3 introduces marginal likelihood as a form of evaluation method to automatically choose hyper-parameters.

Chapter 3

Chapter 3 deals with the problem of scaling GP regression to massively many points. Traditional GPs are computationally infeasible on a standard laptop if the number of data points increase $\mathcal{O}(10^4)$. There exist two main methods to scale a GP regression, one using reduced set of inducing points while another based on mixture of experts methodology. This chapter draws heavily from the works of [Quiñonero-Candela 2005, Seeger 2003, Snelson 2006, Titsias 2009] for the approximation method of inducing points and [Cao 2014, Tresp 2000, Chen 2009, Deisenroth 2015] for the approximation method of mixture of experts, please refer to the individual publications for more detail. We demonstrate the limitations and capabilities of both the methods on a toy dataset, giving directions to choosing optimal parameters and extracting the best possible result.

Description of the other two parts

Chapter 2

Gaussian Process Regression

Suppose we perform a simulation or experiment on an input point $x_j \in \mathbb{R}^{D_{inputs}}$and measure an output $y_j \in \mathbb{R}$. In this chapter we assume that the input is D_{inputs} dimensional and the output is one dimensional. We can thus have a data set of N observations, $\{\mathcal{D} = (x_j, y_j)|j \in [1; N]\}$. The full input and output vectors can be denoted as $X = \{x_1; x_2; \ldots; x_N\}$ and $Y = \{y_1; y_2; \ldots; y_N\}$ such that $X \in \mathbb{R}^{N \times D_{inputs}}$ and $Y \in \mathbb{R}^N$. Given this data, we are interested in making predictions for new input points x_* [1]that are not present in our series of experiments. This means that we need to use our training data and learn the true physical process $f(x)$ that generates our data set.

As discussed in the previous chapter, the first step of a learning algorithm is defining a family of functions. Gaussian Processes (GPs) can be used to probabilistically define these family of functions. More formally, a GP is a distribution over functions such that any finite set of function values $[f(x_1), f(x_2), \ldots, f(x_N)]$ have a joint Gaussian distribution [Rasmussen 2006].

Infinite dimensional random vector

While a normal distribution describes a scalar random variable, for example $X \sim \mathcal{N}(0, 1)$ defines a Gaussian variable with mean 0 and variance 1. A multi variate distribution defines a vector of random variables, for example $\{X\} \sim \mathcal{N}(\{0, 0\}, [1, 0; 0, 1])$ defines a Gaussian vector with mean $\{0, 0\}$ and covariance $[1, 0; 0, 1]$. A GP is the extension of this concept in the functional space. We can also think of a function as an infinite dimensional vector, with each entry in the vector specifying the function value $f(x)$ at a particular point x [2].

Mean

A GP model before conditioning on data can be completely parameterized by its mean $(m(x))$

$$\mathbf{E}[f(x)] = m(x) \tag{2.1}$$

[1]Also called as test point, prediction point or target point.

[2]Yes blew my mind as well!

Covariance

and its covariance function ($k(x_1, x_2)$) also called a kernel. In the context of the GPs a kernel is a measure of similarity between pairs of functional values ($f(x)$) evaluated at input points , often involving an inner product of basis functions $\phi(x)$ [Bishop 2006]. Please refer to chapter 4 for a more detailed insight into kernels. [3]:

$$Cov[f(x) - m(x), f(x') - m(x')] = k(x_1, x_2) \tag{2.2}$$

We can formally write the probability of the function f as:

$$\Pr[f(x)] = GP(m(x), k(x_1, x_2)) \tag{2.3}$$

The notation $\Pr(f(x))$ symbolizes probability distribution of function f at the input point x. A function randomly drawn from a GP yields a random function around the mean function $m(x)$, and its shape is defined by the covariance function $k(x_1, x_2)$.

Tractable

Performing inference on an infinite dimensional vector (function) can be a computationally intensive task. Thankfully, due to the marginalization property of Gaussians, if we ask for properties of the function at a finite number of points, then GP will give us the same answer, if we ignore the infinitely many other points. In other words any finite set of function values $[f(x_1), f(x_2), \ldots, f(x_N)]$ have a joint Gaussian distribution in GP (this is also the formal definition of GP). This property means that GP specified in equation 2.3 also specifies equation 2.4. This makes GPs computationally tractable, which one of the major benefits of GP.

$$\Pr\begin{bmatrix} f(x_1) \\ f(x_2) \end{bmatrix} = \mathcal{N}\left(\begin{bmatrix} m(x_1) \\ m(x_2) \end{bmatrix}, \begin{bmatrix} k(x_1, x_1) & k(x_1, x_2) \\ k(x_2, x_1) & k(x_2, x_2) \end{bmatrix}\right) \tag{2.4}$$

While performing regression in a GP framework, we first define a family of functions also called prior (section 2.1). The next step involves looking at the dataset and eliminating all the functions in our prior hypothesis space which do not pass through the dataset, this step gives us the posterior mean and variance (section 2.2). Finally, we can further improve our predictions by fine-tuning our hyper-parameters (section 2.3).

2.1 Prior

In the Bayesian framework, a prior is a probability distribution before looking at any evidence. In the context of a GP Regression, this is provided by the mean and covariance function.

[3] The terms covariance functions, kernel and kernel functions will be used interchangeably during the reminder of this thesis

2.1.1 Hyperparameters

Both mean and covariance functions are specified by a set of hyper-parameters θ, these are the parameters of the GP. Selecting a prior in GP boils down to choosing an appropriate functional form of the mean and covariance matrix and then choosing the hyper-parameters of the prior [Duvenaud 2013].

Automatically, predicting the values of hyper-parameters is important to choose a good prior. We will look at how to choose good hyper-parameters in section 2.3.

2.1.2 Mean function

The mean function $m(x)$ of a GP represents its trend. In Universal Kriging, we usually choose a mean function of the form $m(x) = \phi(x)^T\theta$, with $\phi(x) = (\phi_1(x), \ldots, f_p(x))$ being a vector of basis functions, generally including a constant function and $\theta \in \mathbb{R}^p$ is a vector of hyper-parameters [Matheron 1963]. In Simple Kriging, we assume a constant mean function $m(x) = constant$.

Without loss of generality, we can assume the mean function to be zero everywhere, since uncertainty about the mean function can be taken into account by adding an extra term to the covariance function (Chapter 4). *Zero mean*

$$\Pr[f(x)] = GP(0, k(x_1, x_2, \theta)) \tag{2.5}$$

We assume a zero mean prior through out this section. After accounting for the zero mean, the GP model can be completely parametrized by the kernel. Hence the problem of learning in a GP is exactly the problem of finding suitable properties of the covariance function [Rasmussen 2006] (equation 2.5). The Matlab Code 2.1 below is a sample of zero Mean function.

```
% zero mean function
meanFunction = @(x) 0*x;
```

Matlab Code 2.1: A zero mean function

2.1.3 Covariance function

The covariance function is a positive definite kernel, such that for any $a_i \in \mathbb{R}$ equation 2.6 holds [Stein 1999].

$$\sum_{i=1}^{N}\sum_{j=1}^{N} a_i a_j k(x, x') \geq 0 \tag{2.6}$$

A popular choice of covariance function is a Standard Exponential (SE) function (equation 2.7), because it defines a family of highly smooth (infinitely differentiable) non-linear functions as shown in figure 2.2.

$$k_{SE}(x_1, x_2, \theta) = \theta^2_{amplitude} exp[-\frac{d^2}{2\theta^2_{lengthScale}}] \tag{2.7}$$

SE kernel

For the case of the SE kernel the hyper-parameters ($\theta = [\theta_{amplitude}, \theta_{lengthScale}]$) are amplitude ($\theta_{amplitude}$), which defines average distance from mean, and the length scale ($\theta_{lengthScale}$), which defines the smoothness of functions. Here, d defines the absolute distance between points $|x - x'|$. Covariance functions which are purely a function of distance d are called as isotropic stationary functions. These covariance functions remains unchanged if the points x_1, x_2 are rotated or translated. Hence a family of functions defined by stationary kernels will have similar local features throughout the input domain.

Length-Scale

When x tends to x', then $k(x_1, x_2)$ approaches $\theta^2_{amplitude}$ this means that $f(x)$ is highly correlated with $f(x')$. This is a good characteristic for smooth functions since points in the neighbourhood must be alike. If x is far away from x', then $k(x_1, x_2)$ tends to zero this means that far away points are loosely correlated. Hence, far off observations will have negligible effect while performing interpolations. How fast or slow the covariance decreases with distance depends on the length scale parameter $\theta_{lengthScale}$, smaller length-scale means a faster moving function. In general we cannot extrapolate more than $\theta_{lengthScale}$ units from the closest data-point [Duvenaud 2014].

The Matlab Code 2.2 below is a sample of SE covariance function, theta(1) is the amplitude hyper-parameter, while theta(2) is the length-scale hyper-parameter.

```
% Standard exponential covariance function
covarianceFunction = @(theta, x1, x2)(theta(1).^2*exp(-(x1
   - x2)^2/(2*theta(2).^2)));
% theta(1): is the amplitude hyperparameter
% theta(2): is the length-scale hyperparameter
```

Matlab Code 2.2: A SE covariance function

2.1.4 Sampling functions from GP priors

To have a look at the constituent functions in a prior we can randomly sample functions from the GP. Since any finite set of function values have a joint Gaussian distribution in a GP. To draw random functions from a GP we choose $N*$ input points $X_* = \{x_{1*}; x_{2*}; \ldots; x_{N*}\}$ and write corresponding mean vector $m(X_*)$ and covariance ma-

trix $K(X_*, X_*)$ [4] using equation 2.4 and 2.7. We then generate a random Gaussian vector $f(X_*)$ for this multi-variate Gaussian (equation 2.5) and plot the generated values as a function of inputs X_*.

$$K(X_*, X_*) = \begin{bmatrix} k(x_{1*}, x_{1*}) & k(x_{1*}, x_{2*}) & \dots & k(x_{1*}, x_{N*}) \\ k(x_{2*}, x_{1*}) & k(x_{2*}, x_{2*}) & \dots & k(x_{2*}, x_{N*}) \\ \vdots & \vdots & \ddots & \vdots \\ k(x_{N*}, x_{1*}) & k(x_{N*}, x_{2*}) & \dots & k(x_{N*}, x_{N*}) \end{bmatrix} \tag{2.8}$$

The Matlab Code 2.3 below is a sample function to evaluate the Gram Matrix. The function evaluateGramMatrix will later be used regularly to calculate, the posterior mean, posterior variance (code 2.5) and choosing hyper-parameters (code 2.7).

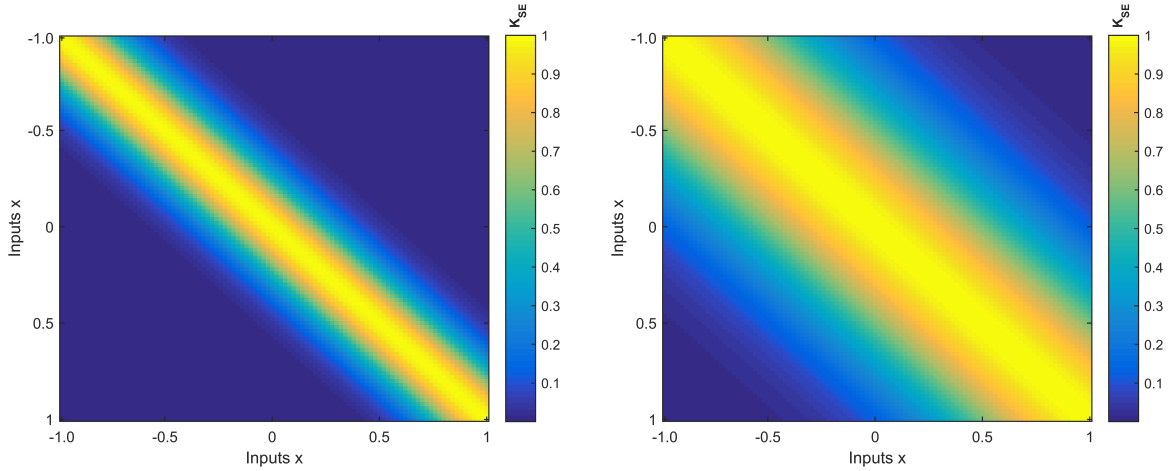
```
% Function to evaluate the gram matrix
function gramMatrix = evaluateGramMatrix(covarianceFunction
   , theta, x1, x2)

        for i=1:length(x1)
                for j=1:length(x2)
                        gramMatrix(i, j) =
                           covarianceFunction(theta, x1(i),
                            x2(j));
                end
        end

end

% Initial parameters
N = 100; % Number of testing points
xStar = linspace(-1, 1, N)'; % Input training points

theta(1) = 1; % Amplitude Hyper-parameter
theta(2) = 0.2; % Length Scale Hyper-parameter

% Visualizing the Gram matrix
gramMatrix = evaluateGramMatrix(covarianceFunction, theta,
   xStar, xStar);
```

[4] The covariance matrix is also called the Gram matrix

```
imagesc(gramMatrix); % Plotting the Gram Matrix
```

Matlab Code 2.3: Plotting the Gram Matrix

Figure 2.1 shows the covariance matrix for SE kernel with different hyper-parameters at the input points $X^* = \{[0 : 0.02 : 1]\}$. The SE kernel of figure 2.1(a) has a lower length-scale than figure 2.1(b). Notice how the covariance values are more spread out for figure 2.1(b).



(a) Covariance matrix for a SE Kernel with ($\theta = [1, 0.2]$) at the input points $X^* = \{[0 : 0.02 : 1]\}$.

(b) Covariance matrix for a SE kernel with ($\theta = [1, 0.5]$) at the input points $X^* = \{[0 : 0.02 : 1]\}$

Figure 2.1: Covariance matrix for a SE kernel with different hyper-parameters at the input points $X^* = \{[0 : 0.02 : 1]\}$. The SE kernel of figure 2.1(a) has a lower length-scale than figure 2.1(b). Notice how the covariance values are more spread out for figure 2.1(b).

To generate a random Gaussian vector $f(X_*)$ of length N_*, we first calculate the Cholesky decomposition[5] of the covariance matrix $K(X_*, X_*) = LL^T$, where L is a lower triangular matrix. We then generate a random vector U, such that $U = \mathcal{N}(0, I)$ and I is an identity matrix of size N_*. The random vector can then be computed as $f(X_*) = m(X_*) + LU$, and when plotted with the inputs X_*, gives a randomly drawn function. The Matlab Code 2.4 shows how can we draw functions randomly from a GP prior.

```
meanVector = meanFunction(xStar);

% Plotting the variance of Prior
f = [meanVector+2*sqrt(diag(gramMatrix)); flipdim(
```

[5]Cholesky Decomposition is also called the square-root of matrix and is defined for positive definite matrices

```
   meanVector-2*sqrt(diag(gramMatrix)),1)];
hold on; fill([xStar; flipdim(xStar,1)], f, [7 7 7]/8)

% Plotting the mean of prior
hold on; plot(xStar, meanVector, 'k');

% Tip: add a jitter term to the gram matrix so that matrix
   inversion is numerically stable
jitter = 10^(-6);

% The cholesky decomposition
L = chol(gramMatrix + eye(N)*jitter);

% Plotting the randomly drawn sample function
randomFunction = meanVector + L'*rand(N, 1);
hold on; plot(xStar, randomFunction, 'b');
```

Matlab Code 2.4: Sampling a random function from the prior

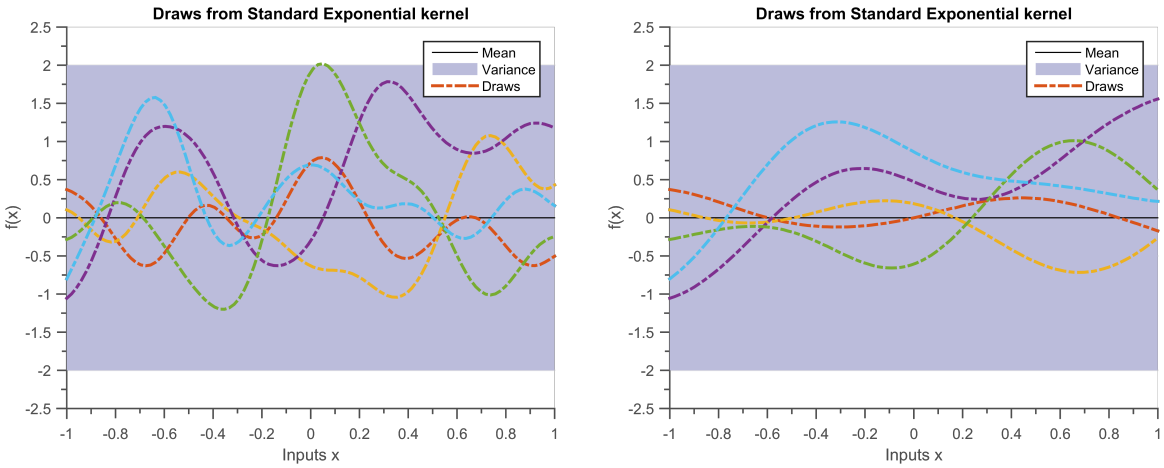
Figure 2.2 shows 5 random functions drawn for a zero mean GP with the covariance matrices of figure 2.1. The solid black line defines the mean function, shaded blue region defines 95% confidence interval (2σ) distance away from the mean. The dashed lines represent five functions drawn at random from a GP prior. We can observe that figure 2.2(a) varies faster when compared to figure 2.2(b) due to smaller length scale hyper-parameter.

2.2 Posterior

Once we have defined an appropriate prior we wish to incorporate the information of training data set into the probabilistic framework. In the Bayesian framework, a posterior is the probability distribution after updating the information of evidence into prior knowledge.

2.2.1 Posterior with Noise-free observations

We first consider the case of noise-free observations, that is $\{y(x_i) = f_i \forall i \in [1; N]\}$. This is the case for many high-fidelity computer simulations since high-fidelity computer simulations can be treated as having no noise [Sacks 1989]. If we desire to interpolate at test points X_*, then the joint distribution of the training outputs $f(X)$ and test outputs



(a) Draws from a GP prior with mean zero and SE Kernel with ($\theta = [1, 0.2]$).

(b) Draws from a GP prior with mean zero and SE kernel with ($\theta = [1, 0.5]$)

Figure 2.2: The solid black line defines the mean function, shaded blue region defines 95% confidence interval (2σ) distance away from the mean. The dashed lines represent five functions drawn at random from a GP prior. We can observe that figure 2.2(a) varies faster when compared to figure 2.2(b) due to smaller length scale hyper-parameter.

$f(X_*)$ is given by equation 2.9.

$$\Pr\begin{bmatrix} f(X) \\ f(X_*) \end{bmatrix} = \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} K(X,X) & K(X,X_*) \\ K(X_*,X) & K(X_*,X_*) \end{bmatrix}\right) \tag{2.9}$$

$K(X, X_*)$ is $N \times N_*$ covariance matrix between the training points X and test points X_* (equation 2.8). The other covariance matrices $K(X, X)$, $K(X_*, X)$ and $K(X_*, X_*)$ can be computed similarly.

The posterior will be the the conditional probability of $f(X_*)$ given the prior and data set. For a multi-variate Gaussian the conditional distribution is also a multi-variate Gaussian and can be calculated tractably, for a more detailed derivation refer to appendix **refer to the appendix here**. Graphically, we can imagine that the Bayes theorem is removing all the functions from our prior family of functions that do not pass through the data set (figure 2.3). The predicted distribution after adding the information of data set into the prior can be written as:

$$\begin{aligned} \Pr(f(X_*) \mid X_*, X, f(X)) = GP(&K(X_*,X)(K(X,X))^{-1}f(X), \\ &K(X_*,X_*) - K(X_*,X)(K(X,X))^{-1}K(X,X_*)) \end{aligned} \tag{2.10}$$

The term $K(X_*, X)(K(X, X))^{-1}f(X)$ is the predicted mean of the posterior at the

test points X_*. The term $K(X_*, X_*) - K(X_*, X)(K(X, X))^{-1}K(X, X_*)$ is the predicted covariance.

We again use the dataset $\mathcal{D}_1$ to calculate the posterior distribution. The Matlab Code 2.5 shows how to calculate the posterior mean and variance. Notice, that calculating the posterior mean and variance are only a few lines of code.

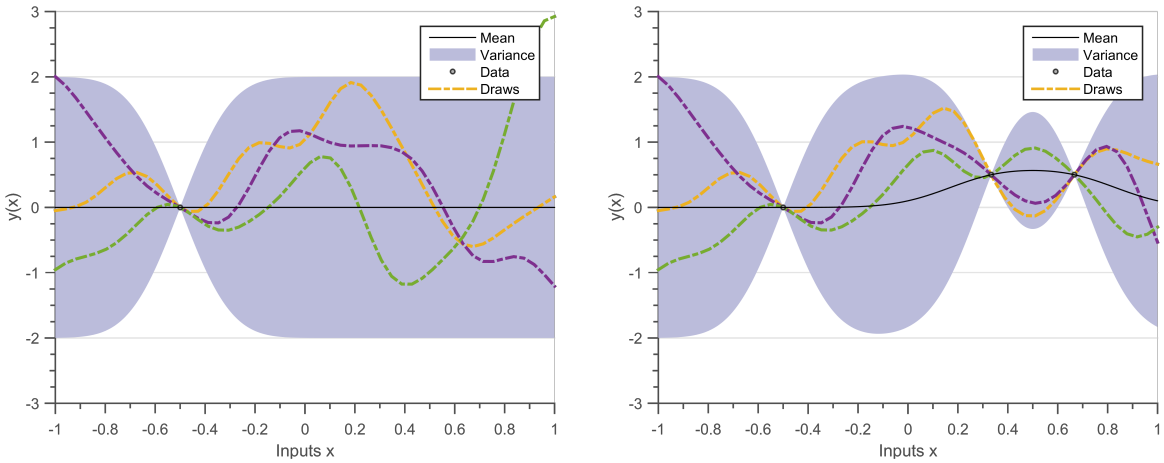
.

```
%% Evaluating the posterior
x = [-0.5, 1/3, 2/3]';
y = [0, 0.5, 0.5]';

gramMatrixXstarX = evaluateGramMatrix(covarianceFunction,
   theta, xStar, x);
gramMatrixXX = evaluateGramMatrix(covarianceFunction, theta
   , x, x);
gramMatrixXstarXstar = evaluateGramMatrix(
   covarianceFunction, theta, xStar, xStar);

meanPosterior = gramMatrixXstarX*inv(gramMatrixXX)*y;
covariancePosterior = gramMatrixXstarXstar -
   gramMatrixXstarX*inv(gramMatrixXX)*gramMatrixXstarX';

% Plotting mean, variance and sample of the posterior
% Plotting the variance of Prior
f = [meanPosterior+2*sqrt(diag(covariancePosterior));
   flipdim(meanPosterior-2*sqrt(diag(covariancePosterior))
   ,1)];
hold on; fill([xStar; flipdim(xStar,1)], f, [7 7 7]/8)

% Plotting the mean of prior
hold on; plot(xStar, meanPosterior, 'k');

% Tip: add a jitter term to the gram matrix so that matrix
   inversion is numerically stable
jitter = 10^(-6);
L = chol(covariancePosterior + eye(N)*jitter);
randomFunction =meanPosterior  + L'*rand(N, 1);
hold on; plot(xStar, randomFunction, 'b');
```

```
% Plotting the data
hold on; plot(x, y, '*');
```

Matlab Code 2.5: Calculating and plotting the mean, the variance and a sample of the posterior

Figure 2.3 shows the posterior GP after adding observed data into the initial prior. We can see that the Bayes theorem eliminates from the prior, all the functions that do not pass through the data. The solid black line defines the mean function, blue region defines 95% confidence interval (2σ) distance away from the mean. The mean of the of the posterior distribution is also used as a point estimate for interpolation. The dashed lines represent three functions drawn at random from a GP posterior. Random functions can be sampled from the posterior distribution as described in the earlier section.



(a) Posterior distribution for the case of noiseless observations. Prior is a GP with mean zero and covariance as SE Kernel with ($\theta = [1, 0.2]$), data set is $\{x = -0.5; f = 0\}$.

(b) Posterior distribution for the case of noiseless observations. Prior is a GP with mean zero and covariance as SE Kernel with ($\theta = [1, 0.2]$), data set is $\{x = [-0.5, 0.33, 0.66]; f = [0, 0.5, 0.5]\}$.

Figure 2.3: Prediction in the case of noiseless observations. The solid black line defines the mean function, blue region defines 95% confidence interval (2σ) distance away from the mean. The dashed lines represent three functions drawn at random from a GP posterior. We can observe that Bayes Theorem eliminates all the functions that do not pass through the observed data-set.

2.2.2 Posterior with Noisy observations

If we assume a more general case of noisy observations, then the measured outputs can be written as:

$$y(x) = f(x) + \epsilon \tag{2.11}$$

Such that ϵ is an independent random noise sampled from a white noise Gaussian $\mathcal{N}(0, \sigma_n^2)$. We can thus write the prior GP of the noisy case as:

$$\Pr[y(x) \mid X, \sigma_n] = GP(0, k(x_1, x_2) + \sigma_n^2 \delta_{xx'}) \tag{2.12}$$

Here, $\delta_{xx'}$ is a Kronecker delta function, which is one iff $x = x'$ and zero otherwise. Since the noise is independent for each observation, there is no noise term for covariances across inputs. The joint distribution of the training outputs $Y(X)$ and true physical process $f(X_*)$ according to the above prior becomes:

$$\Pr\begin{bmatrix} Y(X) \\ f(X_*) \end{bmatrix} = \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} K(X,X) + \sigma_n^2 I & K(X, X_*) \\ K(X_*, X) & K(X_*, X_*) \end{bmatrix}\right) \tag{2.13}$$

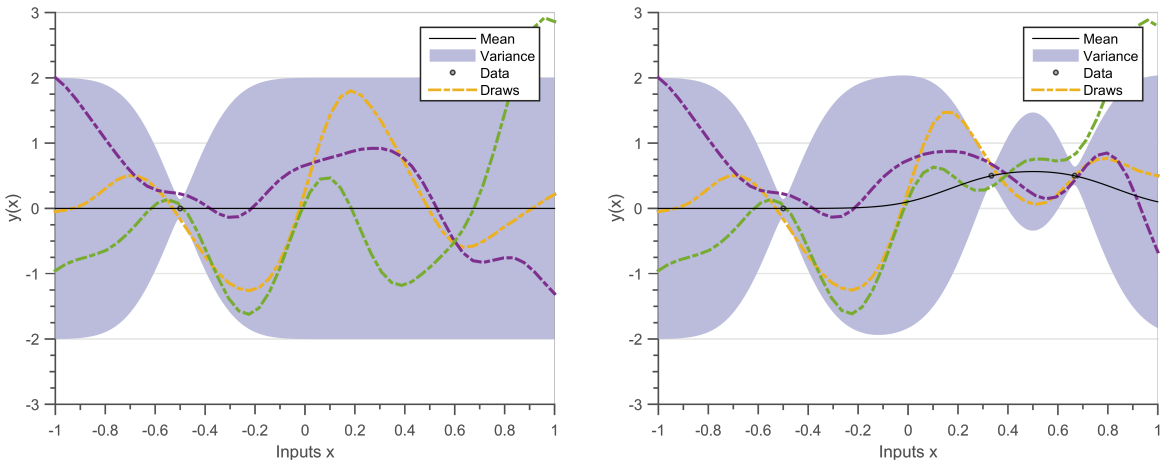
The difference between equation 2.13 and 2.9 is the addition of noise term $\sigma_n^2 I$. Since the noise is assumed to be independent, it is multiplied to an identity matrix. To know how to add more complex noise models please refer to chapter 4. The posterior distribution of $f(X_*)$ can be calculated as:

$$\begin{aligned} \Pr(f(X_*) \mid X_*, X, Y(X)) = GP(&K(X_*, X)(K(X,X) + \sigma_n^2 I)^{-1} f(X), \\ &K(X_*, X_*) - K(X_*, X)(K(X,X) + \sigma_n^2 I)^{-1} K(X, X_*) \end{aligned} \tag{2.14}$$

Figure 2.4 shows the posterior GP after adding observed data into the initial prior. The solid black line defines the mean function, blue region defines 95% confidence interval (2σ) distance away from the mean. The dashed lines represent three functions drawn at random from a GP posterior. Random functions can be sampled from the posterior distribution as described in the earlier section 2.1. Due to the inclusion of noise in the prior, we see that the draws from posterior are not necessarily passing from the observed point.

2.2.3 Interpretation of posterior

We will now introduce a short hand notation and replace the lengthy notation $K(X, X)$ with K_{XX} and $K(X, X_*)$ with K_{XX_*}. For the case where we have only one test point x_*,



(a) Posterior distribution for the case of noisy observations. Prior is a GP with mean zero, covariance as SE Kernel with ($\theta = [1, 0.2]$) and noise as $\sigma_n = [0.02]$), , data set is $\{x = -0.5; f = 0\}$.

(b) Posterior distribution for the case of noisy observations. Prior is a GP with mean zero, covariance as SE Kernel with ($\theta = [1, 0.2]$) and noise as $\sigma_n = [0.02]$), , data set is $\{x = [-0.5, 0.33, 0.66]; f = [0, 0.5, 0.5]\}$.

Figure 2.4: Prediction in the case of noisy observations. The solid black line defines the mean function, blue region defines 95% confidence interval (2σ) distance away from the mean. The dashed lines represent three functions drawn at random from a GP posterior. The mean and the draws do not pass exactly from the observation point.

we can write the predictive mean and variance in short-hand as:

$$\mathbf{E}[f(x_*)] = K_{Xx_*}^T (K_{XX} + \sigma_n^2 I)^{-1} Y \tag{2.15}$$

$$Cov[f(x_*)] = K_{x_*x_*} - K_{Xx_*}^T (K_{XX} + \sigma_n^2 I)^{-1} K_{Xx_*} \tag{2.16}$$

Precision Matrix Both the predictive mean (equation 2.15) and predictive covariance (equation 2.16) need inverse of the covariance matrix $(K_{XX} + \sigma_n^2 I)^{-1}$. The inverse of a covariance matrix is also known as a precision matrix. While the elements of a covariance matrix capture the variance and correlation information, a precision matrix contains the conditional dependence information [MacKay 2003]. Thus, if the $(i, j)^{th}$ element of a precision matrix is zero, the i^{th} and j^{th} random variables are conditionally independent.

Calculating the precision matrix is an $\mathcal{O}(N^3)$ operation for a covariance matrix of size N. After $N \sim 10,000$ a normal computer runs out of RAM, and thus cannot perform the inversion. Fortunately, there exist several approximations to efficiently inverse the covariance matrix and perform predictions, details are available in section 3.

Predicted mean The predictive mean is a linear combination of the observations y_i, and has participation-factor of $K_{Xx_*}^T(K_{XX} + \sigma_n^2 I)^{-1}$. Since a SE kernel $K_{Xx_*}^T$ decreases exponentially with distance, observations closer to x_* have more impact on the final prediction (equation 2.17).

$$\mathbf{E}[f(x_*)] = \sum_{i=1}^{N} K_{Xx_*}^T(K_{XX} + \sigma_n^2 I)^{-1} y_i \tag{2.17}$$

The predictive mean can also be interpreted as a linear combination of the basis functions $K_{x_i x_*}$, and participation factors $(K_{XX} + \sigma_n^2 I)^{-1} Y$ (equation 2.18).

$$\mathbf{E}[f(x_*)] = \sum_{i=1}^{N} K_{x_i x_*}(K_{XX} + \sigma_n^2 I)^{-1} Y \tag{2.18}$$

This means that even though a GP represents an infinite-dimensional vector (function), to predict the mean we only care about the N dimensional multivariate Gaussian (section 2.13). If the precision matrix is cached, then calculating the mean is an $\mathcal{O}(N)$ operation.

Predicted variance The predictive variance is a combination of two terms, $K_{x_* x_*}$ which is the variance due to prior assumptions, and $-K_{Xx_*}^T(K_{XX} + \sigma_n^2 I)^{-1} K_{Xx_*}$, which denotes the decrease in variance due to observations. The predictive distribution of test targets $y(x_*)$ can be calculated by adding a noise term σ_n^2 in predictive covariance equation 2.16.

$$Cov[y(x_*)] = K_{x_* x_*} - K_{Xx_*}^T(K_{XX} + \sigma_n^2 I)^{-1} K_{Xx_*} + \sigma_n^2 \tag{2.19}$$

We observe that the predicted variance in not dependent on the observations y, this is one of the flaws in GP regression. Since the assumption that the dataset $(\mathcal{D})$ comes from a GP might not necessarily be true, the predicted variance can poorly represent the model error. Hence, predicted variance is not necessarily a measure of model error but an efficient method to track uncertainties arising from the prior assumption and non-continuous observations [Shah 2014].

The mean and variance are highly dependent on the kernel hyper-parameters. In order to automatically learn the hyper-parameters, we must perform model selection. Section 2.3 details how to fine-tune hyper-parameters to find an optimal prediction.

2.3 Choosing Hyper-parameters

Since the properties of functions under a GP are controlled by the functional form of the covariance kernel and its hyper-parameters, model selection amounts to choosing a

functional form and learning the hyper-parameters θ from data. In this section we discuss how to select an optimal model by tuning hyper-parameters for a given covariance function. Please refer to chapter **??** for discussion on how to choose covariance functions.

We define a new dataset $\mathcal{D}_2$ which will be used to compare predictions using different hyper-parameters. The function $f(x)$ (equation 2.20) is evaluated at 20 equidistant points between $x \in [-1, 1]$,and are corrupted by an independent white noise having variance $\sigma_{noise} = 0.1$. Matlab code 2.6 is a sample code to generate the dataset $\mathcal{D}_2$.

$$f(x) = \frac{sin(5\pi x)}{5\pi x} \tag{2.20}$$

```
nData = 20; % number of data points

f = @(x)sin(5*pi*x)./(5*pi*x); % Function

noise = 0.1; % Noise in dataset
xData = linspace(-1, 1, nData)';
yData = f(xData) + noise^2*rand(nData, 1);
```

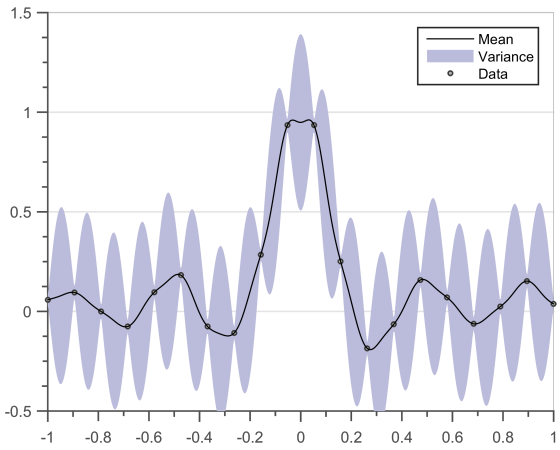
Matlab Code 2.6: Code for dataset D2

Figure 2.5 demonstrates that choosing optimal hyper-parameters is very vital for accurate prediction. It compares the posterior distributions obtained for SE priors with two different hyper-parameters. We observe that the mean of figure 2.5(a) passes through all the observed data points but is more complex. The mean in figure 2.5(b) is a smooth function but does not fit the data properly.

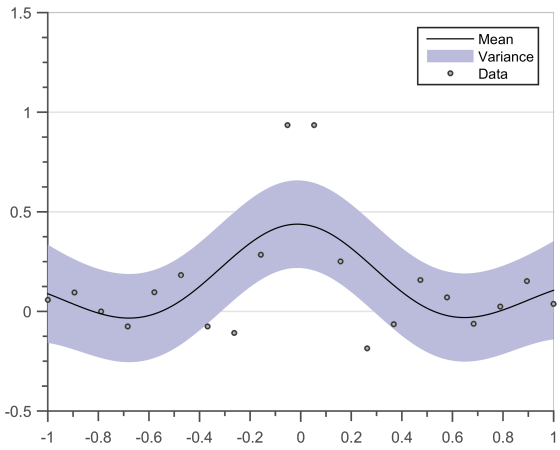
In a pure Bayesian framework we should put a prior over our hyper-parameters $\Pr[\theta]$ and use Bayes Rule to estimate the posterior $\Pr[\theta \mid \mathcal{D}]$ over our data set (just ike we didn in section 1.3). However, this approach becomes intractable and several sampling schemes have been proposed to calculate the posterior of hyper-parameters [Osborne 2010, Neal 2011].

Another method to find the optimal hyper-parameters is by performing Cross-Validation (CV). CV procedure is to split the experimental design set into two disjoint sets, one is used for training and the other one is used to monitor the performance of the surrogate model. A particular case of CV is the Leave-One-Out (LOO) where test sets are obtained by removing one observation at-a-time [Rasmussen 2006, Dubrule 1983, Le Gratiet 2013].

In this manuscript we neither put a prior over our hyper-parameters nor use LOO-CV for choosing hyper-parameters. We use the marginal likelihood, also called evidence, to find optimal hyper-parameters [MacKay 2003]. The probability of generating the observations (Y) at the points (X) from a prior (defined by $k(x_1, x_2, \theta)$) is called the marginal likelihood



(a) Posterior between SE prior with hyper-parameters ($\theta = [0.35, 0.05]; \sigma_{noise} = 0.01$) and data.



(b) Posterior between SE prior with hyper-parameters ($\theta = [0.35, 0.5]; \sigma_{noise} = 0.01$) and data.

Figure 2.5: Posteriors for 2 different sets of hyper-parameters. Solid black line defines the mean function, blue region defines 95% confidence interval (2σ) distance away from mean.

$\Pr[Y \mid X, \theta]$. In other words, marginal likelihood is the probability that our data set $\mathcal{D}$ was generated from a particular prior. Hence, when we maximize a marginal likelihood we are finding the best prior that could generate our data set. Using equation 2.12 and 2.13 we get:

$$\begin{aligned} \Pr[Y(X) \mid X, \theta, \sigma_n] &= \mathcal{N}(0, K(X, X') + \sigma_n^2 I) \\ &= \frac{1}{\sqrt{(2\pi)^{N/2} K_{XX}}} exp^{-\frac{1}{2} Y^T K_{XX} Y} \end{aligned} \tag{2.21}$$

Directly maximizing the marginal likelihood with respect to the hyper-parameters can be inefficient. This is because the marginal likelihood does not vary significantly with the hyper-parameters. Hence to speed up the optimization process we generally maximize the log of marginal likelihood.

$$\log(\Pr[Y \mid X, \theta]) = -\frac{1}{2} Y^T [K_{XX} + \sigma_{noise}^2 I]^{-1} Y - \log \left| K_{XX} + \sigma_{noise}^2 I \right| - \frac{n}{2} \log(2\pi) \tag{2.22}$$

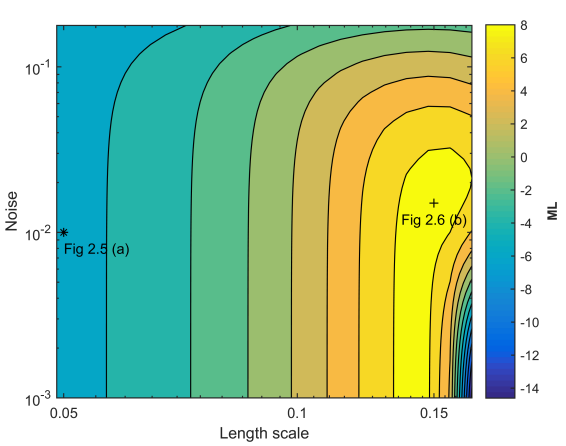
The marginal likelihood is a trade-off between a data-fit term ($\frac{1}{2} Y^T K_{XX}^{-1} Y$) and a model complexity term ($\log |K_{XX}|$). The optimization of log marginal likelihood provides the best compromise in terms of explaining the existing data set $\{(x_i, y_i)\}$ and the initial assumptions encoded in the prior. The Matlab code 2.7 defines the function 'log Marginal Likelihood' (equation 2.22), which is maximized later in the code.

```
function logMarginalLikelihood = logMarginalLikelihood(
   theta, covarianceFunction, x, y)
N = length(x);

% Gram Matrix
noiseTerm = eye(N)*theta(3)^2;
gramMatrixXX = evaluateGramMatrix(covarianceFunction, theta
   , x, x) + noiseTerm;

% The data fit term
dataFitTerm = -1*(0.5)*y'*inv(gramMatrixXX)*y;

% The complexity term
jitter = 10^(-6);
L = chol(gramMatrixXX + eye(N)*jitter);
complexityTerm = -1*sum(log(diag(L)));

% Normalization term
normalizationTerm = -1*N*log(2*pi)/2;
logMarginalLikelihood = dataFitTerm + complexityTerm +
   normalizationTerm;

end

% Optimizing the log marginal likelihood
theta = [1, 0.2, 0.1];
% Amplitude hyp = 1
% Length Scale = 0.2
% Noise  = 0.1

options = optimoptions('fminunc','GradObj','off', 'MaxIter'
   , 100); % indicate gradient is provided
optimizedTheta = fminunc(@(x) -1*logMarginalLikelihood(x,
   covarianceFunction, xData, yData), theta, options);
```

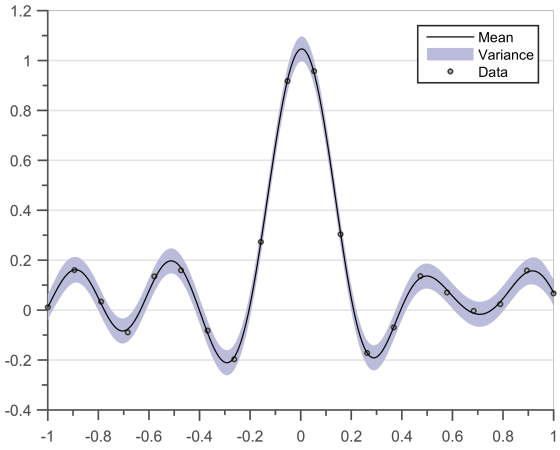
Matlab Code 2.7: Optimizing the Log Marginal Likelihood

Figure 2.6(a) shows the contours of marginal likelihood with respect to length-scale $\theta_{lengthScale}$ and noise σ_n hyper-parameters. The data set is same as used in figure 2.5 and the prior is a zero mean with SE kernel. Figure 2.6(b) shows the posterior for same data set as used in figure 2.5 but for the hyper-parameters where marginal likelihood is maximum.

The marginal likelihood could have multiple maximas in the space of hyper-parameters, hence care should be taken while initializing hyper-parameters for optimizing the log of marginal likelihood.



(a) Marginal likelihood contours for varying noise and length-scale parameter. The amplitude hyper parameter is ($\theta_{amplitude} = [0.35]$). Also shown on the figure are locations of hyper-parameters for figures 2.5(a) and 2.6(b).



(b) Posterior between SE prior with optimized hyper-parameters ($\theta = [0.35, 0.15]; \sigma_{noise} = 0.015$) and data. $\log(ML) = 8.04$. Solid black line defines the mean function, blue region defines 95% confidence interval (2σ) distance away from mean.

Figure 2.6: Maximizing marginal likelihood

2.4 Discussion

In this chapter we provide a brief introduction on how to perform Regression with GPs. GPs are the ideal candidate for regression due to their marginalization property which makes them computationally tractable. Even if GPs define an infinite dimensional random vector, inference on a few points does not require the presence of infinitely other points. This makes drawing functions, calculating posterior distribution, and automating selection of hyper-parameters computationally feasible. Thereby, making GPs an ideal candidate for defining a Prior distribution in a Bayesian Regression framework.

The section 2.1 details the key components of the GPs. A GP can be completely parametrized by its mean and covariance function. While the trend of a GP is defined by its mean function, the structure of its constituent functions is defined by the covariance function. The mean of a GP can be assumed to be zero, since an extra term in the covariance function can represent the mean function. Hence the problem of learning in a GP is exactly the problem of finding suitable properties of the covariance function (subsection

2.1.3). Once a function form of covariance is chosen, we can calculate the Gram matrix at desired points and use it to draw random functions from our prior (subsection 2.1.4).

The next section 2.2 describes how to calculate the posterior distribution. The posterior is the conditional distribution ($\Pr[f(x_*) \mid Y, X, \theta]$) for an assumed Prior distribution ($\Pr[f] = GP(0, k(x_1, x_2, \theta))$ and a set of observed data points ($\mathcal{D} = X, Y$). Due to the Gaussian assumption, the conditional probabilities are all computationally tractable and can be calculated using a few matrix operations, side-stepping the computational burden of performing iterative sampling. Calculating the posterior is easy both in the absence (subsection 2.2.1)and presence (subsection 2.2.2) of noise in observations.

Given a functional form of the covariance, section 2.3 shows the importance of choosing the correct hyper-parameters. In a pure Bayesian framework the posterior distribution of the hyper-parameters should be calculated ($\Pr[\theta \mid \mathcal{D}]$), but this is computationally intractable, needing several iterations for calculation of integrals. A common practice in the community is maximizing the marginal likelihood to automatically choose the hyper-parameters. Marginal likelihood is the probability of a prior distribution $\Pr[f] = GP(0, k(x_1, x_2, \theta)$ generating the observations $\mathcal{D}$. Hence maximizing the marginal likelihood gives the optimal set of hyper-parameters for a functional form of covariance function (figure 2.6).

Calculating the precision matrix $[K_{XX} + \sigma^2_{noise} I]^{-1}$ is an important task in calculating the marginal likelihood (equation **??**), posterior mean (equation **??**) and covariance (equation **??**). Unfortunately, this task has a computational complexity of $\mathcal{O}(N^3)$ and memory footprint of $\mathcal{O}(N^2)$. This puts an upper limit of $N \sim 10^4$ on the number of data points, a standard laptop cannot store such a big matrix for inversion [6]. The next chapter describes few methods of performing approximating inference which scales GPs to $N \sim 10^6$ or more data points.

[6] The computer runs out of memory before we run out of patience :p

Chapter 3

Scaling up Gaussian Process Regression

The GP regression approach, as mentioned in earlier chapter, is intractable for large data sets. For a data set of size N the covariance matrix K_{XX} is of size $N \times N$, where $\mathcal{O}(N^3)$ time is needed for calculating the precision matrix and $\mathcal{O}(N^2)$ memory for storage. Since inverting the covariance matrix takes considerable amount of time and memory, almost all techniques to scale up GP regression try to approximate the inversion of Gram matrix K_{XX}.

Let us take the example of a SE kernel, for a large value of length-scale, the Gram matrix (K_{XX}) is spread out and has a rank lower than N (figure 2.2(b)). Due to this characteristic, the Gram matrix can be approximated using low-rank approximations, reducing the cost of inverting the Gram matrix. In the GP literature, sparse approximations (section 3.1) use a set of inducing points to compress the information of the several observations through the low-rank approximation.

For the same SE kernel, if the length-scale tends to a low value, the Gram matrix is not of low-rank but tends to a diagonal matrix (figure 2.2(a)). In the GP literature the mixture of experts (section 3.2) methodology exploits the block diagonal nature of the Gram matrix by distributing data points into a subset of experts, assuming independence across experts and distributing the calculations into several batches. The first regime suggests global (numerical) low-rank approximations while the second regime suggests local block-diagonal approximations [March 2015, Chenhan 2016].

The remaining chapter unfolds as follows, section 3.1 describes the Sparse Approximations detailing several methods of choosing inducing points and then performing experiments on a toy dataset. While, section 3.2 describes the Distributed GP methodology detailing several methods for merging of experts and then performing experiments on the same toy-dataset.

3.1 Sparse Approximations

Sparse methods use a small subset of input points as support or inducing points to approximate the Gram matrix. Suppose we use M inducing points $X^m = \{x_1^m; x_2^m; \ldots; x_M^m\}$, such that $M < N$. The points X^m can be a subset of training inputs in the input space.

3.1.1 Nyström Approximation

Using Nyström approximation the Gram matrix can be approximated as equation 3.1 [Quiñonero-Candela 2005, Seeger 2003], for more detail refer to appendix **refer to link in the appendix**.

$$K_{nystorm}(X, X) = K(X, X^m)K(X^m, X^m)^{-1}K(X^m, X) \tag{3.1}$$

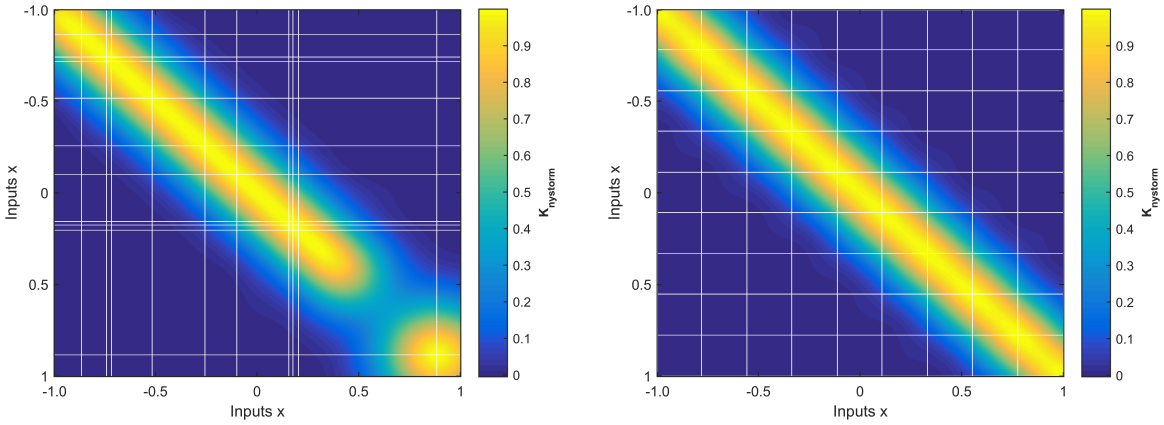
Here, $K(X^m, X^m)$ is a $M \times M$ Gram matrix evaluated at inducing points X^m, $K(X, X^m)$ is an $N \times M$ Gram matrix between training points and inducing points. The inversion of approximate matrix takes $\mathcal{O}(NM^2)$ time to compute. The code 3.1 defines a function 'evaluateNystromGramMatrix' which is used to evaluate the Nström approximation of the gam matrix.

```
% Function to evaluate the approximate gram matrix
function gramMatrix = evaluateNystromGramMatrix(
   covarianceFunction, theta, xm, x1, x2)

        gramMatrixX1M = evaluateGramMatrix(
           covarianceFunction, theta, x1, xm);
        gramMatrixMM = evaluateGramMatrix(
           covarianceFunction, theta, xm, xm);
        gramMatrixMX2 = evaluateGramMatrix(
           covarianceFunction, theta, xm, x2);

        gramMatrix = gramMatrixX1M*inv(gramMatrixMM)*
           gramMatrixMX2;
end
% Test points
nStar = 1000;
xStar = linspace(-1, 1, nStar);

% Inducing points
```

```
nInducing = 20;
inducingPoints = linspace(-1, 1, nInducing);

approximateGramMatrix = evaluateNystromGramMatrix(
   covarianceFunction, theta, inducingPoints, xStar, xStar)
   ;
```

Matlab Code 3.1: Gram Matrix using Nyström Approximation

Figure 3.1(a) is an approximate Gram matrix using Nyström approximation of the matrix in figure 2.1(a) at the input points $X^* = \{[0 : 0.02 : 1]\}$. The inducing points X^m are chosen randomly from the set of input points and their location is denoted by white lines. We can observe that if the gap between inducing points increases, the accuracy of Gram matrix degrades (eg. at $x \sim 0.5$). Figure 3.1(b) is again an approximated Gram matrix using Nyström approximation of the matrix in figure 2.1(a). This time the equally spaced inducing points are chosen in the range of X^*. Notice the significant improvement in the Gram matrix upon different set of inducing inputs.



(a) Approximated Gram matrix using Nyström approximation for a SE Kernel with ($\theta = [1, 0.2]$) (figure 2.1(a)) at the input points $X^* = \{[0 : 0.02 : 1]\}$. The inducing points were chosen randomly, the white lines denote the location of inducing points.

(b) Approximated Gram matrix using Nyström approximation for a SE Kernel with ($\theta = [1, 0.2]$) (figure 2.1(a)) at the input points $X^* = \{[0 : 0.02 : 1]\}$. The white lines denote the location of inducing points, the inducing points are uniformly distributed. Notice the significant improvement in Gram matrix due to different inducing inputs

Figure 3.1: Approximate Gram matrix for a SE kernel using Nyström approximation.

Later, [Snelson 2006] proposed the FITC approach which corrects the diagonal terms of the Gram matrix and improves the prediction capabilities (equation 3.1).

$$K_{FITC}(X, X) = diag[K(X, X) - K_{nystorm}(X, X)] + K_{nystorm}(X, X) \tag{3.2}$$

Note, calculating $diag(K(X, X))$ is an $\mathcal{O}(N)$ operation and thus does not significantly impact the time taken.

The posterior distribution for the approximate prior can be derived similarly as given in appendix **add appendix ref** and is a Gaussian. The predictive mean, and predictive variance are written as equation 3.3 and equation 3.4. Here, $K_{approximate}(X, X')$ can be the approximated Gram matrix either from the Nyström approximation (equation 3.1) or the FITC approximation (equation 3.2).

$$\mathbf{E}[f_{approximate}(x_*)] = K_{Xx_*}^T (K_{approximate}(X, X') + \sigma_n^2 I)^{-1} Y \tag{3.3}$$

$$Cov[f_{approximate}(x_*)] = K_{x_*x_*} - K_{Xx_*}^T (K_{approximate}(X, X') + \sigma_n^2 I)^{-1} K_{Xx_*} \tag{3.4}$$

By approximating the $K(X, X)$ using inducing points, we have effectively changed the GP prior. This means that X^m have also become the hyper-parameters of our GP. Hence we should fine-tune locations of X^m and the hyper-parameters θ to obtain a good prediction of our data. The marginal likelihood for the approximate prior (equation 3.5) be written similarly as equation 2.21.

$$\Pr[Y(X) \mid X, X^m, \theta, \sigma_n] = \mathcal{N}(0, K_{approximate}(X, X') + \sigma_n^2 I) \tag{3.5}$$

The maximization of the marginal likelihood in equation 3.5 with respect to $(X^m; \theta)$, is prone to over-fitting especially when the number of inducing inputs is large. This means that if we keep on increasing the number of inducing points, a time will come when we will tend to decrease the accuracy of our predictions on the test data set. The variational approximation (detailed next) approach overcomes this issue of over-fitting by adding a regularization term penalizing over-fitting.

3.1.2 Variational Approximation

The variational approximation does not attempt to approximate the Gram matrix. Instead, it assumes a probability distribution $q(f)$ of the true posterior distribution $p(f \mid y)$ and minimizes the distance between the two [Titsias 2009].

The $q(f)$ is written in terms of inducing points (X^m) and the KL divergence $KL(q||p)$[1] is minimized between the variational distribution q and true distribution p. When we minimize the KL divergence, we are making the assumed distribution closer to true distribution and hence improving the values of (X^m) and θ. This minimization of KL divergence is

[1]KL divergence is a measure of distance between two probability distributions

equivalently expressed as the maximization of the equation 3.6

$$F_V = log(\mathcal{N}[0, \sigma_n^2 I + K_{nystorm}(X, X)]) - \frac{1}{2\sigma_n^2} Tr(K_{XX} - K_{nystorm}(X, X)) \tag{3.6}$$

Notice the similarity between equation 3.6 and 3.1. The novelty of the above objective function is that it contains a regularization term: $-\frac{1}{2\sigma^2} Tr(K_{XX} - K_{nystorm}(X, X))$. Thus, F_V attempts to maximize the marginal likelihood as derived for Nyström approximation and simultaneously minimizes the trace. When the regularization term tends to zero ($K_{XX} = K_{nystorm}(X, X)$), which means that the inducing variables can exactly reproduce the true Gram Matrix.

The posterior distribution for variational inference approximation is same as the one derived for Nyström approximation. The difference between Nyström approximation and variational approximation is the addition of regularization parameter ($-\frac{1}{2\sigma^2} Tr(K_{XX} - K_{nystorm}(X, X))$) while optimizing X^m and θ, this additional term reduces over-fitting.

3.1.3 Experiments

In this section, we conduct experiments on a toy-data set to observe the accuracy of Nyström approximation for varying number and location of inducing points. The basic toolbox used for this paper is GPML provided with [Rasmussen 2006] on MATLAB 2014b. All experiments were performed on an Intel quad-core processor with 4Gb RAM.

10-fold Cross Validation (CV) will be used to assess the performance of the prediction. CV is a technique where the dataset is partitioned as the test set and training set. A model is learned using the training set and Root Mean Square Error (RMSE) is calculated between the prediction and test set as a measure of accuracy. In the 10-fold version of CV, the dataset will be randomly partitioned into 10 subsets containing an equal number of points. Of the 10 subsets, a single subset is retained as the test dataset, and the remaining 9 (10 - 1) subsets are used as training data. The cross-validation process is then repeated 10 times (the folds), with each of the k subsets used exactly once as the validation data.

The toy data set ($\mathcal{D}_3$) was generated at 1000 input points $X = \{[-1 : 0.002 : 1]\}$ by sampling a random function from a GP[2] with zero mean, SE covariance function ($\theta = [1, 0.1]$) and noise $\sigma_n = 0.3$. Code 3.2 generates the dataset $\mathcal{D}_3$.

```
%% Generating the toy Dataset 3
theta = [1, 0.1];
noiseHyp = 0.3;
```

[2] ($\Pr[Y \mid X, \theta, \sigma_n] = GP(0, K_{SE}(X, X', \theta = [1, 0.1]) + (0.3)^2 I$)

```
nData = 1000;
xData = linspace(-1, 1, nStar)';

kSENoiseFree = evaluateGramMatrix(covarianceFunction, theta
   , xStar, xStar);
kSENoisy = kSENoiseFree + exp(2*noiseHyp)*eye(nStar);

L = chol(kSENoisy);
yData = L'*rand(nStar, 1);%
plot(xData, yData, '.')
```

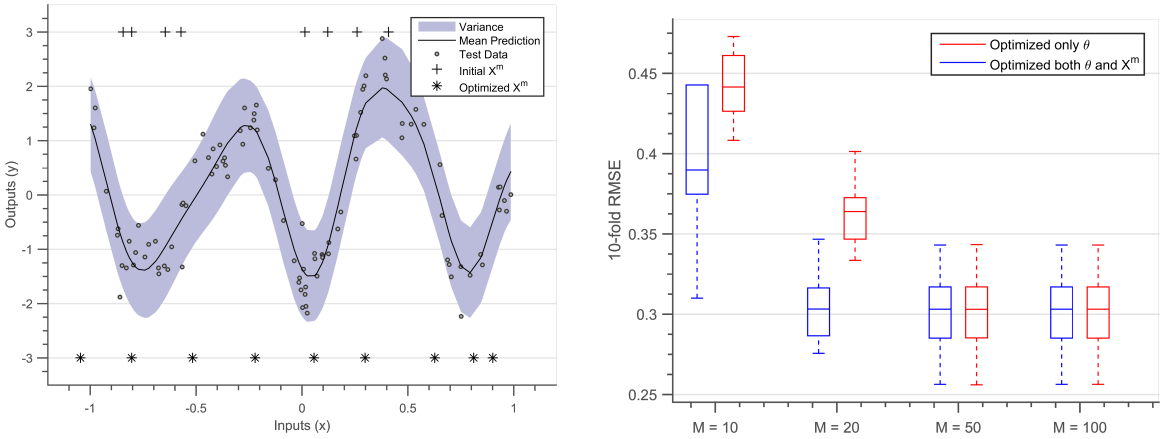
Matlab Code 3.2: Code for toy dataset 3

Figure 3.2(a) is the prediction of the GP obtained after Nyström approximation using 10 inducing points. The solid black line defines the mean function, blue region defines 95% confidence interval (2σ) distance away from the mean. The points denoted by '+' sign are initial locations of inducing points, while the points denoted by '*' sign are locations of inducing points after optimization. The points denoted by '.' are the test points for this fold of the 10-fold CV.

Notice the inducing points, while initially randomly distributed are later uniformly distributed due to optimization of marginal likelihood. In this case, since the training points are randomly distributed, uniformly distributed inducing inputs are better approximations of the Gram matrix. In cases where training data set tends to be dense in one region and sparse in another region, lesser inducing points get allocated at the dense region and more get allocated at the sparse region. This happens because the information contained in a dense cluster of data-points can be approximated by a fewer data points (points in a neighbourhood are similar (section 2.1.3) [Snelson 2006].

Figure 3.2(b) are 10-fold RMSE box-plots for varying number of inducing points. The box-plots in red are cases when only the hyper-parameters were optimized while inducing inputs were distributed randomly.The box-plots in blue are the cases when both locations of inducing points and hyper-parameters are optimized. The accuracy of prediction improves with increasing number of inducing points. Accuracy is better when both locations of inducing points and hyper-parameters are optimized. Since the noise in the generated toy-data is $\sigma_n = 0.3$, it is the best achievable RMSE value. Models constructed when optimizing both θ, X^m reach this RMSE limit for $M = 20$. After $M = 50$, accuracy is similar for both the optimization routines. As a thumb rule, if $M = \frac{N}{10}$, then randomly distributing the inducing points and optimizing θ will be sufficient to give a good prediction [Cao 2013].

Global low-rank approximations are best suited for the case of spread out Gram matrices



(a) Posterior between a Nystorm approximated SE prior with 10 inducing inputs and training data. The solid black line defines the mean function, blue region defines 95% confidence interval (2σ) distance away from the mean. The points denoted by '+' sign are initial locations of inducing points, while the points denoted by '*' sign are locations of inducing points after optimization.

(b) 10-fold RMSE box-plots for varying number of inducing points. The box-plots in red are cases when only the hyper-parameters θ were optimized while inducing inputs were distributed randomly. The box-plots in blue are the cases when both locations of inducing points X^m and hyper-parameters θ are optimized.

Figure 3.2: Results of Nyström Approximation on a toy-data set of size $N = 1000$

(example high length-scale SE priors). We have seen three types of low-rank approximation algorithms in this section. While Nyström and FITC approximations are the simplest method to approximate Gram matrix, finding optimal locations of the inducing points can often lead to over-fitting. We then look at variational approximation procedure which adds a regularization term while finding inducing points, thereby, penalizing over-fitting. The lower computational cost due to sparse approximations, scales sparse GPs to the training set of sizes $N \sim \mathcal{O}(5 \times 10^5)$. [Gal 2014] propose a distributed architecture for scaling variational sparse GPs. In the next section we look at how to approximate Gram matrices using mixture of experts, this enables us to massively scale GPs to sizes $N > \mathcal{O}(10^6)$ by exploiting distributed architecture.

3.2 Distributed Gaussian Process

In the year 2006 Netflix organized a machine learning competition, to create a recommendation algorithm for its users. Teams from all over the world competed in this competition to make the best video recommendation algorithm. As the competition progressed, teams started figuring out that the accuracy increases upon combining algorithms developed by multiple teams. The winner and the runner-up were model ensembles of hundreds of algorithms. Creating model ensembles (also called mixture of experts) is now a standard

practice in many learning competitions [Bauer 1998].

Mixture of experts in GPs use bagging, where subsets of data are generated, individual GPs are trained on these subsets, and their results are finally combined [Chen 2009]. If the data set is partitioned into $N_{experts}$ subsets, such as $\mathcal{D}^{(i)} = X^{(i)}, Y^{(i)}, i \in 1, \ldots N_{experts}$. Then each subset of data $\mathcal{D}^{(i)}$ learns an individual GP model, which are be combined together to give the final predictions . Due to this independent learning, choosing hyper-parameters, and calculating predictions become easily parallelizable and indifferent to the computational infrastructure.

Initially, these model types were used to highlight local features in the data [Rasmussen 2002]. Later, [Ng 2014] proposed to use the parallel feature to speed up predictions. Instead of learning a different GP for each subset, we tie all the different experts using one single set of hyperparameters. This is equivalent to assuming one single GP for the whole data-set, thus there is no correlation across experts, i.e. the experts are independent of each other. This tying of experts greatly reduces the number of hyper-parameters, acts as a regularization, and inhibits over-fitting. Equation 3.7 denotes an independent GP prior for each expert $\mathcal{D}^{(i)}$ such that the hyper-parameters θ and σ_n are same for all experts.

$$\Pr[y^{(i)} \mid x^{(i)}, \theta, \sigma_n] = GP(0, K(x^{(i)}, x^{(i)'}, \theta) + \sigma_n^2 I) \tag{3.7}$$

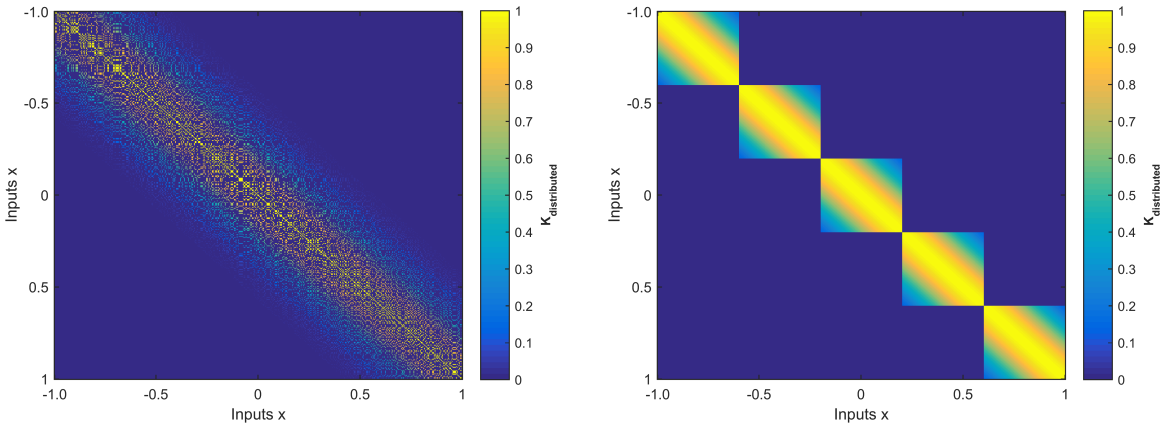
The sample code 3.3 presents a method to randomly clusterize dataset across 6 experts.

```
nStar = 500;
numberOfExperts = 6; % number of experts
sizeOfExperts = floor(nStar/numberOfExperts);

idxRandom = randperm(nStar); % distributing the integers
   1:500 randomly

% Clustering points randomly into experts
for i=1:numberOfExperts
    if i<numberOfExperts
        idxExpert{i} = idxRandom((i-1)*sizeOfExperts+1 : i*
           sizeOfExperts);
    elseif i==numberOfExperts
        idxExpert{i} = idxRandom((i-1)*sizeOfExperts+1 :
           nStar);
    end
end
```

```
% Points in the ith expert
xIthExpert = xData(idxExpert{i});
yIthExpert = yData(idxExpert{i});
```

Matlab Code 3.3: Randomly clustering points into experts

Figure 3.3(a) is an approximate Gram matrix using above approximation, for a SE Kernel with ($\theta = [1, 0.2]$) at the input points $X^* = \{[0 : 0.02 : 1]\}$. 5 experts each having 100 points are chosen, points in the experts are distributed randomly, this gives the approximate Gram matrix the scattered shape. Figure 3.3(b) is an approximate Gram matrix using same approximation, and uniformly distributed experts. 5 experts each having 100 points are chosen, The first expert has first set of 100 points, the second expert has the second set of 100 points and so on. Both the figures are trying to approximate the Gram matrix in of the matrix in figure 2.1(a). The Gram matrix with randomly chosen experts has a more global nature but lacks many high variance regions. The Gram matrix for uniformly chosen experts retains more local features. Inversion of this Gram matrix is an operation of complexity $\mathcal{O}(N_{experts}P^3)$, where P is the number of points in an expert.



(a) Approximated Gram matrix using distributed GP approximation for a SE Kernel with ($\theta = [1, 0.2]$) (figure 2.1(a)) at the input points $X^* = \{[0 : 0.02 : 1]\}$. Points in the experts are distributed randomly, this gives the approximate Gram matrix scattered shape.

(b) Approximated Gram matrix using distributed GP approximation for a SE Kernel with ($\theta = [1, 0.2]$) (figure 2.1(a)) at the input points $X^* = \{[0 : 0.02 : 1]\}$. Points in the experts are distributed uniformly. We can observe that covariance across experts goes to zero.

Figure 3.3: Approximate Gram matrix for a SE kernel using mixture of experts.

Since the experts are independent of each other, we can construct a posterior distribution for each expert (equations 3.8 and 3.9). In the following equations $m^{(i)}(x_*)$ and

$\sigma^{(i)}(x_*)$ are the mean and covariance predictions from expert i at point x_*.

$$m^{(i)}(y(x_*)) = K^T_{x^{(i)}x_*}(K_{x^{(i)},x^{(i)\prime}} + \sigma_n^2 I)^{-1} y^{(i)} \tag{3.8}$$

$$\sigma^{(i)}(y(x_*)) = K_{x_*x_*} - K^T_{x^{(i)}x_*}(K_{x^{(i)},x^{(i)\prime}} + \sigma_n^2 I)^{-1} K_{x^{(i)}x_*} \tag{3.9}$$

3.2.1 Combining experts

There are several methods in the literature, which combine the posterior predictions of individual experts, and give a global estimate. The Product of Experts model, uses the independence assumption between experts, and multiplies the individual posterior distribution[3], but these predictions tend to be overconfident. Another method called the generalized Product of Experts (gPOE), assigns a participation factor to each expert, this is based on the amount of uncertainty in prediction (more confident experts have higher say in prediction)[Cao 2014]. The Bayesian Committee Machine (BCM) imposes the independence assumption between each expert pair using Bayes Rule, but can result in bad predictions when leaving data regime [Tresp 2000].

This thesis will use robust Bayesian Committee Machine (rBCM) model to combine the posterior distributions of experts [Deisenroth 2015]. The rBCM model is an amalgamation of all the above three mentioned methods, it combines the confidence weighting parameter of gPOE, with the Bayesian formulation in BCM technique, to generate the following posterior distributions.

$$Cov(y(x_*))^{-2} = \sum_i \beta_i \sigma^{-2}_{(i)} + (1 - \sum_i \beta_i)(K_{x_*x_*})^{-2} \tag{3.10}$$

$$m(y(x_*)) = (Cov(y(x_*)))^{-2} \sum_i \beta_i (\sigma^{(i)})^{-2} m^{(i)}(x_*) \tag{3.11}$$

$K_{x_*x_*}$ is the auto-covariance of the prior at prediction point x_*. β_k determines the influence of experts on the final predictions [Cao 2014] and is given as $\beta_i = \frac{1}{2}(\log K^2_{x_*x_*} - \log(\sigma^{(i)})^2)$. Experts which are very confident of their predictions at x_* will tend to have low $\sigma^{(i)}$ thereby leading to a higher influence factor β_i.

Due to the independence assumption, the marginal likelihood can be written as a sum of individual likelihoods and then can be optimized to find the best-fit hyperparameters. By approximating the $K(X, X)$ in terms of $\mathcal{D}^{(i)}$ and $N_{experts}$ we have again changed the GP prior. This means that the number of experts $N_{experts}$ and clustering of points also impact

[3] $\Pr[y(x_*) \mid x_*, \mathcal{D}, \theta] \propto \prod \Pr[y^{(i)} \mid x^{(i)}, \mathcal{D}^{(i)}, \theta]$

the prediction capabilities of GP. The below equation 3.12 describes the formulation for marginal likelihood.

$$\log \Pr[y \mid X, \mathcal{D}, \theta] \approx \sum_{k=1}^{N_{experts}} \log \Pr[y^{(i)} \mid X^{(i)}, \theta] \tag{3.12}$$

The sample code 3.4 defines the function 'logMarginalLikelihoodDGP' which calculates the new log marginal likelihood (equation 3.12) using the experts as defined in code 3.3. The optimal hyper-parameters can be obtained after maximizing this function with respect to hyper-parameters.

```
function logMarginalLikelihoodDGP =
   logMarginalLikelihoodDGP(theta, covarianceFunction, x, y
   , idxExpert)

        numberOfExperts = length(idxExpert);

        % Adding log marginal likelihoods of independent
           experts
        logMarginalLikelihoodDGP = 0;
        for i =1:numberOfExperts
                % Points in the iTH expert
                xOfIthExpert = x(idxExpert{i});
                yOfIthExpert = y(idxExpert{i});

                logMarginalLikelihoodDGP =
                   logMarginalLikelihoodDGP + ...
                            logMarginalLikelihood(theta,
                               covarianceFunction,
                               xOfIthExpert, yOfIthExpert);
        end

end

% Hyper parameters
noiseHyp = 0.1;
theta = [1, 0.2, noiseTerm];

logMarginalLikelihoodDGP(theta, covarianceFunction, xData,
```

```
    yData, idxExpert)
```

Matlab Code 3.4: log Marginal Likelihood for Distributed GP

Maximizing the above log-marginal likelihood will give the optimal values of hyper-parameters. Points in the experts can be distributed either randomly, or using a clustering scheme (eg. k-means clustering[4]) for stationary kernels[5] k-means clustering should be preferred. The k-means algorithm clusters points based on a measure of distance, points in separate clusters are far away from each other when compared to points in the same cluster. This means that the covariance (for stationary kernels) between separate clusters is significantly lower when compared to points in same cluster. Hence, the cross-covariance across separate clusters can be more easily assumed to be zero.

3.2.2 Experiments

We again conduct experiments on a toy-data set, this time to observe the accuracy of the distributed GP approximation for varying number of experts.

Again the 10-fold Cross Validation (CV) will be used to assess the performance of the prediction. The same toy-data set as used in section 3.1.3 was used to perform the experiments in this section (1000 data-points from $\Pr[y \mid X, \theta, \sigma_n] = GP(0, K_{SE}(X, X', \theta = [1, 0.1]) + (0.3)^2 I)$.

Figure 3.4(a) is the prediction of the GP obtained after distributed approximation using 9 experts and k-means clustering. The solid black line defines the mean function, blue region defines 95% confidence interval (2σ) distance away from the mean. The colored points in the points denoted by '*' at the bottom show how different points are distributed across experts, similar colored points belong to one expert. The data denoted by '.' is the test data for one fold of the 10-fold CV.

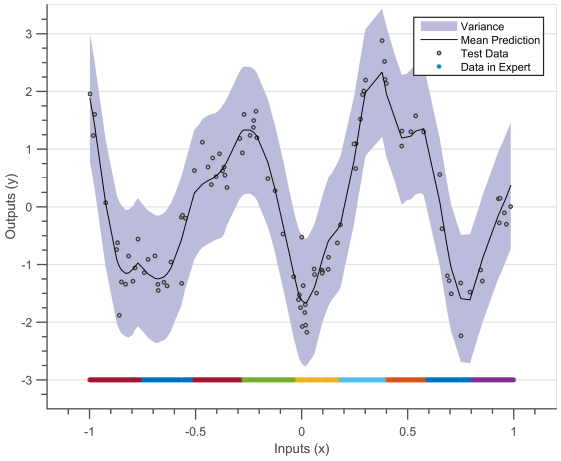
The points across experts are uniformly distributed, as can be observed by the coloring scheme. Since the training points are almost uniformly distributed, the k-means algorithm will cluster the points uniformly. Actually, the training and test set used in figures 3.4(a) and 3.2(a) are same. Notice how Nyström approximation has a global smooth shape while the distributed GP approximation retains the local features of the data set.

Figure 3.4(b) are 10-fold RMSE box-plots for different number of P. The box-plots in red are cases when the points are distributed using k-means clustering. The box-plots in blue are the cases when points are distributed randomly. The accuracy of prediction improves with increasing number of points in an expert. Note, the noise in the generated

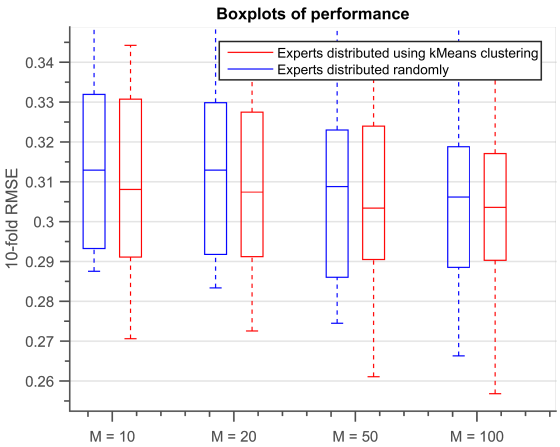
[4]k-means algorithm clusters close by points in one cluster. The notion of closeness is defined by some measure of distance

[5]stationary kernels are only a function of $\tau = |x - x'|$

toy-data is $\sigma_n = 0.3$, it is the best achievable RMSE value. Accuracy is slightly better when experts are distributed using k-means clustering, both being very close to the 0.3 RMSE limit. As a thumb-rule, setting $P = N/10$, and optimizing the hyper-parameters (θ) is a good enough approximation.



(a) Prediction of the GP obtained after distributed approximation using 9 experts and k-means clustering. The solid black line defines the mean function, blue region defines 95% confidence interval (2σ) distance away from the mean. The colored points in the points denoted by '*' at the bottom show how different points are distributed across experts, similar colored points belong to one expert. The data denoted by '.' is the test data for one fold of the 10-fold CV.



(b) 10-fold RMSE box-plots for different number of points across. The box-plots in red are cases when only the hyper-parameters when the points are distributed using k-means clustering. The box-plots in blue are the cases when points are distributed randomly. The accuracy of prediction improves with increasing number of points in an expert

Figure 3.4: Results of distributed GP Approximation on a toy-data set of size $N = 1000$

3.3 Discussion

The calculation of posterior in GPs becomes computationally intractable for large data sets. Calculating the precision matrix is an operation of computational complexity $\mathcal{O}(N^3)$, putting a limit of $N \sim 10^4$ data points for model building. This chapter describes the state of art for scaling up GPs for regression tasks. There exist two methods to scale up GPs, the first called sparse methods, they use a set of inducing inputs to reduce the computational cost of calculating the precision matrix. The second called distributed GP they divide the data set into smaller subsets called experts, distributing the model building into several batches.

Sparse methods use Nyström approximation rewriting the Gram matrix as equation 2.1.4, thereby reducing the computational complexity to $\mathcal{O}(NM^2)$ (M « N), M being the

number of inducing points. Through experiments on a toy-dataset, it can be shown that we can set $M \sim N/10$ when inducing points are randomly distributed, and $M \sim N/50$ when the locations of inducing points are optimized. This approximation pushes the limit of GP Regression to $N \sim 10^6$ data points. Distributed GPs distribute the GP Regression tasks into several batches, thereby reducing the computational complexity to $\mathcal{O}(NP^3)$ (P«N), P being the number of points in an expert. Through experiments on a toy-dataset we demonstrate that $P \sim N/100$ does not effects the regression task significantly. In fact we can further reduce P if we enable repetition of points between experts. This enables to scale GPs to any number of data-points, we will demonstrate this by running a GP regression on millions of data-points in this manuscript (section 5).

There are several reasons why GPs should be preferred to perform regression tasks. GPs provide a probabilistic framework to define a family of functions, while the covariance functions allows to incorporate a wide range of assumptions (part II). GPs are computationally tractable, given a covariance function and observations, the predictive distribution can be calculated exactly. By providing a closed form expression of marginal likelihood GPs provide a powerful method to automatically select hyperparameters. Although GPs suffer in presence of large data sets, there exist several approximate methods to scale GPs to millions of data points.

Part II

Incorporating structure in Gaussian Process Regression

Chapter 4

Basic Covariance Functions

If we assume the mean function as zero, then a GP prior can be completely parametrized by its covariance function. Hence the problem of learning in a GP regression is exactly the problem of finding suitable properties of the covariance function [Rasmussen 2006]. The covariance function consists of two parts a functional form, (which specifies the shape of functions in the hypothesis space) and a set of hyper-parameters (which define the probability of a function in the hypothesis space). In section 2.3 we have seen in detail how to automatically choose hyper-parameters, the part II of this thesis will detail how to choose the functional form.

The part II of the thesis shows how to incorporate prior information of patterns into building GP models, by choosing different types of covariance functions. The chapter 4 shows a few basic kernel types, while the chapter 5 shows how to combine these basic kernels together and incorporate more complex patterns. This part is heavily inspired from prior works of [Duvenaud 2014, Wilson 2014a, Lloyd 2014, Durrande 2001].

Although the functional forms of the covariance discussed here are commonly used in the machine learning community, unfortunately they are not often used to build engineering models. The original contribution of chapter 4 and chapter 5 is application of these kernels to build meaningful engineering models. In chapter 4 we build a GP model to automatically identify structural dynamics parameters (section 4.3.5) [Chiplunkar 2017b], While in chapter 5 we use a combination of basic kernels to identify onset of non-linear behaviour in physical systems. For example we identify the start of flow separation in NACA 0012 airfoil and start of plasticity in AL6061 alloy using a special kernel (section 5.1.4) [Chiplunkar 2016]. We finally build a GP model to predict position of aerodynamic shock in the transonic regime (section 5.2.5) [Chiplunkar 2017a].

The current chapter unfolds as follows; section 4.1 details a few important properties of covariance functions. Section 4.2 gives some insight into non-stationary covariance functions. While, section 4.3 describes stationary kernels and their relationship in the

Fourier domain. We then leverage this relationship to automatically identify the dynamic behaviour of structural experiments (section 4.3.5).

4.1 Properties

A kernel is a function that maps any pair of inputs into a scalar $\mathcal{R}$, the inputs can be scalars, vectors, categorical variables or even images. The covariance of a GP is an example of a kernel, which specifies covariance of a pair of random functions $f(x_i)$ and $f(x_j)$ situated at points x_i and x_j (mostly written as a function of x_i and x_j).

Most of the learning algorithms work on distance measures, i.e. if two points are closer then their observations will also tend to be similar. Covariance functions specify this measure of distance in a GP Regression. If two points have a high value of covariance, then they are similar, and hence will have similar value of outputs (y). Therefore, by defining a covariance function we encode which type of input points will be termed as similar, thereby effectively encoding biases into our family of functions. Biases based on smoothness (section 4.3.1), linearity (section 4.2.1), differentiability (section 4.3.2) etc can be easily encoded using simple covariance functions.

For zero mean GPs as specified in section 2.1.2 a covariance function between $f(x_i)$ and $f(x_j)$ can be written as equation 4.1.

$$k(x_i, x_j) = cov(f(x_i), f(x_j)) \tag{4.1}$$

A covariance function $k(x_i, x_j)$ is always symmetric, since:

$$k(x_i, x_j) = cov(f(x_i), f(x_j)) = cov(f(x_j), f(x_i)) = k(x_j, x_i) \tag{4.2}$$

$k(x_i, x_j)$ corresponds to a covariance function iff it is a symmetric PSD function [Loeve 1978, Durrande 2001]. Consider for a new random vector $T = \sum \alpha_i f(x_i)$:

$$\begin{aligned} var(T) &= cov\left(\sum_i \alpha_i f(x_i), \sum_j \alpha_j f(x_j)\right) = \sum_i \sum_j \alpha_i \alpha_j cov(f(x_i), f(x_j)) \\ &= \sum_i \sum_j \alpha_i \alpha_j k(x_i, x_j) \end{aligned} \tag{4.3}$$

Since a variance is always non-negative, hence:

$$\sum_i \sum_j \alpha_i \alpha_j k(x_i, x_j) \geq 0 \tag{4.4}$$

According to the Mercer's theorem [Mercer 1909] equation 4.4 is a sufficient condition to prove that $k(x_i, x_j)$ is a PSD function. The positive definite requirement means that the covariance kernel corresponds to an inner product in some basis space [Bishop 2006]. It is generally, difficult to prove if a function is PSD, hence creating new covariance functions is a tough task. Fortunately there already exist a wide variety of covariance functions, this chapter will detail a few basic covariances.

4.2 Non-stationary kernels

Earlier in section 2.1 we have seen SE kernel which is an example of stationary kernels. Stationary kernels are covariance functions which are purely a function of $x_i - x_j$. In this section we list some non-stationary kernels and their properties.

4.2.1 Linear Kernel

The Bayesian Linear regression described during section 1.3 can also be seen as a form of GP Regression but with a Linear covariance function. In the Bayesian linear regression we assume a prior distribution of parameters, this is equivalent to assuming a prior distribution over functions. For a function and its prior as defined by equation 4.5.

$$f(x_i) = \phi(x_i)^T w \qquad \Pr[w] = \mathcal{N}(0, \Sigma_{Prior}) \tag{4.5}$$

The equivalent prior over the functions f[1] can be written as equation 4.6

$$\Pr[f(x)] = GP(0, \phi(x)^T \Sigma_{Prior} \phi(x')) \tag{4.6}$$

The above covariance function ($k(x_1, x_2) = \phi(x)^T \Sigma_{Prior} \phi(x')^T$) describes a family of functions which are linear combinations of the basis functions ($\phi(x)$) [Bishop 2006]. The matrix Σ_{Prior} and σ_n are the hyper-parameters of this GP prior and $\phi(x)$ represents its functional form. Hence a linear basis ($\phi(x) = \{1, x\}^T$) describes family of linear functions (equation 4.7), while a polynomial basis ($\phi(x) = \{1, x, x^2, \ldots, x^P\}^T$) encodes a family of P^{th} order polynomial basis functions.

$$k_{Lin}(x_1, x_2) = w_0 + w_1 x_1 x_2 \tag{4.7}$$

[1] By the affine property of Multivariate Gaussian random variables we have that:

$$X = \mathcal{N}(\mu, \Sigma) \implies AX = \mathcal{N}(A\mu, A\Sigma A')$$

The above equation is the covariance function for a Linear kernel, where the w_0 is the hyper-parameter for bias while w_1 is the hyper-parameter for slope. If we assume an independent noise ϵ on the observations then based on the discussion on noisy GPs (section 2.2.2) the GP prior becomes:

$$\Pr[y(x)] = GP(0, w_0 + w_1 x_1 x_2 + \sigma_n^2 \delta_{xx'}) \tag{4.8}$$

Hence, the bias (w_0), the slope (w_1) and noise (σ_n) are the hyper-parameters of the above prior. The above equation is equivalent to performing Bayesian Linear Regression as discussed in section 1.3. The hyper-parameters can be chosen using marginal likelihood and posterior prediction can be performed based on the discussion on section 2.3.

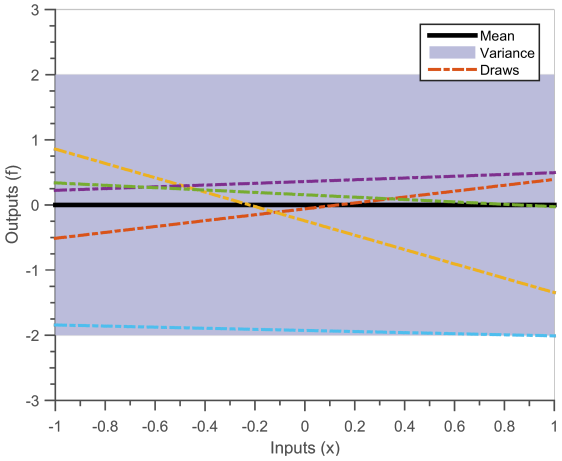
Revisiting Bayesian Linear Regression Let us revisit the experiment performed in section 1.3 but this time using GP regression and a linear kernel. The toy-dataset $\mathcal{D}_1 = \{X = [-0.5, 0.33, 0.66], Y = [0, 0.5, 0.5]\}$ (section 1.3) will be used again. The prior distribution on parameters $\Sigma_{Prior} = [w_0, 0; 0, w_1] \mid w_0 = 1, w_1 = 1$ and prior on noise $\sigma_n = 0.1$ will be the same as used in the earlier experiment. w_0, w_1 and σ_n are the hyper-parameters of this GP prior.

Figure 4.1(a) shows draws from a GP prior with mean zero and Linear kernel as defined in the above paragraph. The solid black line defines the mean function, shaded blue region defines 95% confidence interval (2σ) distance away from the mean. The dashed lines represent five functions drawn at random from a GP prior. Random functions drawn from a linear GP are linear. Figure 4.1(b) show draws from a GP posterior with mean zero and Linear kernel as defined in above para and conditioned on the first data point $X = -0.5, Y = 0$. The posterior mean passes from the data point, random functions drawn from a linear GP are linear.

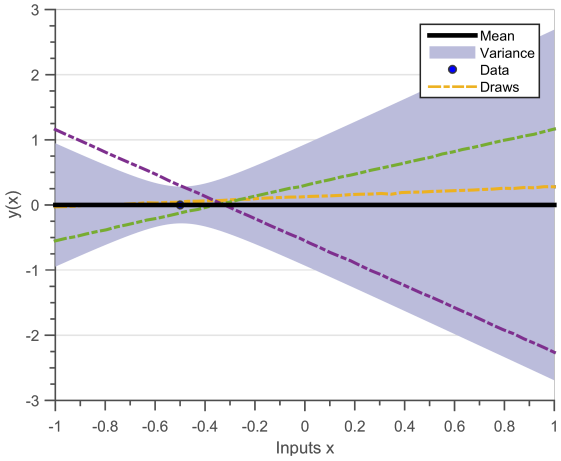
Figure 4.2(a) shows the contours of marginal likelihood with respect to intercept (w_0) and slope (w_1). The marginal likelihood is maximum for $w_0 = 0.2576, w_1 = 0.4584$, this same as posterior mean predicted in equation 1.8. This means that the data $\mathcal{D}_1$ has the highest possibility of coming from a dataset defined by a prior having these hyper-parameters. Figure 4.2(b) shows the posterior for same data set but for the hyper-parameters where marginal likelihood is maximum.

4.2.2 Neural Network Kernel

It can be shown that a neural network with infinitely many hidden units and an error activation function ($erf(z)$) tends to a GP with a Neural network kernel [Neal 2012,

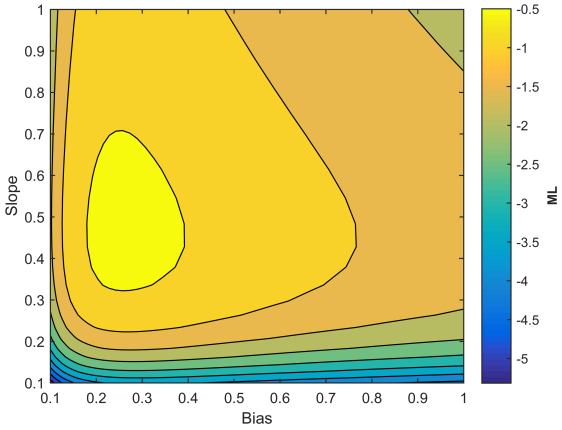


(a) Draws from a GP prior with mean zero and Linear kernel ($k(x_1, x_2) = \phi(x)^T \Sigma_{Prior} \phi(x')^T + \sigma_n^2 \delta_{xx'}$) with $w_0 = 1, w_1 = 1$ and $\sigma_n = 0.1$. Random functions drawn from a linear GP are linear.

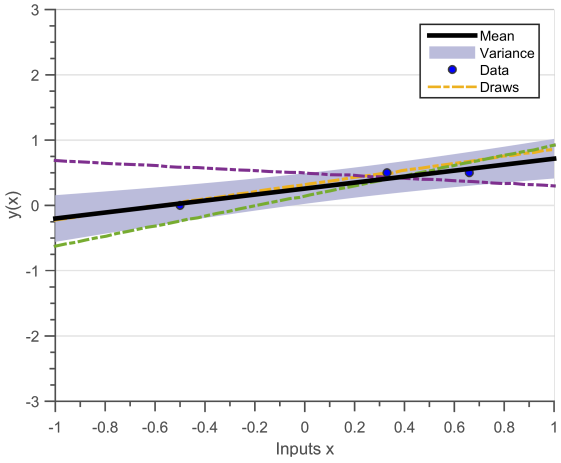


(b) Draws from a GP posterior with mean zero and Linear kernel (figure 4.1(a)) conditioned on the data $X = -0.5, Y = 0$. The posterior mean passes from the data point, random functions drawn from a linear GP are linear

Figure 4.1: Prior and posterior from a GP prior of linear kernel. The solid black line defines the mean function, shaded blue region defines 95% confidence interval (2σ) distance away from the mean. The dashed lines represent five functions drawn at random from a GP prior.



(a) Marginal likelihood contours for varying bias and slope parameter. The noise hyper parameter is ($\sigma_1 = [0.1]$). Marginal likelihood is maximum for $w_0 = 0.2576, w_1 = 0.4584$.



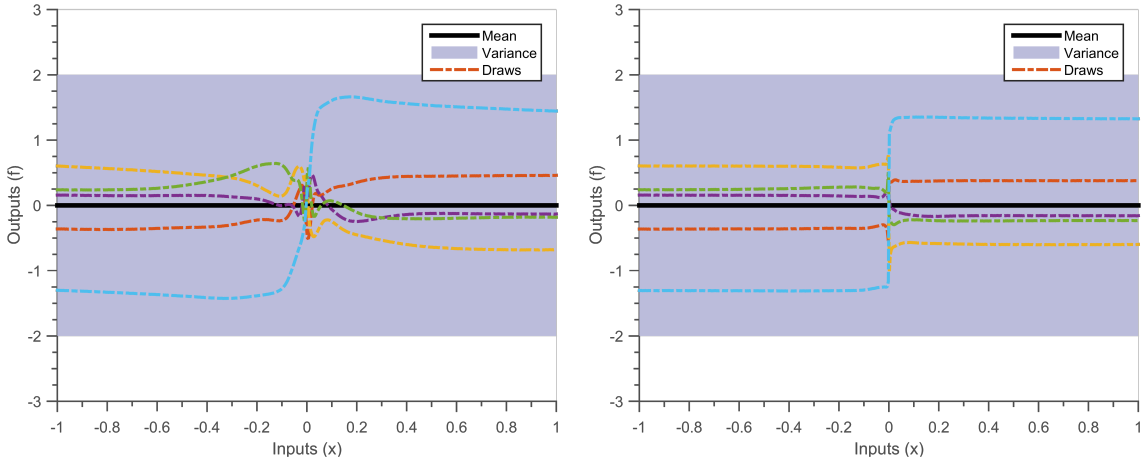
(b) Draws from a GP posterior, conditioned on the dataset $\mathcal{D}_1$ with mean zero and Linear kernel with hyper-parameters that maximize the marginal likelihood.

Figure 4.2: Maximizing Marginal Likelihood Linear kernel

Wilson 2014b] (equation 4.9).

$$K_{NN}(x_1, x_2, \theta) = \theta_1^2 \frac{2}{\pi} sin^{-1}\left(\frac{2x\theta_2 x'}{\sqrt{(1+2x^T\theta_2 x)(1+2x'^T\theta_2 x')}}\right) \tag{4.9}$$

The hyperparameters ($\theta = [\theta_1, \theta_2]$) are; amplitude θ_1 which defines average distance from mean and the length scale θ_2 which define the smoothness of functions. GPs with this covariance function define a space of superimposed sigmoidal functions. Figure 4.3 shows random draws from a Neural Network kernel but varying value of hyper parameters. Figure 4.3(b) has a higher value of smoothness hyper-parameter (θ_2) than figure 4.3(a), which makes the constituent functions have stronger slope. Hence, we can use this kernel to embed the information of discontinuity in our family of functions.



(a) Draws from a GP prior with mean zero and Neural Network kernel (equation 4.9) with $\theta = [1, 10]$.

(b) Draws from a GP prior with mean zero and Neural Network kernel (equation 4.9) with $\theta = [1, 100]$. Higher value of θ_2 signifies increasing slope

Figure 4.3: Draws from Neural Network kernels having different hyper-parameters. The solid black line defines the mean function, shaded blue region defines 95% confidence interval (2σ) distance away from the mean. The dashed lines represent five functions drawn at random from a GP prior.

4.2.3 Constant and Noise kernel

A constant covariance function (equation 4.10) defines a constant function. Here, $\sigma_{constant}$ defines the possible amplitude of the constant function.

$$k_{constant} = \sigma^2_{constant} \tag{4.10}$$

$$k_{noise}(x_i, x_j) = \sigma^2_{noise}\delta_{x_i x_j} \tag{4.11}$$

On the other hand, equation 4.11 defines a white noise kernel, the σ_{noise} defines the amplitude of noise. $\delta_{x_i x_j}$ is a Kronecker delta function which is 1 if $x_i = x_j$ and zero otherwise, this means that observations at two inputs are independent of each other.

4.3 Stationary kernels

Covariance functions which are purely a function of distance $\tau = (x_i - x_j)$ are called as stationary functions, Whereas covariance functions which are functions of absolute value of distance $\tau = |x_i - x_j|$ are called as isotropic covariance functions. Stationary covariance functions remains unchanged if the points x_i, x_j are translated. Hence a family of functions defined by stationary kernels will have similar local features throughout the input domain.

The *Bochner's theorem* defines a relationship between a stationary covariance function and its Fourier transform. It states that Fourier transform of a stationary covariance function exists and is a positive finite measure. If $k(\tau)$ is a stationary covariance function (equation 4.12) then $S(s)$ is its Fourier transform (equation 4.13), also called the power spectrum or spectral density [Bochner 1959, Stein 1999, Cox 1977]. A positive finite measure in this context means that $S(s)$ is non-negative for all frequencies s and integral of $S(s)$ is finite.

$$k(\tau) = \int S(s) e^{2\pi i s^T \tau} ds \tag{4.12}$$

$$S(s) = \int k(\tau) e^{-2\pi i s^T s} d\tau \tag{4.13}$$

The power spectrum is a more interpretable method of understanding constituent functions in a hypothesis space. A covariance function probabilistically defines a hypothesis space, the power spectrum corresponding to this covariance function tells us the strength-/power of the frequencies in this hypothesis space. If a power spectrum has more power at lower frequencies, then the constituent functions in its hypothesis space will be more smooth. Whereas, if the power spectrum has more power at higher frequencies, then the constituent functions in its hypothesis space will be less smooth.

4.3.1 Squared Exponential Kernel

We have already encountered the SE kernel in section 2.1.3. It is one of the most widely used kernel because it defines a hypothesis space of infinitely differentiable (infinitely smooth) functions. The SE kernel is Gaussian in shape (equation 4.14), which makes its Fourier transform also Gaussian in shape (equation 4.15).

$$k_{SE}(\tau, \theta) = \theta^2_{amplitude} exp[-\frac{\tau^2}{2\theta^2_{lengthScale}}] \tag{4.14}$$

$$S_{SE}(s, \theta) = \theta^2_{amplitude} exp[-2\pi\theta^2_{lengthScale} s^2] \tag{4.15}$$

The two hyper-parameters of the SE kernel are the amplitude hyper parameter ($\theta_{amplitude}$), which defines the amplitude of functions and the length-scale hyper-parameter ($\theta_{lengthScale}$), which defines the smoothness of functions. Figure 4.4(a) plots the covariance values of SE kernel with hyper-parameters $\theta_{amplitude} = 1$ and $\theta_{lengthScale} = 1$, for varying values of τ whereas figure 4.4(b) plots the power spectrum for SE kernel for the same hyper-parameters .

As discussed earlier, the covariance function (k_{SE}) defines the measure of similarity between two input points, if two points have high value of covariance then their output values (y) will be similar. If we increase $\theta_{lengthScale}$ then k_{SE} will also increase, for the same value of τ and $\theta_{amplitude}$. This means that, now the points for same value of τ will become more similar and hence the constituent functions in the hypothesis space will become more smooth. As length-scale increases the constituent functions tend to become smoother. Another way to interpret the effect of length-scale is by observing the power spectrum (S_{SE}). If we increase $\theta_{lengthScale}$ then S_{SE} will start decreasing, for the same values of s and $\theta_{amplitude}$. This means that less power will be allocated to higher frequencies and hence the constituent functions in the hypothesis space will become more smooth (Figure 2.2).

Note, as $\theta_{lengthScale}$ tends to infinity, SE kernel tends to a constant covariance function (Fourier transform of a constant function is a delta function). Whereas, if $\theta_{lengthScale}$ tends to zero, an SE kernel tends to a white noise kernel (Fourier transform of a delta function is a constant function).

4.3.2 Matérn Kernel

The Matérn kernel is the second most popular kernel after the squared exponential. This kernel is derived using a Student-t distribution as the power spectrum ($S(s)$) and calculating its inverse Fourier transform. A student-t distribution has more weight on higher-frequencies (when compared to a Gaussian distribution) and hence gives rise to more

non-smooth functions.

$$k_{Matern}(\nu, \tau, \theta) = \theta^2_{amplitude} \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu(\tau))}}{\theta_{lengthScale}} \right)^{\nu} K_{\nu} \left(\frac{\sqrt{2\nu(\tau))}}{\theta_{lengthScale}} \right) \tag{4.16}$$

k_{Matern} is the covariance function for a Matérn kernel, ν is a positive parameter which signifies the degrees of freedom in the Student-t distribution of the power spectrum. $\Gamma(\nu)$ is a Gamma function, while K_ν is a modified Bessel function. The Matérn kernel provides the flexibility to define a hypothesis space of functions with varying degree of derivability. The degree of derivability of the functions in the hypothesis space can be set as $[\nu - 1/2]$, i.e. $\nu = 1/2$ (equation 4.17) defines family of non-differentiable but continuous functions, $\nu = 3/2$ (equation 4.18) defines a family of functions differentiable only once and $\nu = 5/2$ (equation 4.19) defines a family of functions twice differentiable functions. Note, as ν tends to ∞ a Student-t distribution tends to Gaussian distribution, similarly as ν tends to ∞ the Matérn kernel tends to a SE kernel.

$$k_{Matern}(\nu = 1/2, \tau, \theta) = \theta^2_{amplitude} exp[-\frac{\tau}{\theta_{lengthScale}}] \tag{4.17}$$

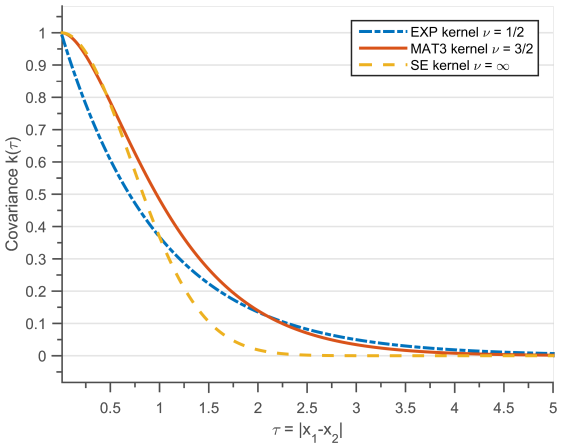
$$k_{Matern}(\nu = 3/2, \tau, \theta) = \theta^2_{amplitude}(1 + \frac{\sqrt{3}\tau}{\theta_{lengthScale}}) exp[-\frac{\sqrt{3}\tau}{\theta_{lengthScale}}] \tag{4.18}$$

$$k_{Matern}(\nu = 5/2, \tau, \theta) = \theta^2_{amplitude}(1 + \frac{\sqrt{5}\tau}{\theta_{lengthScale}} + \frac{5\tau^2}{3\theta^2_{lengthScale}}) exp[-\frac{\sqrt{5}\tau}{\theta_{lengthScale}}] \tag{4.19}$$

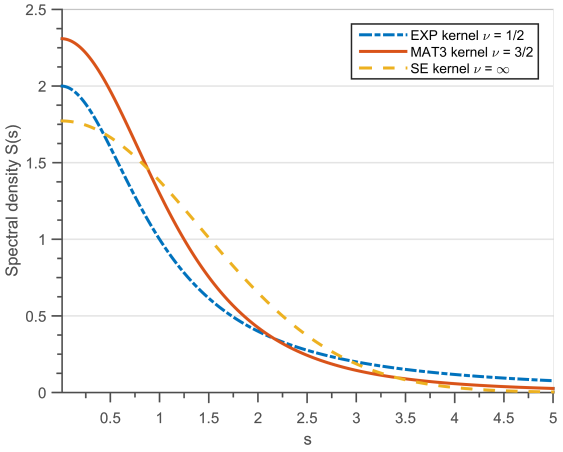
When $\nu = 1/2$ the Matérn kernel is also called the Ornstein-Uhlenbeck or Exponential kernel, this creates a hypothesis space of non-differentiable continuous functions and was used to explain the Brownian motion [Uhlenbeck 1930]. Figure 4.4(a) plots the covariance values of Exponential ($\nu = 1/2$) and Matérn/ ($\nu = 3/2$) kernel (($\theta_{amplitude} = 1$) and ($\theta_{lengthScale} = 1$)) whereas figure 4.4(b) plots the power spectrum for same kernels. We can see that the power spectrum ($S(s)$) of the SE kernel has lowest power for higher frequencies followed by Matérn ($\nu = 3/2$) and Exponential kernel, meaning that the SE kernel has more smooth functions in its hypothesis space followed by Matérn ($\nu = 3/2$) and Exponential kernel.

4.3.3 Experiments

Let us revisit the dataset $\mathcal{D}_2$ used to calculate the posterior in section 2.2, but this time using three different covariance functions. We will use Exponential kernel, Matérn ($\nu = 1/2$) and SE kernel to compare their performance.



(a) Kernel density for exponential, Matérn ($\nu = 3/2$)and SE kernel. The hyper-parameters are amplitude ($\theta_{amplitude} = 1$) and length-scale ($\theta_{lengthScale} = 1$)



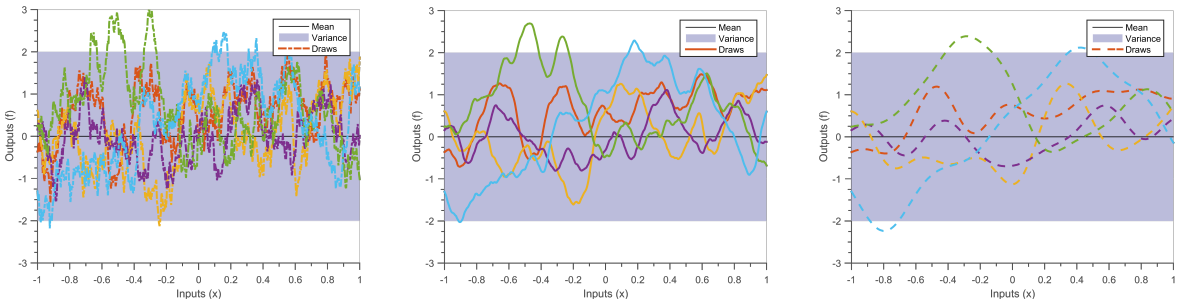
(b) Power spectrum for Exponential, Matérn ($\nu = 3/2$)and SE kernel. The hyper-parameters are amplitude ($\theta_{amplitude} = 1$) and length-scale ($\theta_{lengthScale} = 1$). Exponential and Matérn have more power at higher frequencies when compared to SE kernel.

Figure 4.4: Covariance functions and Power spectrums for three different kernels

We follow the standard framework of GP regression; we first draw random functions from a the covariance functions to judge the hypothesis space, we then calculate the posterior distribution conditioned on dataset $\mathcal{D}_2$, and finally optimize the marginal likelihood to compare the final predictions of the three covariance functions.

Figure 4.5 shows 5 random functions drawn for a zero mean GP with the three covariance functions for same values of hyper-parameters ($\theta_{lengthscale} = 1, \theta_{amplitude} = 1$), their corresponding covariance functions and power spectrums are shown in figure 4.4. The solid black line defines the mean function, shaded blue region defines 95% confidence interval (2σ) distance away from the mean. The dashed lines represent five functions drawn at random from a GP prior. We can observe that figure 4.5(a) draws non-differentiable functions while 4.5(b) and 4.5(c) draw smoother functions. For the same value of the length-scale, Matérn ($\nu = 3/2$) kernel has higher variation, when compared to SE kernel.

Figure 4.6 shows the posterior GP conditioned on the dataset $\mathcal{D}_2$ for three different covariance functions with same hyper-parameters, these hyper-parameters are not optimized with respect to the marginal likelihood. For similar values of hyper-parameters and dataset, Exponential kernel has highest variance followed by Matérn and SE kernel. This is a consequence of the bias vs variance trade-off since the SE kernel has the most restrictive hypothesis space (infinite differentiability is a stricter assumption), followed



(a) Draws from a GP prior with mean zero, Exponential kernel (Matérn with $\nu = 1/2$) and hyper-parameters $\theta_{lengthscale} = 1, \theta_{amplitude} = 1$. Functions drawn from this kernel are non-differentiable.

(b) Draws from a GP prior with mean zero, Matérn kernel $\nu = 3/2$ and hyper-parameters $\theta_{lengthscale} = 1, \theta_{amplitude} = 1$. Functions drawn from this kernel are differentiable only once

(c) Draws from a GP prior with mean zero, SE kernel (Matérn with $\nu = \infty$) and hyper-parameters $\theta_{lengthscale} = 1, \theta_{amplitude} = 1$. Functions drawn from this kernel are infinitely differentiable

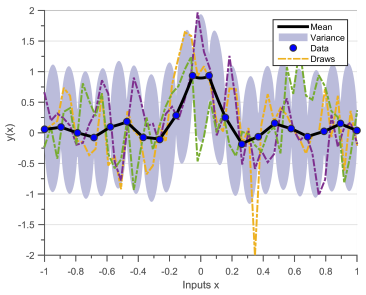
Figure 4.5: Prior distribution and five random draws from 3 different covariance functions. The solid black line defines the mean function, shaded blue region defines 95% confidence interval (2σ) distance away from the mean. The dashed lines represent five functions drawn at random from a GP prior.

by Matérn $\nu = 3/2$ and Exponential kernel.

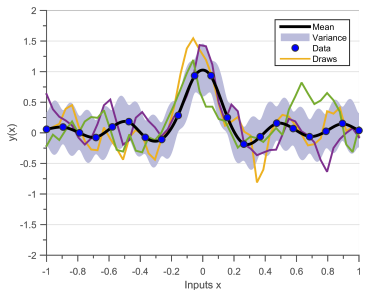
Figure 4.7 shows the posterior GP conditioned on the dataset $\mathcal{D}_2$, for three different covariance functions with optimized hyper-parameters. All the three covariance functions estimate the same value of amplitude hyper-parameter, while having significantly different value of noise hyper-parameter (lowest noise estimated for Exponential kernel followed by Matérn ($\nu = 3/2$) and SE kernel).

The Exponential kernel defines a hypothesis space of non-differentiable functions, hence functions in its hypothesis space have the flexibility to pass through all the datapoints. Whereas, the SE kernel has less number of constituent functions in its hypothesis space, due to its strict bias (infinitely differentiable). If the dataset does not have the same level of smoothness the SE kernel will find the closest function in its hypothesis space, and associate the difference to noise. Hence Exponential kernel will almost always have a low noise estimate than the SE kernel. The infinitely smooth assumption of the squared exponential function is unrealistic in several cases, making the Matérn kernels ($\nu = 5/2$) second most popular choice of kernels [Stein 2012, Cornford 2002].

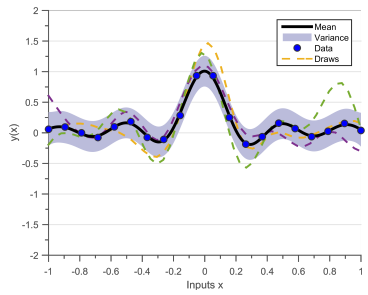
This experiment only shows the different posteriors obtained for same observational data and different functional forms of the covariance function. All three predictions can be the correct interpolations depending on the type of experiment, this is where



(a) Draws from a GP posterior with mean zero and Exopnential kernel (figure 4.5(a)) conditioned on the data $\mathcal{D}_2$. The posterior mean passes through the data points, random functions drawn from exponential kernel are non-differentiable



(b) Draws from a GP posterior with mean zero and Matérn kernel $\nu = 3/2$ (figure 4.5(b)) conditioned on the data $\mathcal{D}_2$. The posterior mean passes through the data points, random functions drawn from this kernel are differentiable only once



(c) Draws from a GP posterior with mean zero and SE kernel $\nu = \infty$ (figure 4.5(b)) conditioned on the data $\mathcal{D}_2$. The posterior mean passes through the data points, random functions drawn from exponential kernel are infinitely differentiable

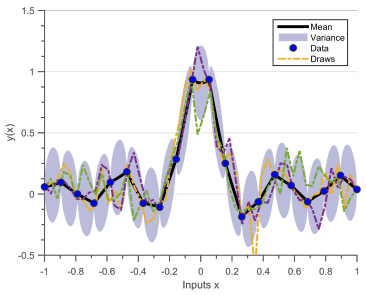
Figure 4.6: Posterior distribution and three random draws from 3 different covariance functions. The solid black line defines the mean function, shaded blue region defines 95% confidence interval (2σ) distance away from the mean. The dashed lines represent five functions drawn at random from a GP prior.

engineering judgment is required. For example, if the dataset $\mathcal{D}_2$ was sampled from a Brownian motion then figure 4.7(a) would be the best fit.

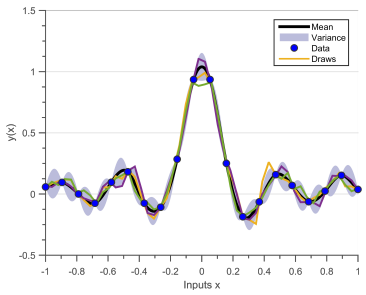
If nothing is known about the data or type of experiment, and the number of hyper-parameters are same, then the covariance function with the maximum optimized marginal likelihood should be preferred. By this logic the SE kernel should be preferred functional form of covariance function for dataset $\mathcal{D}_2$. If the number of hyper-parameters are not same then marginal likelihood tends to be higher for covariance functions with greater number of hyper-parameters. [Duvenaud 2014, Lloyd 2014] propose to use the BIC (equation 4.26) to choose optimal covariance functions if number of hyper-parameters are not same.

4.3.4 Spectral Mixture Kernels

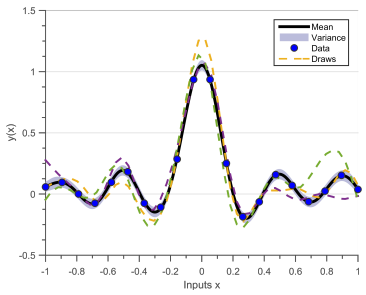
Spectral mixture kernels define a more general class of stationary kernels exploiting the Bochner's theorem [Bochner 1959]. They define the power spectrum ($S(s)$) as a scale location mixture of Gaussians [Wilson 2013]. This has two benefits firstly, with enough Gaussian components, scale location mixtures of Gaussians can approximate a curve up to arbitrary precision[Kostantinos 2000, Bishop 2006]. Secondly, the inverse Fourier transform of a scale location mixture of Gaussians is analytically tractable and is also a mixture



(a) Draws from a GP posterior, conditioned on the dataset $\mathcal{D}_1$ with mean zero and Exponential kernel with hyper-parameters ($\theta_{lengthscale} = 0.215$, $\theta_{amplitude} = 0.312$ and $\sigma_n = 2.8e^{-}5$) that maximize the marginal likelihood $max(ML) = -1$.



(b) Draws from a GP posterior, conditioned on the dataset $\mathcal{D}_1$ with mean zero and Matérn ($\nu = 3/2$) kernel with hyper-parameters ($\theta_{lengthscale} = 0.2$, $\theta_{amplitude} = 0.347$ and $\sigma_n = 1.7e^{-}5$) that maximize the marginal likelihood $max(ML) = 2$



(c) Draws from a GP posterior, conditioned on the dataset $\mathcal{D}_1$ with mean zero and se kernel with hyper-parameters ($\theta_{lengthscale} = 0.151$, $\theta_{amplitude} = 0.358$ and $\sigma_n = 0.01$) that maximize the marginal likelihood $max(ML) = 8$

Figure 4.7: Posterior distributions from three different covariance functions after maximizing the hyper-parameters.

of Gaussians.

$$S_{SM}(s, \mu, \sigma, w) = \sum_{q=1}^{Q} \frac{w_q}{\sqrt{2\pi\sigma_q^2}} \left(\exp\left[-\frac{(s-\mu_q)^2}{2\sigma_q^2} \right] + \exp\left[-\frac{(-s-\mu_q)^2}{2\sigma_q^2} \right] \right) \tag{4.20}$$

Here, S_{SM} is the power spectrum of the spectral mixture kernel. Each Gaussian component q has a mean μ_q, variance σ_q and weight w_q , there are total Q such components. The second term $\exp\left[-\frac{(-s-\mu_q)^2}{2\sigma_q^2} \right]$ is needed because a power spectrum of a valid kernels should be symmetric around $s = 0$. The inverse Fourier transform of such a power spectrum will be a valid kernel and can be written as equation 4.21, for a detailed derivation refer to [Wilson 2014a].

$$k_{SM}(\tau, \mu, \sigma, w) = \sum_{q=1}^{Q} w_q cos(2\pi\mu_q) exp[-2\pi^2\tau^2\sigma_q^2] \tag{4.21}$$

Here, k_{SM} is the covariance function of the above power spectrum (equation 4.20). The parameter w_q is the weight of the Gaussian component q, the mean of the Gaussian component μ_q defines the period of kernel while the variance σ_q of the Gaussian component denotes inverse of the length scale . Spectral mixture kernels are easily interpretable and

can be used as a replacement for many available kernels. They can be used to perform pattern discovery, infer negative covariances and perform extrapolation [Wilson 2014a].

We propose to use this relationship between the covariance function and its Fourier transform to automatically identify parameters of a dynamic system. Dynamic engineering systems are generally parametrized by their modal frequencies and participation factors. In structural engineering, identification of modal frequencies is an important step for certification, while minor change in modal frequencies can help in speedy discovery of failure. In the next section we apply the Spectral Mixture kernel to automatically identify the modal frequencies of a structural system, parts of the following work have been published in [Chiplunkar 2017b].

4.3.5 Application: Identifying Structural Dynamics Parameters

Modal analysis has been widely used as a means of identifying dynamic properties such as modal frequencies, damping ratios and mode shapes of a structural system. Traditionally, the system is subjected to artificial input excitations and output deformations (displacements, velocities or accelerations) are measured. These later help in identifying the modal parameters of the system, this process is called Experimental Modal Analysis (EMA).

Since the last decade Operational Modal Analysis (OMA) has gained considerable interest in the community. OMA identifies the modal parameters only from the output measurements while assuming ambient excitations as random noise. OMA is cheaper because it does not require expensive experimental setup and and can be used in real time operational use cases such as health monitoring [Peeters 2005, Shahdin 2010, Rainieri 2007].

In the last few decades several algorithms primarily using the assumption of second order differential, Multi Degree Of Freedom (MDOF) system (equation 4.22) have been developed to find modal parameters in [Guillaume 2003, Richardson 1982].

$$[M]\{\ddot{x}(t)\} + [C]\{\dot{x}(t)\} + [K]\{x(t)\} = \{f(t)\} \tag{4.22}$$

Here, $[M]$, $[C]$ and $[K]$ denote the mass, damping and stiffness matrices respectively. While, $\{x(t)\}$ and $\{f(t)\}$ denote the displacement and force vectors at the time t. Figure 4.8(a) shows an example of ambient measurements $x(t)$ on a structure. In almost all OMA algorithms the measurement $x(t)$ is assumed to be generated from a random force excitation.

Earlier Work The Natural Excitation Technique [James III 1995] proves that the auto-correlation function $k(\tau)$ can be written as sum of decaying sinusoid's [Spitznogle 1970, Ibrahim 1977, Guillaume 2003]. The auto-correlation describes the similarity between mea-

surement as a function of time lag τ between them (figure 4.8(b)).

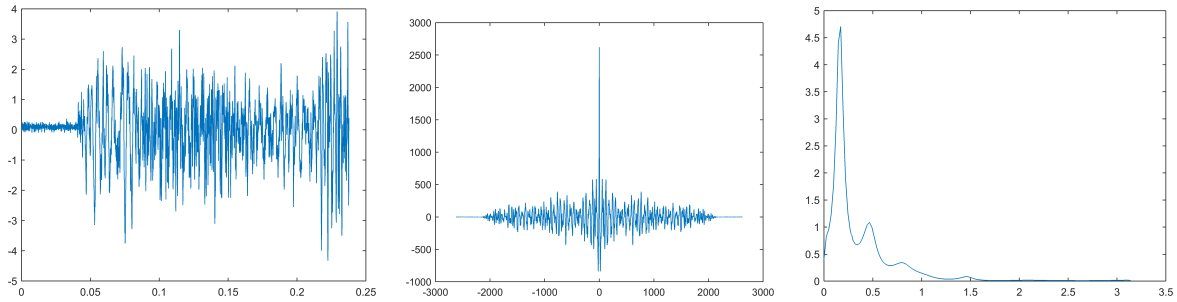
$$k(\tau) = \int x(t)x(t-\tau)dt \quad k(\tau) = \sum A_i exp(-\lambda_i \tau) sin(B_i \tau) \tag{4.23}$$

Here, $k(\tau)$ denotes the auto-correlation for random vector $x(t)$ as a function of time lag τ. While, λ_i and A_i denotes the modal frequency and mode shapes for the i^{th} mode. The above coefficients are found by minimizing the least square error between the measured $k(\tau)$ and the predicted $k(\tau)$ from equation 4.23

If we assume the measurement $x(t)$ to be a stationary random process, then according to Bochner's theorem the Fourier transform of $k(\tau)$ (power spectrum $S(s)$) exists [Bochner 2016]. Figure 4.8(c) shows the power spectrum calculated for the measurement $x(t)$ shown in figure 4.8(a). Using the above mentioned second order differential assumption a Rational Fractional Polynomial (RFP) (equation 4.24) can be used to fit a power spectrum [Richardson 1982, Allemang 1998, Chauhan 2007].

$$S(s) = \int k(\tau) e^{-2\pi i s^T s} d\tau \quad S(s) = \frac{\sum a_k(s)^k}{\sum b_l(s)^l} \tag{4.24}$$

Here, the poles of the polynomial denote the modal frequencies, while other modal parameters can be derived from the coefficients a_k and b_l. The coefficients of the polynomial can be found by minimizing the least squared error. RFP based algorithms face problems of numerical stability a value of number of modes (l) increases.



(a) Measured output on accelerometers $x(t)$

(b) Auto-correlation $k(\tau)$ of the measured output fig: 4.8(a)

(c) Power spectrum density $S(s)$ of the output fig: 4.8(a)

Figure 4.8: Different types of measurements for estimation of Modal parameters in OMA

Contribution The above mentioned OMA algorithms "Natural Excitation Technique" in the time lag (τ) domain and "Rational Fractional Polynomial" in the frequency domain (s), assume the dynamic behaviour to be a second order differential system. This assumption fails for non-linear systems and for cases where modal fre-

Measurement	Auto-correlation	Power Spectrum
$x(t)$	$k(\tau) = \int x(t)x(t-\tau)dt$	$S(s) = \int k(\tau)exp(-2\pi i s^T \tau)d\tau$
Assumption: Second Order Differential		
	$\sum A_i exp(-\lambda_i \tau)sin(B_i \tau)$	$\frac{\sum a_k(s)^k}{\sum b_l(s)^l}$
Assumption: Gaussian Mixture Model		
$GP(0, k_{SM})$	$\sum w_i cos(2\pi\mu_i\tau)exp\{-2\pi^2\sigma_i^2\tau^2\}$	$\sum w_i \frac{1}{\sqrt{2\pi\sigma_i^2}} exp\{\frac{1}{2\sigma_i^2}(s-\mu_i)^2\}$

Table 4.1: Comparison of fitting functions

quencies are very close. Instead we propose to use the spectral mixture kernel to fit the measurement $x(t)$.

Using the same hypothesis used in section 4.3.4, we can say that a scale-location mixture of Gaussian components can approximate any distribution. We thus place a scale-location mixture of Gaussians on the power spectrum ($S(s)$), this means that we are assuming a prior distribution of $x(t)$ as a GP with a Spectral Mixture kernel (equation 4.25). The hyper-parameters of the system are: the mean μ_q defines the modal frequency, the variance σ_q which is a measure of damping, the weight w_q which defines the participation factor of component q and the total number of components Q.

$$\Pr[x(t)] = GP(0, k_{SM} = \sum_{q=1}^{Q} w_q cos(2\pi\mu_q)exp[-2\pi^2\tau^2\sigma_q^2) \tag{4.25}$$

We would like to emphasize that keeping the computational complexities aside, fitting a Spectral Mixture GP in time-domain (equation 4.25), fitting spectral mixture (equation 4.21) on covariance values and fitting a scale-location mixture of Gaussians (equation 4.20) on the power spectrum are equivalent. In the publication [Chiplunkar 2017b] we fit a scale-location mixture of Gaussians on the power spectrum, here we propose to fit the GP on the measurements $x(t)$. Refer to table 4.1 for a more comprehensive view at various fitting functions.

While the modal parameters can be chosen by minimizing the least square error, how to choose the number of modes is a recurring question in several OMA algorithms. This problem is partially resolved by using stabilization diagrams or mode identification functions [Allemang 1998, Williams 1985, Shih 1988]. But in practical situations, engineering judgment is often required to estimate the optimal modal order. To automatically choose Q, we use the Bayesian Information Criteria (BIC) [Findley 1991] which penalizes more complex models to estimate the parameter Q. The BIC esti-

mate has been earlier used to find the optimal functional form of covariance function [Duvenaud 2013]

$$BIC(Q) = N \ln(ML) + N_{hyp} \ln(N) \tag{4.26}$$

Here, n denotes the number of data-points to fit, ML denotes the marginal likelihood of the fit and N_{hyp} denote the number of hyper-parameters to fit. The BIC performs a trade-off between the data-fit term $N \ln(ML)$ and the complexity penalty term $N_{hyp} \ln(N)$, basically penalizing for over-fitting. Lowest value of BIC is preferred.

Define an algorithm here on how to find the Q etc

Generally, identification of modal frequencies requires engineering judgment, i.e. engineers have to look at the stabilization diagrams and figure out the optimal value of modal order. Using the Spectral mixture kernel we have encoded the knowledge of peaks in the power spectrum which in combination with BIC has eliminated the need to perform costly engineering judgment exercises and made the process automatic. We now compare the accuracy of finding modal frequencies using a spectral mixture kernel vs NeXT methodology for a toy-dataset, and then later for a real dataset of HTC building [Brincker 2000].

Results on a toy data-set

In this section we conduct experiments, applying our approach on a highly damped 3 degree of freedom system[2]. As stated earlier we fit a GP model having a spectral mixture covariance on the measurements $x(t)$ for varying number of Gaussian components Q. We then evaluate the BIC to find the optimal value of Q for this measurement. The results of automatic identification are then compared to that of NeXT identification method.

All experiments were performed on an Intel quad-core processor with 4Gb RAM. The Spectral Mixture technique suffers from multiple minimas and thus care should be taken while initializing the hyper-parameters[3].

Table 4.2 shows the comparison of frequencies for Modal frequencies predicted using NeXT algorithm and Spectral Mixture algorithm. We can observe that the NeXT method cannot predict the third frequency with enough accuracy.

Figure 4.9(a) shows the stabilization diagram with increasing number of Gaussians Q. As we progressively increase the number of components we start getting more and more modal frequencies. We call a frequency as stabilized if the difference between modal frequencies of two subsequent Q is less than 1%, the green points are stabilized frequencies.

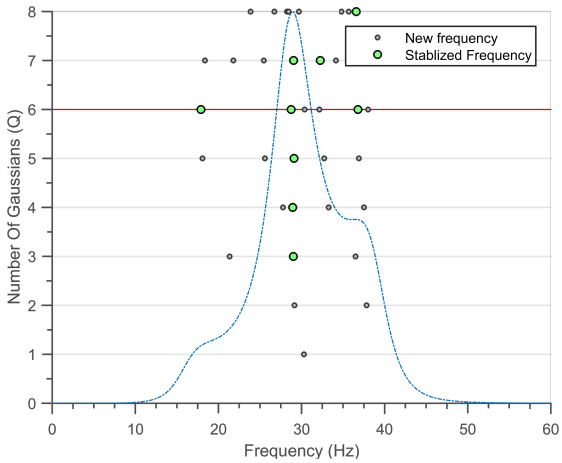
[2]Dataset is available at github link: **insert github link**

[3]We first fit a Gaussian Mixture model on the power spectrum to initialize the hyper-parameters and then optimize the marginal likelihood for speedy optimization.

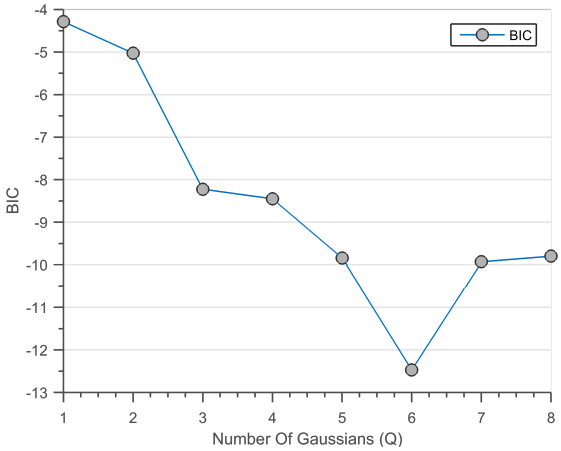
	Real Frequency	NeXT Frequency	Spectral Mixture
First Frequency (Hz)	17.5	17.88	17.55
Second Frequency (Hz)	30	28.97	28.75
Third Frequency (Hz)	35	39.32	36.80

Table 4.2: Comparison of Modal frequencies for toy data-set

The, figure 4.9(b) shows the BIC criterion with increasing number of Gaussian's Q. We can see that that the BIC is minimum for $Q = 6$ and hence if we add anymore Gaussians for our dataset we will be over-fitting. The red line in figure 4.9(a) shows the Q with minimum BIC and hence the stabilized frequencies for this automatically become our modal frequency predictions.



(a) Stabilization diagram with increasing number of Gaussians Q, the dots denote the stabilized frequencies. We can observe that as the number of Q increases the algorithm starts finding better and better modes.



(b) The BIC criterion with increasing number of Gaussian's Q. We can see that that the BIC is minimum for $Q = 6$ and hence if we add anymore Gaussian's for our dataset we will be performing over-fitting

Figure 4.9: Results of spectral mixture kernels on a toy dataset

Results on a HTC building dataset

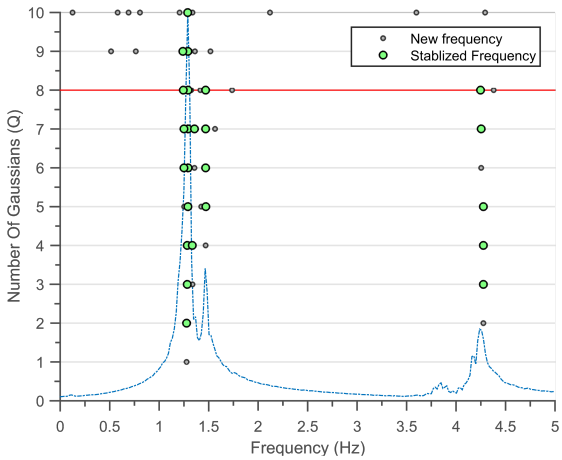
We now apply our methodology on real ambient response of Heritage Court Building[4]. The responses are measured at 6 different points on the building resulting in 6 different power spectrums. We compare the modal frequencies obtained using Spectral Mixture kernel with the modal frequencies obtained in the original paper [Brincker 2000]. To compare the performance we automatically identify modal frequencies present in each of the 6 power spectrums and take the mean of the stabilized frequencies.

[4]The data is available at : http://www.brinckerdynamics.com/oma-toolbox

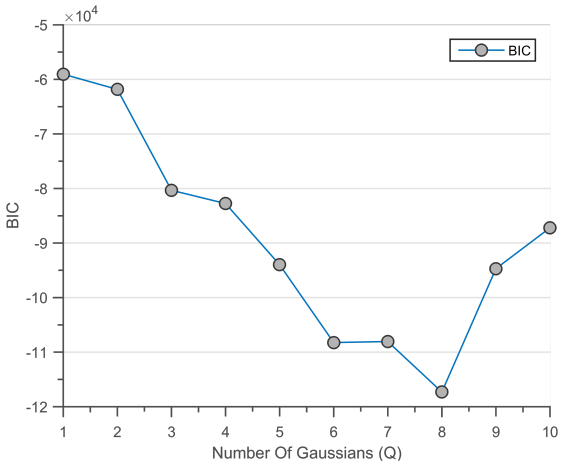
Table 4.2 shows the comparison of Modal frequencies predicted in [Brincker 2000] with Spectral Mixture algorithm. The results of the two methods are very similar, although the frequencies obtained by spectral mixture GP are completely automatic.

	Spectral Mixture	[Brincker 2000]
First Frequency (Hz)	1.23	1.23
Second Frequency (Hz)	1.29	1.27
Third Frequency (Hz)	1.43	1.45
Fourth Frequency (Hz)	3.87	3.86
Fifth Frequency (Hz)	4.28	4.25

Table 4.3: Comparison of Modal frequencies for HTC data-set



(a) Stabilization diagram for one of the 6 power spectrums. The green dots denote the stabilized frequencies, the red line is denotes minimum BIC. We can observe that as the number of Q increases the algorithm starts finding better and better modes.



(b) The BIC criterion with increasing number of Gaussian's Q. We can see that that the BIC is minimum for $Q = 8$ and hence if we add anymore Gaussian's for our dataset we will be performing over-fitting

Figure 4.10: Results of spectral mixture kernels on real data from HTC building

Figure 4.10(a) shows the stabilization diagram with increasing number of Gaussians Q. We can observe that as the number of Q increases the algorithm starts finding better and better modes. We can observe that there are three modes which start stabilizing from $Q = 5$. The, figure 4.10(b) shows the BIC criterion with increasing number of Gaussian's Q. We can see that that the BIC is minimum for $Q = 8$ and hence if we add anymore Gaussians for our dataset we will be performing over-fitting.

more detailed conclusion In the current setting of the Spectral Mixture model we propose an automatic way to identify the most important frequencies of a structural system.

Neither the mode shapes nor the damping ratios are estimated in the current format. In future we would like to derive a method to estimate mode-shape and damping ratio such that the contributions of neighbouring Gaussians are also taken into account.

4.4 Discussion

This chapter discuss only a small number of possible covariance functions, table 4.4 provides a list of basic covariance functions that we have encountered thus far. Several other other functional forms of the covariance function exist the literature, for example Gibbs kernel, Rational Quadratic kernel, Periodic kernel etc for more detailed list please refer to works of [Rasmussen 2005, Duvenaud 2013, Wilson 2014a]. .

Kernel Name	Expression $k(x_i, x_j)$	Constituent functions
Constant	σ_c^2	Constant functions
Noise	$\sigma_n^2 \delta_{x_i x_j}$	White noise
Linear	$\phi(x_i)\Sigma\phi(x_j)$	Linear combination of ϕ
Neural Network	$\theta_1^2 \frac{2}{\pi} sin^{-1}\left(\frac{2x\theta_2 x'}{\sqrt{(1+2x^T\theta_2 x)(1+2x'^T\theta_2 x')}}\right)$	Linear combinations of Sigmoids
Standard Exponential	$\theta^2_{amplitude} exp[-\frac{\tau^2}{2\theta^2_{lengthScale}}]$	Infinitely Differentiable functions
Matérn	$\theta^2_{amplitude} \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu(\tau))}}{\theta_{lengthScale}}\right)^{\nu} K_\nu \left(\frac{\sqrt{2\nu(\tau))}}{\theta_{lengthScale}}\right)$	Adjustable differentiability
Spectral Mixture	$\sum_{q=1}^{Q} w_q cos(2\pi\mu_q) exp[-2\pi^2\tau^2\sigma_q^2]$	Summation of periodic kernels

Table 4.4: List of covariance functions

We start off this chapter by defining a few important properties of the covariance functions and then list a few covariance functions. For each covariance function we try to give a visualization of the shape of functions by randomly drawing functions from their priors. The main contribution of this chapter is demonstration on how to use GP regression to automatically detect parameters of structure dynamics. Using the spectral mixture kernels we demonstrate how to build models for structural dynamic experiments and automatically identify dynamic parameters such as modal frequency. Without doubt this is very nascent stage of application of Spectral Mixture Kernel for system identification, and there remains problems such as identification of mode-shape, and damping ratio in

this algorithm. We wish to tackle these problems in the future.

As mentioned earlier the core aim of machine learning was to identify patterns and extrapolate on data automatically. There are two main approaches in pattern discovery for GPs; one is based on increasing the hypothesis space by defining a process over all stationary kernels [Wilson 2012]. The second which defines a language of kernels and iteratively adds basic kernels to come up with an explanation to the pattern in the data [Lloyd 2014]. In the next chapter we will look in detail how to build more complex kernels using basic kernels. Which approach will be retained in the long term is tough to say but research in automatic pattern detection is a highly active subject.

Chapter 5

Combining Basic Covariance Functions

What if, the kind of patterns that we wish to encode are not possible by using the basic kernels described in chapter 4? What if, we wish to encode several different patterns in our hypothesis space? Or what if, we want to build models for data sets with more than one input dimensions? Thankfully, many new kernels can be constructed by merging a few basic kernels, in this we answer the above questions.

The original contribution of this chapter is to apply the newly created covariance functions to create engineering design models. The set of equations below are a few simple methods to create valid covariance functions [Bishop 2006, MacKay 2003, Durrande 2001, Durrande 2013].

$$k(x_1, x_2) = k_1(x_1, x_2) + k_2(x_1, x_2) \tag{5.1}$$
$$k(x_1, x_2) = k_1(x_1, x_2) \times k_2(x_1, x_2) \tag{5.2}$$
$$k(x_1, x_2) = q(x_1)k_1(x_1, x_2)q(x_2) \tag{5.3}$$
$$k(x_1, x_2) = k_1(h(x_1), h(x_2)) \tag{5.4}$$
$$k(x_1, x_2) = g(g(k_1(x_1, x_2), x_1), x_2) \tag{5.5}$$

Here, $k_1(x_1, x_2)$ and $k_2(x_1, x_2)$ are valid covariance function, while $q(x)$ and $h(x)$ are any continuous functions. While, $g(.) \in \mathcal{C}^2$ is an linear operator, for more discussion on equation 5.5 refer to chapter **??**. Let us take the case for a 2-dimensional input vector x such that $x = \{x^1, x^2\}$ (x^1, x^2 are values of x in the two dimensions). Then the following covariance functions are also valid.

$$k(x_1, x_2) = k_1(x_1^1, x_2^1) + k_2(x_1^2, x_2^2) \tag{5.6}$$
$$k(x_1, x_2) = k_1(x_1^1, x_2^1) \times k_2(x_1^2, x_2^2) \tag{5.7}$$

The current chapter is written to provide intuition on what happens when we combine covariance functions. This chapter unfolds as follows; section 5.1 provides intuition on combining kernels for one-dimensional inputs, while section 5.2 details how to create covariance functions for higher-dimensions (equation 5.6 and 5.7) inputs.

5.1 One dimensional inputs

Combining kernels can give rise to interesting features, in this section we provide intuition on combining kernels in one dimension. Section 5.1.1 details of effects of multiplying kernels (equation 5.2) while section 5.1.2 describes effects of adding kernels (equation 5.1). Section 5.1.4 describes the change-point kernel, we use the change-point kernel to detect start of plasticity in an elastic beam and start of flow separation on an airfoil [Chiplunkar 2016].

5.1.1 Multiplying Kernels

Multiplying two covariance functions acts as an AND operator, the resulting kernel has high value only if we have high value on both the kernels. Multiplying a Linear kernel T times, will result in a T^{th} order polynomial regression (equation 5.8).

$$k_{Lin} = w_0 + w_1 x_1 x_2 \quad k_{Poly} = \prod_{i=1}^{T} \left(w_0^i + w_1^i x_1 x_2\right) \tag{5.8}$$

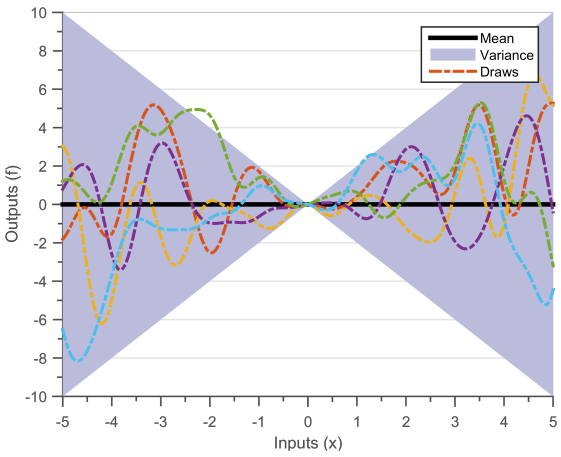
Equation 5.9 shows the prior obtained after multiplying a Linear and a SE kernel. This covariance function resembles a SE covariance function but with the amplitude hyper-parameter ($\theta_{amplitude}$) proportional to distance x.

$$k_{Multi} = \overbrace{x_1 x_2}^{\theta_{amplitude}} \; exp[-\frac{\tau^2}{2}] \tag{5.9}$$

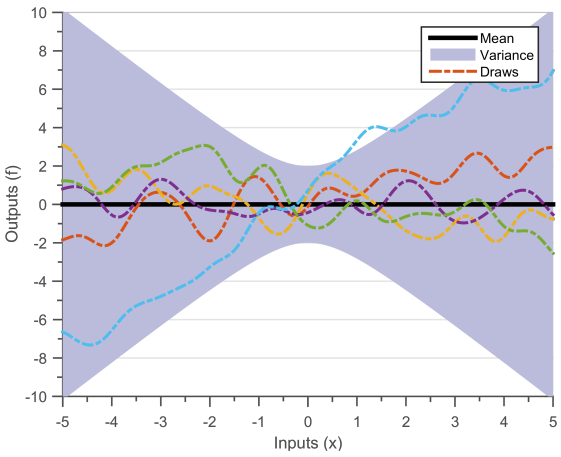
Figure 5.1(a) shows random draws obtained using k_{Multi} covariance, the hyper-parameters of the linear kernel are $w_0 = 0$ and $w_1 = 1$, this means that there is no intercept and $k_{Lin}(x_1, x_2) = x_1 x_2$. The hyper-parameters of the SE part are $\theta_{amplitude} = 1$ and $\theta_{lengthScale} = 1$, similar to figure 4.5(c). Since multiplying two kernels is an AND operation, k_{Multi} tends to zero at $x = 0$ since k_{Lin} is zero at $x = 0$.

5.1.2 Adding Kernels

Adding two kernels acts as an OR operator, this means that the resulting kernel will have high value if either of the two kernels have high value [Durrande 2011].



(a) Draws from a GP prior with mean zero and kernel obtained by **multiplying** a Linear kernel with SE kernel. The hyper-parameters of the linear kernel are $w_0 = 0$ and $w_1 = 1$ while the hyper-parameters of the SE part are $\theta_{amplitude} = 1$ and $\theta_{lengthScale} = 1$. We see that the variance at $x = 0$ goes to zero, since multiplication is an AND operation



(b) Draws from a GP prior with mean zero and kernel obtained by **adding** a Linear kernel with SE kernel. The hyper-parameters of the linear kernel are $w_0 = 0$ and $w_1 = 1$ while the hyper-parameters of the SE part are $\theta_{amplitude} = 1$ and $\theta_{lengthScale} = 1$. We see that the variance at $x = 0$ goes to variance of SE ekrnel, since multiplication is an OR operation

Figure 5.1: Random draws from combining a Linear and SE kernel. The solid black line defines the mean function, shaded blue region defines 95% confidence interval (2σ) distance away from the mean. The dashed lines represent five functions drawn at random from a GP prior. Random functions drawn from a linear GP are linear.

Figure 5.1(a) shows the prior obtained after adding a Linear and a SE kernel. The hyper-parameters of the linear kernel are $w_0 = 0$ and $w_1 = 1$, this means that there is no intercept and $k_{Lin}(x_1, x_2) = x_1 x_2$. The hyper-parameters of the SE part are $\theta_{amplitude} = 1$ and $\theta_{lengthScale} = 1$, similar to figure 4.5(c).

The Linear kernel discussed in section 4.2.1 is a sum of three covariance function; a constant covariance, a linear covariance and a white noise covariance function (equation 5.10). Similarly, the noisy posterior case discussed in section 2.3 is a case of adding a SE kernel with a white noise kernel, a more complicated noise model can be created by adding several kernels together.

$$k(x_1 x_2) = \overbrace{w_0}^{Constant} + \overbrace{w_1 x_1 x_2}^{Linear} + \overbrace{\sigma_n^2 \delta_{xx'}}^{Noise} \tag{5.10}$$

An interesting consequence of adding kernels is that, now we can decompose the result into additive parts. Suppose k_{Sum} is a covariance function by adding n covariance functions $k_1, k_2, \ldots, k_n$ (equation 5.1), then the posterior mean and covariance can be written as

equation 5.11 and 5.12.

$$\mathbf{E}[f(x) \mid X, Y, k_{Sum}] = \sum_{i=1}^{n} K_i(x_* X)(K_{Sum}(X, X))^{-1} Y \tag{5.11}$$

$$Cov[f(x) \mid X, Y, k_{Sum}] = \sum_{i=1}^{n} \left[K_i(x_* x_*) - K_i(x_* X)(K_{Sum}(X, X))^{-1} K_i(X x_*)\right] \tag{5.12}$$

This comes very handy while iteratively discovering structure in the data. [Rasmussen 2005] use a sum of several kernels to interpolate CO_2 content in the atmosphere through the years. [Ghahramani] propose to automatically detect pattern by iteratively adding new kernels until the posterior error variance represents a white noise .

5.1.3 Change-Point kernels

The Change Point (CP) kernel was introduced to recognize changes in regimes, and adapt the covariance function accordingly. They were initially introduced to identify change points in time-series modelling [Osborne 2010, Saatçi 2010]. These kernels can be defined through addition and multiplication with sigmoidal functions (equation 5.13).

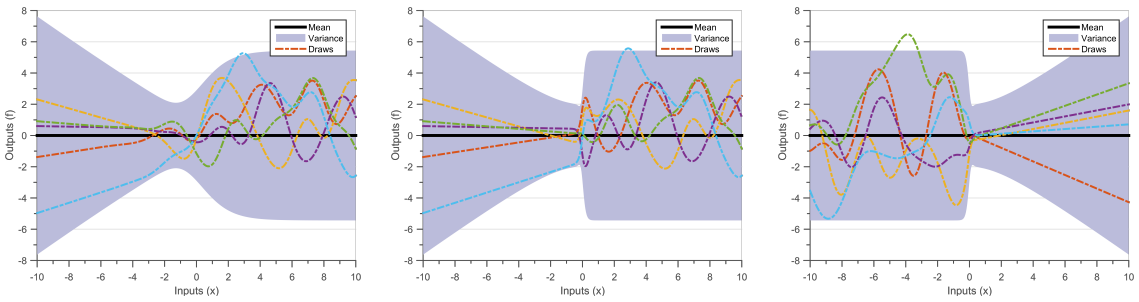
$$sigm(x, \theta) = \frac{1}{\theta_{intensity} + e^{\theta_{changeLocation} - x}} \tag{5.13}$$

$$CP(k_1, k_2, x_1, x_2) = sigm(x_1) k_1 sigm(x_2) + (1 - sigm(x_1)) k_2 (1 - sigm(x_2)) \tag{5.14}$$

The hyper-parameters of this kernel are the $\theta_{changeLocation}$ which determines the location of the change point, and $\theta_{intensity}$ determine the intensity of change between the two patterns. Figure 5.2 shows the randomly drawn functions from a change point kernel for varying values of $\theta_{intensity}$, where the regime changes from a Linear kernel to a SE kernel. The hyper-parameters of the linear kernel are $w_0 = 0$ and $w_1 = 1$, this means that there is no intercept and $k_{Lin}(x_1, x_2) = x_1 x_2$. The hyper-parameters of the SE part are $\theta_{amplitude} = 1$ and $\theta_{lengthScale} = 1$, similar to figure 4.5(c). We see that as the value of $\theta_{intensity}$ increases the regime change happens more rapidly, while if $\theta_{intensity}$ changes sign then the order of regime changes from SE kernel to LInear kernel.

5.1.4 Application: Identifying onset of non-linearity using CP kernel

Several physical processes can be represented using a linear approximation in some part of their regime. This approximation is possible because the linear effects dominate in that



(a) Draws from a GP prior with mean zero and CP kernel (equation 5.14) between a Linear and a SE kernel with $[\theta_{intensity}, \theta_{changeLocation}] = [1, 0]$.

(b) Draws from a GP prior with mean zero and CP kernel (equation 5.14) between a Linear and a SE kernel with $[\theta_{intensity}, \theta_{changeLocation}] = [10, 0]$. Notice if the value of $\theta_{intensity}$ increases then the change between two patterns becomes more significant.

(c) Draws from a GP prior with mean zero and CP kernel (equation 5.14) between a Linear and a SE kernel with $[\theta_{intensity}, \theta_{changeLocation}] = [-10, 0]$. Notice if the sign of $\theta_{intensity}$ changes then the order of kernels gets reversed.

Figure 5.2: Random draws by having a change-point between a Linear and SE kernel. The solid black line defines the mean function, shaded blue region defines 95% confidence interval (2σ) distance away from the mean. The dashed lines represent five functions drawn at random from a GP prior. Random functions drawn from a linear GP are linear.

part of the regime, but they eventually wear off and second and third order effects start becoming more powerful.

This basic assumption is used in making simple models in several domains; for example in a material during the elastic regime $Stress \propto Strain$ is a basic linear approximation. The proportionality constant between Stress and Strain is called as the Young's Modulus, which is unique for each materials. When the elastic regime starts wearing off, non-linear behaviour called plasticity starts taking over and the approximation $Stress \propto Strain$ is not valid anymore. Similarly in aerodynamics, when characterizing a flow over an airfoil during the Linear regime $Lift \propto AngleOfAttack$, the airflow is attached on the airfoil during this regime. When the airflow starts separating from the airfoil the non-linear effects start dominating.

The value of these basic physical parameters such as slope (eg. Young's Modulus, coefficient of lift) and location of change in regime (eg. start of plasticity and separation of flow) are progressively fed into further simulations. Generally, the slope and location of change point are evaluated painstakingly using engineering judgment, this is a painfully slow and costly process. We propose to estimate the slope and location of change-point automatically using a GP with change point kernel. Using the CP kernel which transitions from linear domain (linear kernel) to non-linear domain (SE kernel),

prior information of the transition is encoded in the kernel structure.

We perform our experiments on openly available Stress and Strain data of Aluminum Alloy 6061 [Kaufman 1999] and Lift and Angle data of NACA 0012 airfoil[a]. To estimate the CP hyper-parameters, we again perform a 10-fold cross validation. The dataset will be randomly partitioned into 10 subsets containing an equal number of points. Of the 10 subsets, a single subset is retained as the test dataset, and the remaining 9 (10 - 1) subsets are used as training data. The cross-validation process is then repeated 10 times (the folds), with each of the k subsets used exactly once as the validation data. The marginal-likelihood is optimized for each of the training dataset and location of change-point and slope of the linear regime is noted. We then compare their average with the values available in the literature.

[a]Airfoil data from: http://airfoiltools.com/airfoil/details?airfoil=n0012-il

Table 5.1 shows the results of physical parameters for AL 6061 when calculated using change-point kernel vs that available in the literature. We can observe that the change-point automatically predicts the correct values of Young's Modulus and start of non-linearity.

	Change-point	Literature
Young's Modulus (GPa)	68.5	68.9
Start of plasticity	0.94%	0.95%

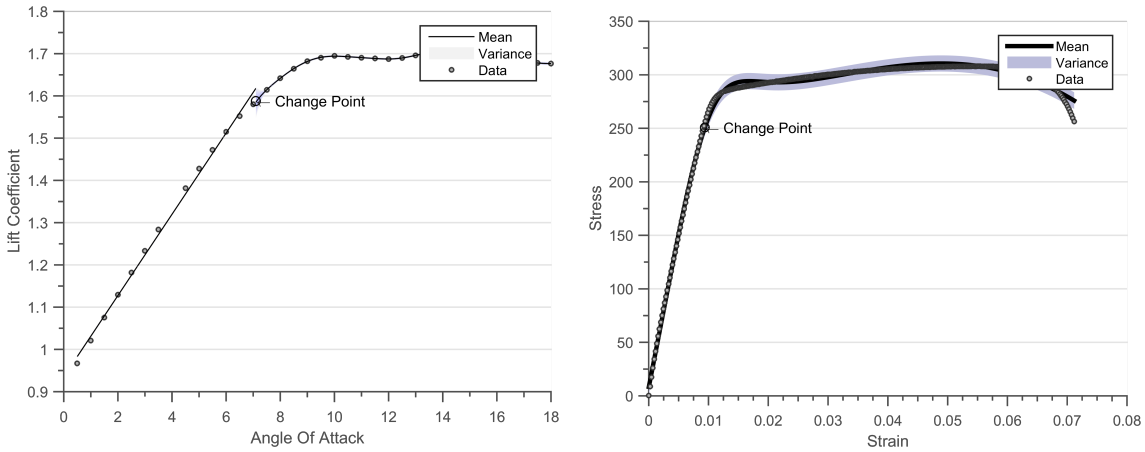
Table 5.1: Comparison of Young's Modulus for Al 6061 data-set

Figure 5.3 shows the posterior predictions when using the CP kernel. Figure 5.3(a) is the posterior distribution for the case of NACA 0012 airfoil, the Linear and SE regimes are plotted independently. Figure 5.3(b) shows the posterior distribution for the case of AL 6061. We can observe that the algorithm predicts a CP for the dataset, this is the point where the non-linear effects start dominating.

The marginal likelihood of a change-point kernel has many local minimas, there is a local minima at every observation point. This is because the kernel puts a non-linear regime at every observation point, hence a global optimizer should be used for optimization. The results of this study were be presented in the SIAM Uncertainty Quantification 2016 Conference [Chiplunkar 2016].

5.2 Multi-dimensional kernels

In this section we develop intuitions on how to build kernels for higher-dimensional inputs. This section demonstrates what happens when we Add or Multiply kernels across



(a) Posterior distribution for the case of NACA 0012 airfoil, the Linear and SE regimes are plotted independently.

(b) Posterior distribution for the case of AL 6061 alloy, the Linear and SE regimes are plotted independently.

Figure 5.3: Estimation of linear regimes using a change-point kernels

dimensions (equation 5.6 and 5.7), while also provide kernels on how to perform sensitivity analysis or to encode lower-dimensional structure (equation 5.4). We then apply the multi-dimensional covariance function to interpolate aerodynamic pressure (section 5.2.5), comparing the accuracy of GP interpolation with common Proper Orthogonal Decomposition technique [Chiplunkar 2017a]. Results of this exercise were used in a recent Airbus Flight Test campaign.

5.2.1 Adding across dimensions

Consider an input dataset which is multi-dimensional $x \in \mathbb{R}^{D_{inputs}}$. A simple additive kernel can be constructed by adding the kernels for individual dimensions [Hastie 1990]. This operation encodes the information that added dimensions are independent of each other (equation 5.15).

$$k(\tau, \theta) = \sum_{i=1}^{D_{inputs}} (\theta^i_{amplitude})^2 exp\left[-\frac{(\tau^i)^2}{2(\theta^i_{lengthScale})^2}\right] \tag{5.15}$$

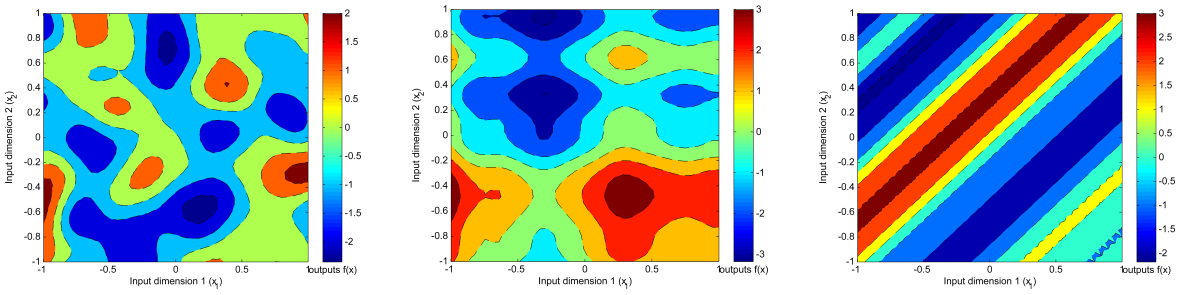
Here, $\tau^i = x^i_1 - x^i_2$ is the distance between two input points at the i^{th} dimension. Figure 5.4(b) is a randomly drawn function after adding two SE kernels, the hyper-parameters of both the SE kernels are $\theta_{amplitude} = 1$ and $\theta_{lengthscale} = 0.2$.

5.2.2 Multiplying across dimensions

If we want to include interactions between two dimensions then their kernels can be multiplied together (equation 5.7). A kernel which allows for interaction between all the possible D_{inputs}-dimensions can be constructed by multiplying all kernels for all the dimensions. The multi-dimensional Automatic Relevance Determination (ARD) kernel (equation 5.16) can be looked as a multiplication of several one-dimensional kernels with different length-scales [Rasmussen 2005]. It is called ARD because the value of length-scale determines which dimensions are more relevant.

$$k(\tau, \theta) = (\theta_{amplitude})^2 \prod_{i=1}^{D_{inputs}} exp\left[-\frac{(\tau^i)^2}{2(\theta^i_{lengthScale})^2}\right] \tag{5.16}$$

Figure 5.4(a) is obtained after multiplying 2 SE kernels, the hyper-parameters of both the SE kernels are $\theta_{amplitude} = 1$ and $\theta_{lengthscale} = 0.2$



(a) Random draw from a 2 dimensional prior obtained after **multiplying** two SE kernels. The hyper-parameters of both the SE kernels are $\theta_{amplitude} = 1$ and $\theta_{lengthscale} = 0.2$

(b) Random draw from a 2 dimensional prior obtained after **adding** two SE kernels. The hyper-parameters of both the SE kernels are $\theta_{amplitude} = 1$ and $\theta_{lengthscale} = 1$

(c) Random draw from a 2 dimensional prior which encodes a **low-dimensional** structure. The hyper-parameters of the kernel are $\theta_{amplitude} = 1$ and $\theta_{lengthscale} = 1$

Figure 5.4: Random draw from a 2 dimensional prior.

Matérn kernels have been found to have superior performance on datasets with high-dimensions [Le 2013]. It is argued that the Matérn kernel accounts for the concentration of measure effect in higher-dimensions. Imagine a high-dimensional orange (it is tough to imagine more than 3 dimensions), but if there was a high dimensional orange than most of its mass will be concentrated in its skin and not its pulp [Domingos 2012]. Since GP assigns use this measure of distance to learn patterns, we will use Matérn kernel in high-dimensions to perform regression.

5.2.3 Sensitivity analysis

[Duvenaud 2011, Durrande 2013, Chastaing 2015] define a class of additive kernels which are formed upon adding several low-dimensional interactions. The equation 5.17 is the basic form of covariance function which can be used to perform sensitivity analysis.

$$k(x_1, x_2) = (\theta)^2 \prod_{i=1}^{D_{inputs}} \left(1 + k^i(x_1^i, x_2^i)\right) \tag{5.17}$$

The above kernel will include all the possible interactions across dimensions. For example for a 3-dimensional input space, the above kernel will include all the first order terms (equation 5.18), all the second order terms (equation 5.20), and the third order term (equation 5.21).

$$\begin{aligned}
k_{first-order}(x_1, x_2) =& k^1(x_1^1, x_2^1) + k^2(x_1^2, x_2^2) + k^3(x_1^3, x_2^3) && (5.18)\\
k_{second-order}(x_1, x_2) =& k^1(x_1^1, x_2^1) \times k^2(x_1^2, x_2^2) + k^2(x_1^2, x_2^2) \times k^3(x_1^3, x_2^3) && (5.19)\\
&+ k^3(x_1^3, x_2^3) \times k^1(x_1^1, x_2^1) && (5.20)\\
k_{third-order}(x_1, x_2) =& k^1(x_1^1, x_2^1) \times k^2(x_1^2, x_2^2) \times k^3(x_1^3, x_2^3) && (5.21)\\
& && (5.22)
\end{aligned}$$

These kinds of kernels can be used to analyze the sensitivity of interactions between various dimensions (ANOVA).

5.2.4 Low dimensional structure

We can encode a low dimensional structure into family of functions by specifying the kernel as $k_{low} = k(x_1 H, x_2 H)$ (equation 5.4), here H is a low rank matrix.

$$k_{low}(\tau, \theta) = (\theta_{amplitude})^2 exp\left[-\frac{1}{2}\tau\Sigma\tau^T\right] \tag{5.23}$$

Here, Σ is HH^T which encodes the low dimensional structure. Figure 5.4(c) is obtained after encoding a low-dimensional into SE kernels, the hyper-parameters of the SE kernel are $\theta_{amplitude} = 1$ and $\theta_{lengthscale} = 0.2$ while $\Sigma = [1, 0; -1, 0]$. When Σ is a diagonal matrix

we get a ARD kernel

$$k(\tau,\theta) = (\theta_{amplitude})^2 exp\left[-\frac{1}{2}\tau\Sigma\tau^T\right] = (\theta_{amplitude})^2 exp\left[\sum_{i=1}^{D_{inputs}} -\frac{(\tau^i)^2}{2(\theta^i_{lengthScale})^2}\right]$$
$$= (\theta_{amplitude})^2 \prod_{i=1}^{D_{inputs}} exp\left[-\frac{(\tau^i)^2}{2(\theta^i_{lengthScale})^2}\right] \tag{5.24}$$

When the number of dimension increases significantly, the number of hyper-parameters also increases this makes the optimization of marginal likelihood inefficient. [Bouhlel 2016] first reduce the dimensionality of the dataset and then perform interpolation thereby circumventing the problem of higher dimensions. [Garnett 2013, Tripathy 2016] use this covariance to reduce the dimensionality of input domain.

In the current section we have seen how to build covariance functions for multi-dimensional inputs. We now apply multi-dimensional kernels to build a surrogate model for Aerodynamic pressures. We validate our method on 2 testcases: the first in subsonic regime on a Flap Track Fairing (FTF) and the second in transonic regime on NASA's Common Research Model (CRM) Wing. The results of these experiments were used during a recent Airbus Flight test campaign.

5.2.5 Application: Interpolation of aerodynamic pressures

Accurate prediction of aerodynamic pressures at a flight configuration is computationally expensive. Hence, it becomes advantageous to use surrogate models as approximations of high-fidelity aerodynamic models. A popular method of surrogate modelling in the aerodynamics community is by interpolating Reduced Order Models (ROM). A set of aerodynamic pressure snapshots is generated by performing CFD simulations for different aerodynamic parameters (eg. angle of attack α, Mach). Then orthogonal basis vectors are found in the parameter space for the set of pressures snapshots. Generally, Proper Orthogonal Decomposition (POD) [Tan 2003, Rosenbaum 2013, Braconnier 2011] (also called as Principal Component Analysis (PCA) or Singular Value Decomposition (SVD)) is used to find the linear subspace. Finally, the reduced models are interpolated at the desired point in the parameter-space [Beckert 2001, Barrault 2004].

Proper Orthogonal Decomposition Let us first start by defining a pressure snapshot. There exists a 3 dimensional spatial vector $\omega_i \in \mathbb{R}^3$ such that $\omega_i = \{(\omega_i^1, \omega_i^2, \omega_i^3)\}$. Here, $i \in [1, N_{nodes}]$ are the spatial coordinates of the i^{th} pressure node in a CFD mesh containing N_{nodes} pressure nodes. Similarly there exists a D dimensional parameter vector $d_j \in \mathbb{R}^D$,

for $d_j = \{(d_j^1, d_j^2, \ldots, d_j^D)\}$. Here, $j \in [1, N_{parameter}]$ correspond to the j^{th} parameter set. The parameters can be any non-spatial parameter which are desired to be interpolated, some common examples include Mach, Angle of Attack for steady aerodynamics and time or frequency for unsteady aerodynamics. We will only concentrate on interpolating steady aerodynamics in this section.

The pressure measured on the i^{th} pressure node for the j^{th} parameter set will be denoted as $p_j(\omega_i)$ defined by the equation 5.25. We next define the matrix $\Omega = \{\omega_1; \omega_2; \ldots; \omega_{N_{nodes}}\}$ for $\Omega \in \mathbb{R}^{N_{nodes} \times 3}$ containing the full spatial information of the CFD mesh. Finally, the pressure snapshot for the CFD run j will be denoted as $P_j(\Omega) = \{p_j(\omega_1); p_j(\omega_2); \ldots; p_j(\omega_{N_{nodes}})\}$ for $P_j(\Omega) \in \mathbb{R}^{N_{nodes}}$ defined by the equation 5.26.

$$p_j(\omega_i) = CFD(\omega_i, d_j) \tag{5.25}$$

$$P_j(\Omega) = CFD(\Omega, d_j) \tag{5.26}$$

The POD methodology decomposes the set of pressure snapshots ($P_j(\Omega)$) into their eigen vectors ($\phi^l(\Omega)$) and participation factors ($a^l(d_j)$). To reconstruct the pressure snapshot at a new point d_{new}, the participation factors are interpolated and then linearly combined to give the new pressure snapshot (equation 5.27). For more details please refer to appendix **reference to the appendix**

$$P_{new}(\Omega) = \sum_{l=1}^{p} a^l(d_{new})\phi^l(\Omega) \tag{5.27}$$

Contribution Due to the assumption of linear subspace, interpolation through ROM is highly efficient both in terms of cost and performance in the subsonic regime [Verveld 2016]. Unfortunately in the transonic regime, the shock creates a highly non-linear, almost discontinuous pressure distribution and the assumption of linear subspace does not hold [Li 2016]. Although the performance of ROM interpolators can be improved with larger number of samples [Franz 2014, Forrester 2008], we propose to improve the accuracy of prediction using distributed GP Regression for the same number of samples.

We interpolate the pressure $p_j(\omega_i)$ by simply multiplying the kernels across dimensions (equation 5.28). To interpolate in subsonic regime we multiply Matérn $\nu = 5/2$ across dimensions, while to interpolate in transonic regime we use the Neural Network

kernel in the dimension of shock.

$$\Pr[p_j(\omega_i)] = GP\left(0, k(x_1, x_2 = \prod_{i=1}^{D_{inputs}} k(x_1^i, x_2^i)\right) \tag{5.28}$$

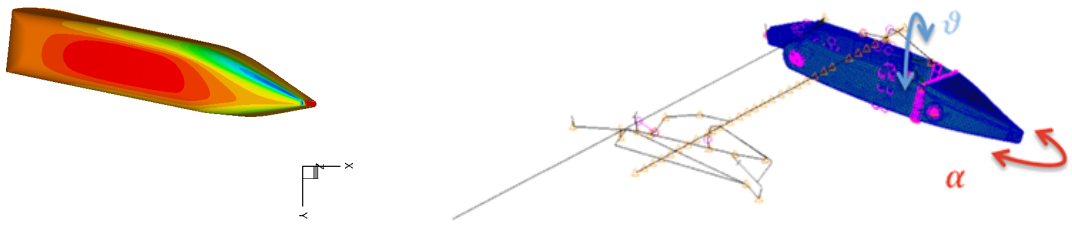
We now test the performance of distributed GP and POD+I on two sets of numerical experiments. Firstly, we test the accuracy on a detailed FTF design [Bosco 2016b] in subsonic regime (section 5.2.5) based on simulation from the elsA code [Cambier 2008]. Finally, we compare the accuracy on CRM wing in the transonic regime using elsA solver and kOmega-SST turbulence model [Vassberg 2014].

elsA® [Cambier 2008] is a pluri-function CFD simulation platform that allows representation of internal and external aerodynamics from the low subsonic to the high supersonic flow regime. Several formulations of the 3D Navier-Stokes equations can be chosen for arbitrary moving bodies.

Interpolation in subsonic regime

A Flap Track Fairing (FTF) is situated below the wing and is used to deploy flaps for landing and take-off configurations. A FTF experiences heavy dynamic excitation due to the exhaust coming from engine. The dynamic nature makes the design of FTF a challenging task, where each simulation can last for 2 days. If we can effectively interpolate pressure snapshots then a 2 day dynamic simulation can be reduced to a few hours.

Interpolation of FTF pressure snapshots has been earlier studied using POD methodology [Bosco 2016a]. In this section we use this dataset to validate the interpolation capabilities of a distributed GP.



(a) Flap Track Fairing Pressure snapshot

(b) Flap Track Fairing DoF

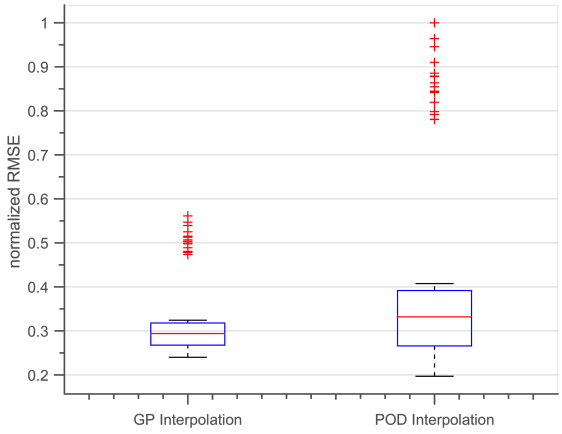
Figure 5.5: Details of the Flap track Fairing

The FTF degree of freedom chosen for this analysis are θ, the rotation around the longitudinal axis of the FTF, and α, the rotation around the *spigot* axis, main connection

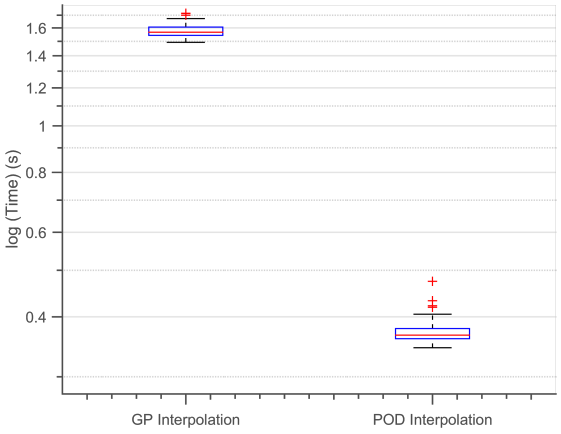
between the track and the wing structure figure 5.5(b). The surface mesh of the CFD skin contains almost 36k nodes more precisely $N_{nodes} = 36802$. We run the simulation for 9 different values of α and 9 different values of θ, more precisely we have run the CFD-Rans computation for $N_{parameter} = 9 \times 9 = 81$ number of times. In this particular case Reynolds Averaged Navier-Stokes, *RANS*, equations are used with Spalart-Allmaras turbulence model to simulate the flow around the FTF. Figure 5.5(a) shows one particular pressure snapshot.

We use Leave One Out (LOO) Cross Validation method to quantify the performance of the two methodologies. $[\alpha, \theta]$ pairs are removed one by one from the database to create a new training set. The new training set is used to perform interpolation according to POD+I and distributed GP. Pressure snapshot is reconstructed for the missing $[\alpha, \theta]$ pairs. The methods are finally compared by evaluating the Root Mean Square Error (RMSE) and time of prediction for each pair case.

While performing distributed GP regression, we learn a GP model between p_{ij} and input vector $X = [\omega_i^1, \omega_i^2, \omega_i^3, \alpha, \theta]$. We build a multi-dimensional kernel by multiplying 5 Matérn kernels, different length-scale for each dimension. As discussed earlier we propose to use Matérn kernel since the SE kernel has a very restrictive hypothesis space for high-dimensional regression. Effectively we are learning a GP model for $N = 36802 \times 80 = 2.9$ million data-points.



(a) Box plots of normalized RMSE for the two different model types. POD has a RMSE of 0.48 ± 0.27 whereas distributed GP has a RMSE of 0.37 ± 0.1. The mean distributed GP prediction is 20% better than POD. Outliers are extrapolation cases



(b) Time taken to perform prediction for the two different model types. POD takes $2.35s \pm 0.11s$ whereas distributed GP takes $37.84s \pm 4.35s$ to perform the interpolation. The average POD method is 19 times faster than distributed GP.

Figure 5.6: Results for elsA interpolation

Figures 5.6(a) denote the RMSE estimates for different pairs. For RMSE, the distributed GP performs marginally better than POD. The outliers in the boxplots denote

cases where the removed pairs are on the edges of database. Since the removed pairs are on the edges of database matrix, there are no CFD snapshots surrounding these pairs. Hence extrapolation is performed during the edge cases. POD has a RMSE of 0.48 ± 0.27 whereas distributed GP has a RMSE of 0.37 ± 0.1. The average distributed GP prediction is 20% better than POD. Figure 5.6(b) shows the time taken to perform prediction for the two model types. POD takes $2.35s \pm 0.11s$ whereas distributed GP takes $37.84s \pm 4.35s$ to perform the interpolation. The POD method is the faster sometimes performing 30 times faster than the distributed GP.

Although the GP technique can more efficiently capture non-linear behavior we see a relatively low improvement in performance for the amount of time invested. Deciding between the simple and time-tested POD method or costly and accurate distributed GP interpolation in the subsonic regime can be a tough task and mostly depends on the preferences of the final user.

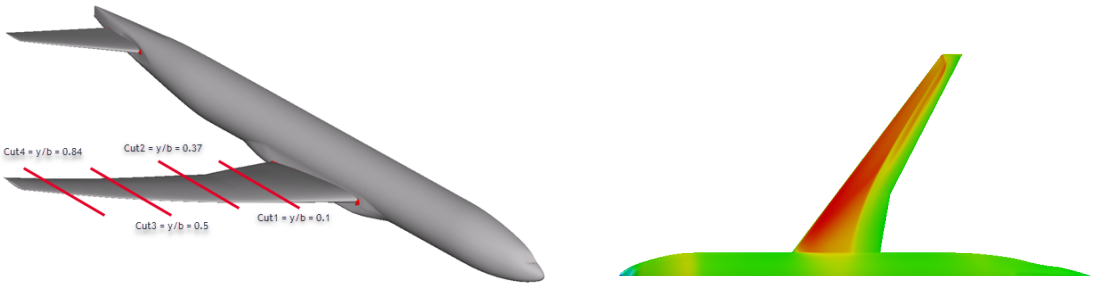
Interpolating in transonic regime

We now proceed to compare the accuracy of the two methods in transonic regime on the Common Research Model (CRM) proposed by NASA. Since the introduction of the model for the 4th Drag Prediction Workshop, the CRM has been become a very widely used test case for applied computational aerodynamics. Due to the widespread experience and availability of wind-tunnel test results for the CRM configuration, it is a natural case to benchmark interpolations.

Due to the shape of an airfoil, airflow is accelerated on the upper surface of the wing. This causes shocks to appear on the upper surface of the wing in the transonic regime. Shocks are sudden changes (almost discontinuous) in the pressure and are important for estimating performance of the aircraft. Moreover an aircraft flies in the transonic regime for 80% of its journey (cruise). Hence accurate prediction of location and strength of a shock is very important during design. Since POD is a linear subspace reduction method it has difficulty while a discontinous reconstructing shock regime.

Again we use the elsA® solver to perform simulations on the design. We use the $\kappa - \omega$ SST turbulence model to perform predictions since it has good performance in the fuselage wing interaction regions [Menter 2003, Vassberg 2014]. The CFD was run for a combination of 21 values of $\alpha \in [1 : 0.1 : 3]$ and 5 values of $Mach \in [0.84 : 0.005 : 0.86]$, hence $N_{parameter} = 21 \times 5 = 105$. Figure, 5.7(b) shows one of the pressure snapshots for $\alpha = 2$ and $Mach = 0.85$. We then cut the wing at four distinct locations $y/b = [0.105, 0.37, 0.5, 0.84]$ (figure 5.7(a)) to clearly observe different types of aerodynamic behavior. Here, y denotes the y-distance from aircraft axis and b denotes the span of one wing.

As detailed in the earlier section we again use LOO-CV for comparing the performance of the two methods. The POD+I method has been run as described in Appendix **reference**



(a) Common research model. The four red lines are the four cuts at $y/b = [0.105, 0.37, 0.5, 0.84]$. Here, y denotes the y-distance from aircraft axis and b denotes the span of one wing.

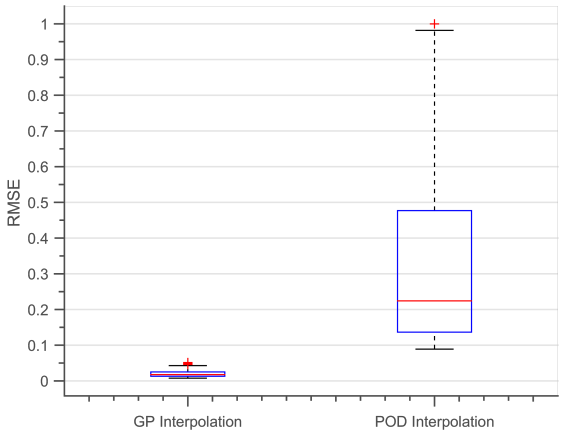
(b) Pressure snapshot on the Common Research model for $\alpha = 2$ and $Mach = 0.85$. We can observe double shock pattern appearing on the outer sections of the wing

Figure 5.7: Common Research Model

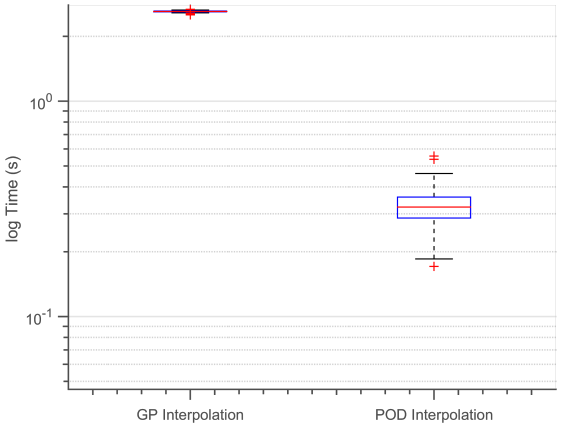
to appendix on PODi. For GP regression, we learn a GP model between p_{ij} and input vector $X = [chordDimension, \alpha, Mach]$. We build a multi-dimensional kernel by multiplying 2 Matérn kernels (for α and $Mach$ dimensions) one Neural Network kernel (for spatial dimension). The shock will appear in the spatial dimension and hence using a Neural network kernel in that dimension lets us capture the discontinuity more accurately.

The above figures 5.8(a) denote the RMSE estimates for different pairs. POD has a RMSE of 0.32 ± 0.23 whereas distributed GP has a RMSE of 0.02 ± 0.01. The average distributed GP prediction is 16 times better than POD in transonic regime. The outliers in the boxplots denote cases where the removed pairs are on the edges of database. Since the removed pairs are on the edges of database matrix, there are no CFD snapshots surrounding these pairs. Hence extrapolation is performed during the edge cases. Figure 5.8(b) shows the time taken to perform prediction for the three different model types. POD takes $0.5s \pm 0.03s$ whereas distributed GP takes $40.6s \pm 2.3s$ to perform the interpolation. In transonic regime GP has a significantly better error performance and becomes the obvious choice for interpolation.

Figure 5.9(a) shows the comparison between POD method and distributed GP for interpolation. Reconstruction is performed on the pressure snapshot at $\alpha = 2$ and $Mach = 0.85$ for the $y/b = 0.105$. We can observe that the shape of shock has been smoothed out by POD method. Figure 5.9(b) shows comparison between POD method and distributed GP for extrapolation. Reconstruction is performed on the hidden pressure snapshot of $\alpha = 2$ and $Mach = 0.84$ which is an extrapolation case. We can observe that POD introduces errors both for the intensity of shock, and location of shock for the extrapolation case, this explains the high amount of error in figure 5.8(a).



(a) Normalized RMSE for the two different model types. POD has a RMSE of 0.32 ± 0.23 whereas distributed GP has a RMSE of 0.02 ± 0.01. The average distributed GP prediction is 16 times better than POD in transonic regime. The outliers in the boxplots denote cases where the removed pairs are on the edges of database. Since the removed pairs are on the edges of database matrix, there are no CFD snapshots surrounding these pairs. Hence extrapolation is performed during the edge cases.



(b) Time taken to perform prediction for the two different model types. POD takes $0.5s \pm 0.03s$ whereas distributed GP takes $40.6s \pm 2.3s$ to perform the interpolation. The average POD method is 80 times faster than distributed GP. Here we are interpolating the pressure on the whole wing.

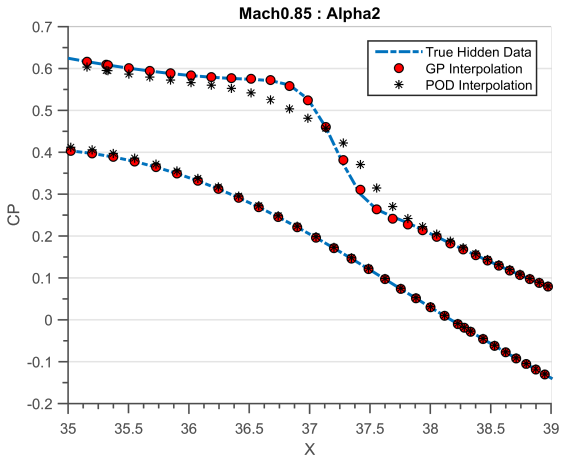
Figure 5.8: Results for CRM interpolation

Comparison across cuts

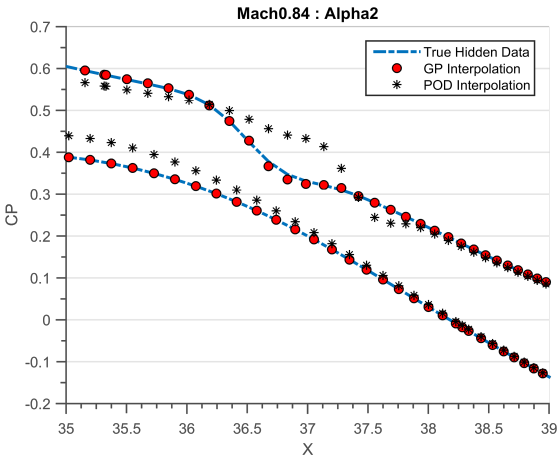
We next study the accuracy of distributed GP for different airfoils on the wing. Using the methodology described earlier we build a distributed GP model for each airfoil and measure the accuracy of interpolation performed for each cut using the LOO-CV methodology.

Figure 5.10(a) shows the RMSE performance across cuts. The performance of interpolation deteriorates as we go further away from the fuselage. This is primarily because as we go further away from the fuselage double shocks start appearing on the airfoil as observable from figure 5.7(b). Figure 5.10(b) shows interpolation performed by the POD method and distributed GP methods at $\alpha = 2$ and $Mach = 0.85$. While, POD smooths out the double shock pattern distributed GP also lacks the accuracy observed in figure 5.9(a).

Figures 5.11(a) and 5.11(b) show the evaluation of pressures upon varying $\alpha \in [1, 3]$ at locations $y/b = 0.105$ and $y/b = 0.84$ respectively. The color coding denotes coefficient of pressure for upper side of airfoil, The x-axis denotes chord-wise location and y-axis denotes α. White lines denote presence of a pressure snapshot due to CFD run, everything in between is interpolation. Dashed black lines denote constant pressure contours Color



(a) Comparison between POD method and distributed GP for **interpolation**. The X-axis denotes chord dimension, only showing chord-section near shock for clarity. The Y-axis denotes the coefficient of pressure. Reconstruction is performed on the pressure snapshot at $\alpha = 2$ and $Mach = 0.85$ for the $y/b = 0.105$. We can observe that the intensity of shock has been smoothed out by POD method



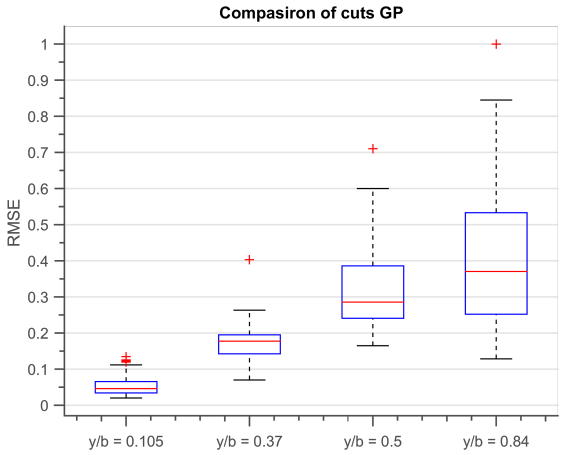
(b) Comparison between POD method and distributed GP for **extrapolation**. The X-axis denotes chord dimension, only showing chord-section near shock for clarity. The Y-axis denotes the coefficient of pressure. Reconstruction is performed on the pressure snapshot at $\alpha = 2$ and $Mach = 0.84$ for the $y/b = 0.105$. We can observe that POD introduces errors both for the intensity of shock and location of shock for this case

Figure 5.9: Comparison of pressure interpolations for first cut $y/b = 0.105$. Here we compare the accuracy of prediction for interpolation and extrapolation cases.

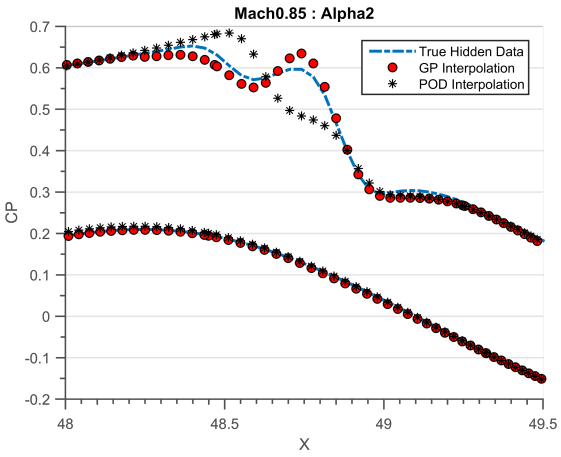
between two contours has been smoothed for clarity. For figure 5.11(a) we observe a strong shock near $\alpha = 3$ which slowly gets converted to a weak shock near $\alpha = 1$. The presence of a single shock is also the reason why distributed GP performs better at this cut location. For figure 5.11(b) we observe a single shock near $\alpha = 3$ which slowly gets converted to a double shock pattern. The zone of from single to double shock is a very interesting point for performance, since the wing drag is minimum during this transition phase. Distributed GP starts performing badly near the transition phases, this can be observed by the small pools of $C_P = 0.5$ at the transition phase from single to double shock.

The current section presents a comparison between two different surrogate model building methods: time-tested surrogate modelling methods such as POD coupled with spline interpolation and upcoming machine learning methods such as distributed GP for subsonic and transonic regimes. The distributed GP performs marginally better than POD technique in subsonic regime but is a many times slower. On the contrary for the case of transonic regime 5.2.5 distributed GP clearly outperforms POD+I method. This is mainly due to the presence of shock on the airfoil.

Plots like figure 5.11(b) give a quick understanding of the flight regime for very few



(a) Normalized RMSE for different airfoils based on distributed GP. The mean RMSE of different cuts from $y/b = [0.105, 0.37, 0.5, 0.84]$ is $[0.053, 0.17, 0.31, 0.40]$ respectively. The accuracy of interpolation deteriorates as we go farther away from the fuselage. This is due to appearance of double shock on the outer section of wing.



(b) Comparison between POD method and distributed GP for interpolation. The X-axis denotes chord dimension, only showing chord-section near shock for clarity. The Y-axis denotes the coefficient of pressure. Reconstruction is performed on the pressure snapshot at $\alpha = 2$ and $Mach = 0.85$ for the location $y/b = 0.84$. While POD smooths out the double shock pattern distributed GP also lacks the accuracy observed in figure 5.9(a).

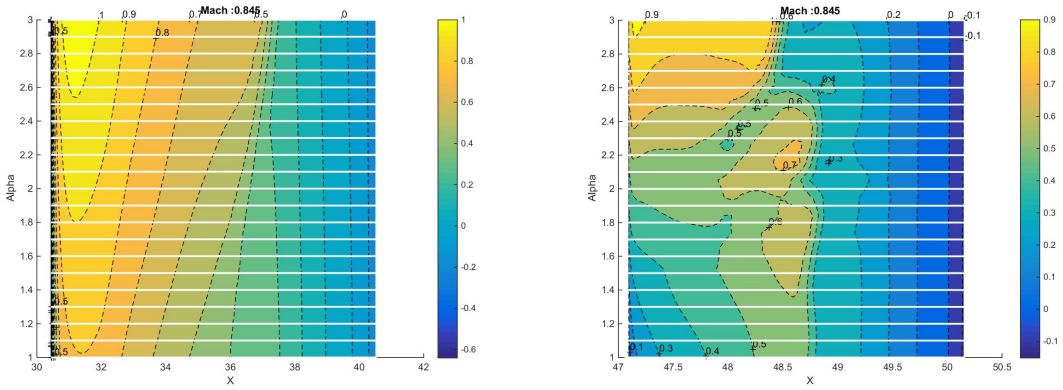
Figure 5.10: Performance of distributed GP across cuts

simulation runs. These plots can also be used to find transition regimes from single shock to double shock, which are very interesting performance points. In the future we wish to improve reconstruction of double shock patterns by improving the length scale and improve the choice of experts for distributed GP.

5.3 Discussion

In the last two chapters we have tried to answer the question: How to add *apriori* information of a pattern in a learning algorithm? We present a small sneak peek into the wide variety of available covariance functions. Due to the ability to create new kernels, encoding prior information into the structure can be performed easily. If we have an *apriori* information about the pattern of function to be learned, then embedding this information into the GP algorithm greatly improves accuracy and cost of prediction. Table 4.4 lists a few commonly known combination of covariance functions known in the literature.

In the next two chapters we will tackle the remaining two questions posed in section ?? of the thesis. Chapter ?? discusses how to encode prior information of relationships



(a) Interpolation performed at constant $Mach = 0.845$ and $\alpha = [1, 3]$ for the location $y/b = 0.105$. The color coding denotes coefficient of pressure for upper side of airfoil, The x-axis denotes chord-wise location and y-axis denotes α. White lines denote presence of a pressure snapshot due to CFD run, everything in between is interpolation. Dashed black lines denote constant pressure contours color between two contours has been smoothed for clarity. We observe a strong shock near $\alpha = 3$ which slowly gets converted to a weak shock near $\alpha = 1$.

(b) Interpolation performed at constant $Mach = 0.845$ and $\alpha = [1, 3]$ for the location $y/b = 0.84$. The color coding denotes coefficient of pressure for upper half of airfoil, The x-axis denotes chord-wise location and y-axis denotes α. White lines denote presence of a pressure snapshot due to CFD run, everything in between is interpolation. Dashed black lines denote constant pressure contours color between two contours has been smoothed for clarity. We observe a single shock near $\alpha = 3$ which slowly gets converted to a double shock pattern.

Figure 5.11: Pressure reconstructions for constant $Mach = 0.845$ and sweeping $\alpha \in [1, 3]$

Model	Structure	Citation
Linear Regression	$k_{constant} + k_{linear} + k_{noise}$	
Polynomial	$k_{constant} + \prod k_{linear} + k_{noise}$	
Ordinary kriging	$k_{SE} + k_{noise}$	[Krige 1951]
Simple kriging	$k_{constant} + k_{SE} + k_{noise}$	
Universal Kriging	$\prod k_{linear} + k_{SE} + k_{noise}$	[Matheron 1963]
Multiple Kernel	$\sum k_{SE} + k_{noise}$	
Spectral Mixture	$\sum cosk_{SE} + k_{noise}]$	[Wilson 2013]
Change point	$\sum CP(k_{Lin}, k_{SE}) + k_{noise}]$	[Osborne 2010]
Additive GPs	$\prod_i (1 + k^i_{SE})]$	[Duvenaud 2011]

Table 5.2: Combination of covariance functions available in literature

between measurements into a GP regression. While chapter **??** discusses how to perform extrapolation given a computer simulation of experiments.

Part III

Incorporating multiple outputs Gaussian Process Regression

Bibliography

[Allemang 1998] Randall J Allemang et DL Brown. *A unified matrix polynomial approach to modal identification.* Journal of Sound and Vibration, vol. 211, no. 3, pages 301–322, 1998.

[Barrault 2004] Maxime Barrault, Yvon Maday, Ngoc Cuong Nguyen et Anthony T Patera. *An 'empirical interpolation'method: application to efficient reduced-basis discretization of partial differential equations.* Comptes Rendus Mathematique, vol. 339, no. 9, pages 667–672, 2004.

[Bauer 1998] Eric Bauer et Ron Kohavi. *An empirical comparison of voting classification algorithms: Bagging, boosting, and variants.* Machine learning, vol. 36, no. 1, page 2, 1998.

[Beckert 2001] Armin Beckert et Holger Wendland. *Multivariate interpolation for fluid-structure-interaction problems using radial basis functions.* Aerospace Science and Technology, vol. 5, no. 2, pages 125–134, 2001.

[Bishop 2006] Christopher M Bishop. *Pattern Recognition.* Machine Learning, 2006.

[Bochner 1959] S. Bochner, M. Tenenbaum et H. Pollard. Lectures on fourier integrals. Annals of mathematics studies. Princeton University Press, 1959.

[Bochner 2016] Salomon Bochner. Lectures on fourier integrals.(am-42), volume 42. Princeton University Press, 2016.

[Bosco 2016a] Elisa Bosco, Albert Lucchetti, Simon Trapier, Fausto Gill Di Vincenzo, N Gourdain et J Morlier. *Nonlinear transient Fluid/Structure interaction approach using surrogate models: Industrial application to aircraft fairing vibration excited by engine jet flow.* In 15th Dynamics Specialists Conference, page 2049, 2016.

[Bosco 2016b] Elisa Bosco, Albert Lucchetti, Simon Trapier, Fausto Gill Di Vincenzo, Joseph Morlier et Nicolas Gourdain. Nonlinear transient fluid/structure interaction approach using surrogate models: Industrial application to aircraft fairing vibration

excited by engine efflux. AIAA SciTech Forum. American Institute of Aeronautics and Astronautics, Jan 2016. 0.

[Bouhlel 2016] Mohamed Amine Bouhlel. *Optimisation auto-adaptative en environnement d'analyse multidisciplinaire via les modèles de krigeage combinés à la méthode PLS*. PhD thesis, Toulouse, ISAE, 2016.

[Braconnier 2011] T Braconnier, M Ferrier, J-C Jouhaud, M Montagnac et P Sagaut. *Towards an adaptive POD/SVD surrogate model for aeronautic design*. Computers & Fluids, vol. 40, no. 1, pages 195–209, 2011.

[Brincker 2000] Rune Brincker, Lingmi Zhang et P Andersen. *Modal identification from ambient responses using frequency domain decomposition*. In Proc. of the 18*'International Modal Analysis Conference (IMAC), San Antonio, Texas, 2000.

[Cambier 2008] L Cambier et JP Veuillot. *Status of the elsA CFD software for flow simulation and multidisciplinary applications*. AIAA paper, vol. 664, page 2008, 2008.

[Cao 2013] Yanshuai Cao, Marcus A Brubaker, David J Fleet et Aaron Hertzmann. *Efficient optimization for sparse gaussian process regression*. In Advances in Neural Information Processing Systems, pages 1097–1105, 2013.

[Cao 2014] Yanshuai Cao et David J Fleet. *Generalized product of experts for automatic and principled fusion of Gaussian process predictions*. arXiv preprint arXiv:1410.7827, 2014.

[Chastaing 2015] Gaëlle Chastaing et Loic Le Gratiet. *ANOVA decomposition of conditional Gaussian processes for sensitivity analysis with dependent inputs*. Journal of Statistical Computation and Simulation, vol. 85, no. 11, pages 2164–2186, 2015.

[Chauhan 2007] S Chauhan, R Martell, RJ Allemang et DL Brown. *Unified matrix polynomial approach for operational modal analysis*. In Proceedings of the 25th IMAC, Orlando (FL), USA, 2007.

[Chen 2009] Tao Chen et Jianghong Ren. *Bagging for Gaussian process regression*. Neurocomputing, vol. 72, no. 7, pages 1605–1610, 2009.

[Chenhan 2016] D Yu Chenhan, William B March, Bo Xiao et George Biros. *INV-ASKIT: A Parallel Fast Direct Solver for Kernel Matrices*. In Parallel and Distributed Processing Symposium, 2016 IEEE International, pages 161–171. IEEE, 2016.

[Chiplunkar 2016] Ankit Chiplunkar, Emmanuel Rachelson, Michele Colombo et Joseph Morlier. *Identification of Physical Parameters Using Change-Point Kernels*. Society for Industrial and Applied Mathematics, Uncertainty Quantification, 2016, Avril 2016. Poster.

[Chiplunkar 2017a] Ankit Chiplunkar, Elisa Bosco et Joseph Morlier. *Gaussian Process for Aerodynamic Pressures Prediction in Fast Fluid Structure Interaction Simulations.* In 12th World Congress on Structural and Multidisciplinary Optimization, Braunschweig, DE, 2017.

[Chiplunkar 2017b] Ankit Chiplunkar et Joseph Morlier. *Operational Modal Analysis in Frequency Domain Using Gaussian Mixture Models.* In Topics in Modal Analysis & Testing, Volume 10, pages 47–53. Springer, 2017.

[Clapeyron 1834] B.P.É. Clapeyron. Mémoire sur la puissance motrice de la chaleur. 1834.

[Cornford 2002] Dan Cornford, Ian T Nabney et Christopher KI Williams. *Modelling frontal discontinuities in wind fields.* Journal of nonparametric statistics, vol. 14, no. 1-2, pages 43–58, 2002.

[Cox 1977] David Roxbee Cox et Hilton David Miller. The theory of stochastic processes, volume 134. CRC Press, 1977.

[Cummings 2015] Russell M Cummings, William H Mason, Scott A Morton et David R McDaniel. Applied computational aerodynamics: A modern engineering approach, volume 53. Cambridge University Press, 2015.

[Deisenroth 2015] Marc Peter Deisenroth et Jun Wei Ng. *Distributed Gaussian Processes.* arXiv preprint arXiv:1502.02843, 2015.

[Domingos 2012] Pedro Domingos. *A few useful things to know about machine learning.* Communications of the ACM, vol. 55, no. 10, pages 78–87, 2012.

[Domingos 2015] P. Domingos. The master algorithm: How the quest for the ultimate learning machine will remake our world. Basic Books. Basic Books, 2015.

[Dubrule 1983] Olivier Dubrule. *Cross validation of kriging in a unique neighborhood.* Mathematical Geology, vol. 15, no. 6, pages 687–699, 1983.

[Durrande 2001] Nicolas Durrande. *Étude de classes de noyaux adaptées à la simplification et à l'interprétation des modèles d'approximation. Une approche fonctionnelle et probabiliste.* PhD thesis, Ecole Nationale Supérieure des Mines de Saint-Etienne, 2001.

[Durrande 2011] Nicolas Durrande, David Ginsbourger, Olivier Roustant et Laurent Carraro. *Additive covariance kernels for high-dimensional Gaussian process modeling.* arXiv preprint arXiv:1111.6233, 2011.

[Durrande 2013] Nicolas Durrande, David Ginsbourger, Olivier Roustant et Laurent Carraro. *ANOVA kernels and RKHS of zero mean functions for model-based sensitivity analysis.* Journal of Multivariate Analysis, vol. 115, pages 57–67, 2013.

[Duvenaud 2011] David K Duvenaud, Hannes Nickisch et Carl E Rasmussen. *Additive gaussian processes*. In Advances in neural information processing systems, pages 226–234, 2011.

[Duvenaud 2013] David Duvenaud, James Robert Lloyd, Roger Grosse, Joshua B Tenenbaum et Zoubin Ghahramani. *Structure discovery in nonparametric regression through compositional kernel search*. arXiv preprint arXiv:1302.4922, 2013.

[Duvenaud 2014] David Duvenaud. *Automatic Model Construction with Gaussian Processes*. PhD thesis, Computational and Biological Learning Laboratory, University of Cambridge, 2014.

[Findley 1991] David F Findley. *Counterexamples to parsimony and BIC*. Annals of the Institute of Statistical Mathematics, vol. 43, no. 3, pages 505–514, 1991.

[Forrester 2008] Alexander Forrester, Andras Sobester et Andy Keane. Engineering design via surrogate modelling: a practical guide. John Wiley & Sons, 2008.

[Fox 2004] Bernard Fox, Michael Boito, John C Graser et Obaid Younossi. *Test and evaluation trends and costs for aircraft and guided weapons*. Rapport technique, RAND CORP SANTA MONICA CA, 2004.

[Franz 2014] Thomas Franz, Ralf Zimmermann, Stefan Görtz et Niklas Karcher. *Interpolation-based reduced-order modelling for steady transonic flows via manifold learning*. International Journal of Computational Fluid Dynamics, vol. 28, no. 3-4, pages 106–121, 2014.

[Gal 2014] Yarin Gal, Mark van der Wilk et Carl E. Rasmussen. *Distributed Variational Inference in Sparse Gaussian Process Regression and Latent Variable Models*. arXiv:1402.1389, 2014.

[Garnett 2013] Roman Garnett, Michael A Osborne et Philipp Hennig. *Active learning of linear embeddings for Gaussian processes*. arXiv preprint arXiv:1310.6740, 2013.

[Ghahramani] Zoubin Ghahramani. *Automatic Statuistician*.

[Ghahramani 2013] Zoubin Ghahramani. *Bayesian non-parametrics and the probabilistic approach to modelling*. Phil. Trans. R. Soc. A, vol. 371, no. 1984, page 20110553, 2013.

[Goodfellow 2016] Ian Goodfellow, Yoshua Bengio et Aaron Courville. Deep learning. MIT Press, 2016. `http://www.deeplearningbook.org`.

[Guillaume 2003] Patrick Guillaume, Peter Verboven, Steve Vanlanduit, Herman Van Der Auweraer et Bart Peeters. *A poly-reference implementation of the least-squares complex frequency-domain estimator*. In Proceedings of IMAC, volume 21, pages 183–192, 2003.

[Hastie 1990] Trevor Hastie et Robert Tibshirani. Generalized additive models. Wiley Online Library, 1990.

[Ibrahim 1977] SR Ibrahim et EC Mikulcik. *A method for the direct identification of vibration parameters from the free response*. 1977.

[James III 1995] OH James III et TG Came. *THE NATURAL EXCITATION TECHNIQUE (NEXT) FOR MODAL PARANIETER EXTRACTION FROM OPERATING STRUCTURES*. 1995.

[Jameson 2012] Antony Jameson. *Computational Fluid Dynamics: Past, Present and Future*, 2012.

[Kaufman 1999] John Gilbert Kaufman. Properties of aluminum alloys: tensile, creep, and fatigue data at high and low temperatures. ASM international, 1999.

[Kennedy 2000] Marc C Kennedy et Anthony O'Hagan. *Predicting the output from a complex computer code when fast approximations are available*. Biometrika, vol. 87, no. 1, pages 1–13, 2000.

[Kleene 1951] Stephen Cole Kleene. *Representation of events in nerve nets and finite automata*. Rapport technique, RAND PROJECT AIR FORCE SANTA MONICA CA, 1951.

[Kostantinos 2000] N Kostantinos. *Gaussian mixtures and their applications to signal processing*. Advanced signal processing handbook: theory and implementation for radar, sonar, and medical imaging real time systems, pages 3–1, 2000.

[Krige 1951] Daniel G Krige. *A statistical approach to some basic mine valuation problems on the Witwatersrand*. Journal of the Southern African Institute of Mining and Metallurgy, vol. 52, no. 6, pages 119–139, 1951.

[Le Gratiet 2013] Loic Le Gratiet. *Multi-fidelity Gaussian process regression for computer experiments*. PhD thesis, Université Paris-Diderot-Paris VII, 2013.

[Le 2013] Quoc Le, Tamás Sarlós et Alexander Smola. *Fastfood-computing hilbert space expansions in loglinear time*. In Proceedings of the 30th International Conference on Machine Learning, pages 244–252, 2013.

[LeCun 2015] Yann LeCun, Yoshua Bengio et Geoffrey Hinton. *Deep learning.* Nature, vol. 521, no. 7553, pages 436–444, 2015.

[Li 2016] Jing Li et Weiwei Zhang. *The performance of proper orthogonal decomposition in discontinuous flows.* Theoretical and Applied Mechanics Letters, vol. 6, no. 5, pages 236–243, 2016.

[Lloyd 2014] James Robert Lloyd, David Duvenaud, Roger Grosse, Joshua B Tenenbaum et Zoubin Ghahramani. *Automatic construction and natural-language description of nonparametric regression models.* arXiv preprint arXiv:1402.4304, 2014.

[Loeve 1978] M. Loeve. Probability theory ii. F.W.Gehring P.r.Halmos and C.c.Moore. Springer, 1978.

[MacKay 2003] David JC MacKay. Information theory, inference and learning algorithms. Cambridge university press, 2003.

[March 2015] William B March, Bo Xiao et George Biros. *ASKIT: Approximate skeletonization kernel-independent treecode in high dimensions.* SIAM Journal on Scientific Computing, vol. 37, no. 2, pages A1089–A1110, 2015.

[Marszalek 2007] Marcin Marszalek et Cordelia Schmid. *Semantic hierarchies for visual object recognition.* In Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on, pages 1–7. IEEE, 2007.

[Matheron 1963] Georges Matheron. *Principles of geostatistics.* Economic geology, vol. 58, no. 8, pages 1246–1266, 1963.

[Menter 2003] FR Menter, M Kuntz et R Langtry. *Ten years of industrial experience with the SST turbulence model.* Turbulence, heat and mass transfer, vol. 4, no. 1, pages 625–632, 2003.

[Mercer 1909] James Mercer. *Functions of positive and negative type, and their connection with the theory of integral equations.* Philosophical transactions of the royal society of London. Series A, containing papers of a mathematical or physical character, vol. 209, pages 415–446, 1909.

[Murthy 2014] P Srinivasa Murthy. *Computational Fluid Dynamics: A Design Tool for Aircraft Industries.* In Innovative Design, Analysis and Development Practices in Aerospace and Automotive Engineering, pages 65–66. Springer, 2014.

[Neal 2011] Radford M Neal*et al.* *MCMC using Hamiltonian dynamics.* Handbook of Markov Chain Monte Carlo, vol. 2, no. 11, 2011.

[Neal 2012] Radford M Neal. Bayesian learning for neural networks, volume 118. Springer Science & Business Media, 2012.

[Ng 2014] Jun Wei Ng et Marc Peter Deisenroth. *Hierarchical Mixture-of-Experts Model for Large-Scale Gaussian Process Regression.* arXiv preprint arXiv:1412.3078, 2014.

[Osborne 2010] Michael Osborne. Bayesian gaussian processes for sequential prediction, optimisation and quadrature. Oxford University New College, 2010.

[Peeters 2005] Bart Peeters, Herman Van der Auweraer, Steven Pauwels et Jan Debille. *Industrial relevance of Operational Modal Analysis–civil, aerospace and automotive case histories.* In Proceedings of IOMAC, the 1st International Operational Modal Analysis Conference, 2005.

[Quiñonero-Candela 2005] Joaquin Quiñonero-Candela et Carl Edward Rasmussen. *A unifying view of sparse approximate Gaussian process regression.* Journal of Machine Learning Research, vol. 6, no. Dec, pages 1939–1959, 2005.

[Rainieri 2007] Carlo Rainieri, Giovanni Fabbrocino et E Cosenza. *Automated Operational Modal Analysis as structural health monitoring tool: theoretical and applicative aspects.* In Key Engineering Materials, volume 347, pages 479–484. Trans Tech Publ, 2007.

[Rasmussen 2002] Carl Edward Rasmussen et Zoubin Ghahramani. *Infinite mixtures of Gaussian process experts.* Advances in neural information processing systems, vol. 2, pages 881–888, 2002.

[Rasmussen 2005] Carl Edward Rasmussen et Christopher K. I. Williams. Gaussian processes for machine learning (adaptive computation and machine learning). The MIT Press, 2005.

[Rasmussen 2006] Carl Edward Rasmussen. *Gaussian processes for machine learning.* 2006.

[Raymer 2012] D.P. Raymer. Aircraft design: A conceptual approach. AIAA education series. American Institute of Aeronautics and Astronautics, 2012.

[Richardson 1982] Mark H Richardson et David L Formenti. *Parameter estimation from frequency response measurements using rational fraction polynomials.* In Proceedings of the 1st international modal analysis conference, volume 1, pages 167–186. Union College Schenectady, NY, 1982.

[Rosenbaum 2013] Benjamin Rosenbaum et Volker Schulz. *Efficient response surface methods based on generic surrogate models.* SIAM Journal on Scientific Computing, vol. 35, no. 2, pages B529–B550, 2013.

[Saatçi 2010] Yunus Saatçi, Ryan D Turner et Carl E Rasmussen. *Gaussian process change point models.* In Proceedings of the 27th International Conference on Machine Learning (ICML-10), pages 927–934, 2010.

[Sacks 1989] Jerome Sacks, William J Welch, Toby J Mitchell et Henry P Wynn. *Design and analysis of computer experiments.* Statistical science, pages 409–423, 1989.

[Schwab 2016] K. Schwab. The fourth industrial revolution. World Economic Forum, 2016.

[Seeger 2003] Matthias Seeger, Christopher Williams et Neil Lawrence. *Fast forward selection to speed up sparse Gaussian process regression.* In Artificial Intelligence and Statistics 9, numéro EPFL-CONF-161318, 2003.

[Shah 2014] Amar Shah, Andrew Wilson et Zoubin Ghahramani. *Student-t processes as alternatives to Gaussian processes.* In Artificial Intelligence and Statistics, pages 877–885, 2014.

[Shahdin 2010] Amir Shahdin, Joseph Morlier, Hanno Niemann et Yves Gourinat. *Correlating low energy impact damage with changes in modal parameters: diagnosis tools and FE validation.* Structural Health Monitoring, 2010.

[Shih 1988] CY Shih, YG Tsuei, RJ Allemang et DL Brown. *Complex mode indication function and its applications to spatial domain parameter estimation.* Mechanical systems and signal processing, vol. 2, no. 4, pages 367–377, 1988.

[Snelson 2006] Edward Snelson et Zoubin Ghahramani. *Sparse Gaussian Processes using Pseudo-inputs.* In ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS, pages 1257–1264. MIT press, 2006.

[Spitznogle 1970] Frank R Spitznogle et Azizul H Quazi. *Representation and Analysis of Time-Limited Signals Using a Complex Exponential Algorithm.* The Journal of the Acoustical Society of America, vol. 47, no. 5A, pages 1150–1155, 1970.

[Stein 1999] M. L. Stein. Interpolation of Spatial Data: Some Theory for Kriging. Springer, New York, 1999.

[Stein 2012] Michael L Stein. Interpolation of spatial data: some theory for kriging. Springer Science & Business Media, 2012.

[Tan 2003] Bui Thanh Tan. *Proper orthogonal decomposition extensions and their applications in steady aerodynamics.* Master of Engineering in High Performance Computation for Engineered Systems (HPCES), Singapore-MIT Alliance, 2003.

[Titsias 2009] Michalis K. Titsias. *Variational learning of inducing variables in sparse Gaussian processes.* In In Artificial Intelligence and Statistics 12, pages 567–574, 2009.

[Tresp 2000] Volker Tresp. *A Bayesian committee machine.* Neural computation, vol. 12, no. 11, pages 2719–2741, 2000.

[Tripathy 2016] Rohit Tripathy, Ilias Bilionis et Marcial Gonzalez. *Gaussian processes with built-in dimensionality reduction: Applications to high-dimensional uncertainty propagation.* Journal of Computational Physics, vol. 321, pages 191–223, 2016.

[Uhlenbeck 1930] George E Uhlenbeck et Leonard S Ornstein. *On the theory of the Brownian motion.* Physical review, vol. 36, no. 5, page 823, 1930.

[Vassberg 2014] John C Vassberg, Edward N Tinoco, Mori Mani, Ben Rider, Tom Zickuhr, David W Levy, Olaf P Brodersen, Bernhard Eisfeld, Simone Crippa, Richard A Wahls*et al. Summary of the fourth AIAA computational fluid dynamics drag prediction workshop.* Journal of Aircraft, 2014.

[Verveld 2016] Mark Johannes Verveld, Thiemo Kier, Niklas Karcher, Thomas Franz, Mohammad Abu-Zurayk, Matteo Ripepi et Stefan Görtz. *Reduced Order Models for Aerodynamic Applications, Loads and MDO.* 2016.

[Williams 1985] Roger Williams, John Crowley et Havard Vold. *The multivariate mode indicator function in modal analysis.* In International Modal Analysis Conference, pages 66–70, 1985.

[Wilson 2012] Andrew Gordon Wilson. *A Process over all Stationary Covariance Kernels.* Rapport technique, 2012.

[Wilson 2013] Andrew Gordon Wilson et Ryan Prescott Adams. *Gaussian process kernels for pattern discovery and extrapolation.* arXiv preprint arXiv:1302.4245, 2013.

[Wilson 2014a] Andrew Gordon Wilson. *Covariance kernels for fast automatic pattern discovery and extrapolation with Gaussian processes.* PhD thesis, University of Cambridge, 2014.

[Wilson 2014b] Andrew Gordon Wilson. *Covariance kernels for fast automatic pattern discovery and extrapolation with Gaussian processes.* University of Cambridge, 2014.

[Wolpert 1997] David H Wolpert et William G Macready. *No free lunch theorems for optimization.* Evolutionary Computation, IEEE Transactions on, vol. 1, no. 1, pages 67–82, 1997.

[Wright 1934] Orville Wright. *How we made the first flight.* Journal of the Franklin Institute, vol. 217, no. 2, pages 239–254, 1934.

Résumé — Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed non risus. Suspendisse lectus tortor, dignissim sit amet, adipiscing nec, ultricies sed, dolor. Cras elementum ultrices diam. Maecenas ligula massa, varius a, semper congue, euismod non, mi. Proin porttitor, orci nec nonummy molestie, enim est eleifend mi, non fermentum diam nisl sit amet erat. Duis semper. Duis arcu massa, scelerisque vitae, consequat in, pretium a, enim. Pellentesque congue. Ut in risus volutpat libero pharetra tempor. Cras vestibulum bibendum augue. Praesent egestas leo in pede. Praesent blandit odio eu enim. Pellentesque sed dui ut augue blandit sodales. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Aliquam nibh. Mauris ac mauris sed pede pellentesque fermentum. Maecenas adipiscing ante non diam sodales hendrerit. Ut velit mauris, egestas sed, gravida nec, ornare ut, mi. Aenean ut orci vel massa suscipit pulvinar. Nulla sollicitudin. Fusce varius, ligula non tempus aliquam, nunc turpis ullamcorper nibh, in tempus sapien eros vitae ligula. Pellentesque rhoncus nunc et augue. Integer id felis.

Mots clés : Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed non risus. Suspendisse lectus tortor.

Abstract — Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed non risus. Suspendisse lectus tortor, dignissim sit amet, adipiscing nec, ultricies sed, dolor. Cras elementum ultrices diam. Maecenas ligula massa, varius a, semper congue, euismod non, mi. Proin porttitor, orci nec nonummy molestie, enim est eleifend mi, non fermentum diam nisl sit amet erat. Duis semper. Duis arcu massa, scelerisque vitae, consequat in, pretium a, enim. Pellentesque congue. Ut in risus volutpat libero pharetra tempor. Cras vestibulum bibendum augue. Praesent egestas leo in pede. Praesent blandit odio eu enim. Pellentesque sed dui ut augue blandit sodales. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Aliquam nibh. Mauris ac mauris sed pede pellentesque fermentum. Maecenas adipiscing ante non diam sodales hendrerit. Ut velit mauris, egestas sed, gravida nec, ornare ut, mi. Aenean ut orci vel massa suscipit pulvinar. Nulla sollicitudin. Fusce varius, ligula non tempus aliquam, nunc turpis ullamcorper nibh, in tempus sapien eros vitae ligula. Pellentesque rhoncus nunc et augue. Integer id felis.

Keywords: Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed non risus. Suspendisse lectus tortor.

Nom du laboratoire, adresse du laboratoire
Ville du laboratoire