Part I

Introduction

Chapter 1

Context

Résumé

En raison du coût important des expériences réalisées sur les systèmes physiques, concevoir ces derniers purement par l'expérimentation devient un exercice coûteux. Par conséquent les modèles numériques deviennent des moyens importants dans le processus de conception des systèmes physiques. Traditionnellement, ces modèles ont été construits en effectuant des expériences dans un environnement contrôlé, itérativement. Une méthode plus rentable pour construire ces modèles est d'utiliser des algorithmes d'apprentissage automatique, qui déduisent ces modèles de données (expérimentales ou simulées) et peuvent être utilisés pour interpoler et extrapoler. Dans cette thèse, nous développons des modèles d'apprentissage automatique en combinant la connaissance acquise sur le design (conception) d'avion avec les données expérimentales.

Les modèles d'apprentissage automatique constituent un compromis entre le biais et la variance, formalisé par Wolpert dans son célèbre théorème "No free lunch" [Wolpert 1997]. Les fonctions constitutives dans un espace d'hypothèse représentent le biais (préjugé) ou les hypothèses de l'algorithme apprenant (eg. des fonctions linéaires pour la régression linéaire). En l'absence de biais suffisants, la famille de fonctions dans l'espace de recherche devient très grande, et mène à une variance élevée dans le modèle. Au contraire, un biais trop important signifie que la vraie fonction de transformation n'existe pas dans l'espace d'hypothèse. Dans ce cas, l'algorithme d'apprentissage trouve la fonction la plus proche de f dans l'espace d'hypothèse, ce qui mène à un sous-ajustement.

Une méthode pour surmonter ce compromis est l'utilisation d'un grand nombre de données. Étant donné la facilité d'accès à un nombre de plus en plus grand de données, nous pouvons progressivement réduire le biais dans l'apprentissage de modèles. C'est

aussi le concept principal de 'Apprentissage profond', où plusieurs couches de réseaux de neurones définissent un vaste espace d'hypothèse [Goodfellow 2016, LeCun 2015].

Nous proposons de construire de meilleurs modèles d'apprentissage de Processus Gaussiens (GPs) en intégrant les connaissances préalablement testées sur les systèmes des avions avec les données expérimentales. Un modèle généré par la fusion des deux méthodologies sera à la fois cohérent avec la physique du système et sera plus rapide à évaluer. Nous intégrons trois types de connaissances antérieures en répondant aux questions suivantes :

1. **Motifs**: Comment ajouter des informations *a-priori* d'un motifs dans un algorithme d'apprentissage ? (part II).
2. **Données de simulation**: Comment fusionner l'information *a-priori* des simulations avec des expériences ? (chapter 6).
3. **Relations**: Comment ajouter des informations *a-priori* des relations entre les mesures ? (chapter 7).
4. **Le passage à l'échelle**: Comment appliquer la régression GP à de grands volumes de données ? (chapter 3 et section 7.4).

1.1 Introduction

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, 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, while 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.

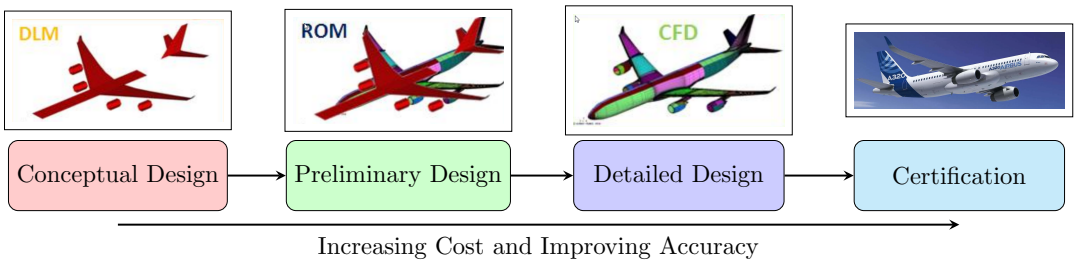


Figure 1.1: Phases of an aircraft design cycle

1.2 Aircraft design cycle

Design phases

To simplify this process an aircraft design is broken down into several design phases (figure 1.1). Each phase requires an ever increasing amount of predictions and fidelity of the target objectives (aircraft weight or lift to drag ratio). Preliminary design phase requires a few low-fidelity design trade-offs between major disciplines, whereas detailed design phase, requires intensive intra-disciplinary and inter-disciplinary optimization. Finally, during the flight-test and certification phase, capability to predict target objectives in real-time can provide significant gains. These analyses cover large parts of the flight envelope and require high-fidelity predictions. Hence the capability to accurately and quickly predict the target objectives 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 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

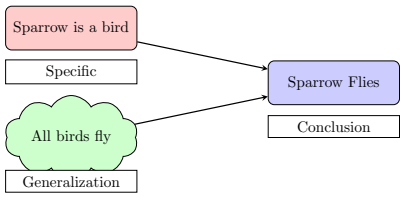
phase, during the detailed design phase, physical complexity is needed to find a robust optimum design point [Raymer 2012].

Need for accuracy

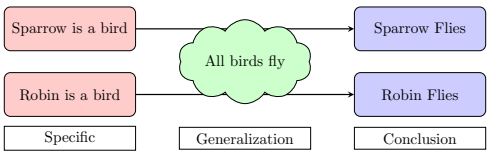
Instead of approximating physical complexity, surrogate models[1] simplify mathematical complexity [Verveld 2016]. Surrogate models learn patterns between the input and output data-set and then are used to make predictions on the desired point. This property is very useful in quickly exploring the design 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 [Gazaix 2017, Lefebvre 2017].

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 1.2). 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

Model building in aircraft engineering is traditionally outsourced to research institutes. Researchers perform iterative experiments in a controlled environment and discover pat-

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

terns between the physics of the system. This is a rigorous and time-consuming method of developing models. 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]. Machine learning is a much more cost effective 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 [McCulloch 1943, Rosenblatt 1958], 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.

Developing faster models

1.3 Machine Learning

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

Components of learning

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.

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.

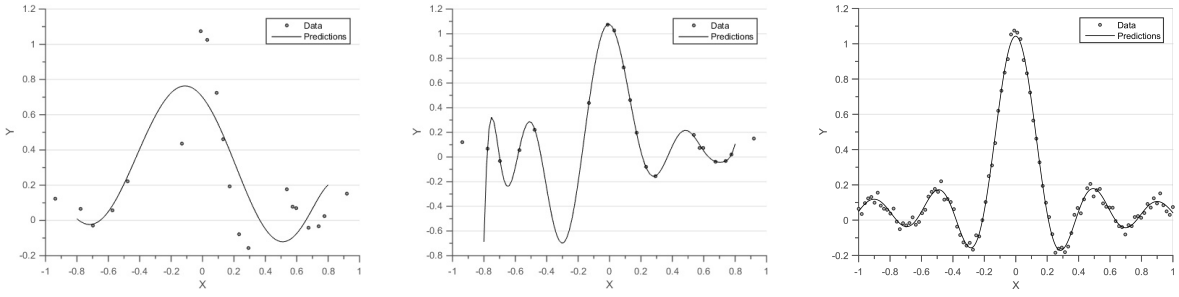
Bias vs Variance

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

[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

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, stringent assumptions mean 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)).



(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

Soft and hard constraints

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 functions ($y = f(x)$) 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 biases used while model building 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, Forrester 2008] 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 these physical systems. We propose to build better machine learning models by integrating the time-tested prior knowledge of physical systems with experimental data.

To integrate the prior information we propose to use the Bayesian inference for model

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

building. Bayesian inference is a method of statistical inference in which the Bayes theorem is used to update an initial probability (prior) using evidence (data-set) to give the final probability (posterior). More specifically, we will use the Gaussian Processes (GP) Regression or Kriging framework which is a subset of the Bayesian Inference algorithms to define prior information of physical systems. This choice is driven by the fact that GPs have been shown to perform better when data is scarce or costly [Stein 1999], and GPs provide a functional method to define the hypothesis space. But, before deep diving into the details of GP let us have a look at a simple Bayesian linear regression algorithm.

1.4 Bayesian linear regression

Suppose we have access to a training set of observations (or outputs) $\boldsymbol{y} = (y(\boldsymbol{x_1}), \ldots, y(\boldsymbol{x_i}), \ldots, y(\boldsymbol{x_N}))^T$, evaluated at a set of known inputs $\boldsymbol{X} = (\boldsymbol{x_1}, \ldots, \boldsymbol{x_i}, \ldots, \boldsymbol{x_N})^T$, and we wish to predict $\boldsymbol{y(x_*)}$ at a test input $\boldsymbol{x_*}$. The input and output can be multi-dimensional; $\boldsymbol{x_i} \in \mathbb{R}^{D_{inputs}}$ and $\boldsymbol{y(x_i)} \in \mathbb{R}^{D_{outputs}}$. The process of learning the transformation function 'f' to make a prediction at a new point is called regression. In the following section we follow the formulation for Bayesian linear regression provided by [MacKay 2003].

Basis functions

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

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

Likelihood

Here, $\boldsymbol{w}$ are the parameters of the function, such that $w_0 \in \mathbb{R}$ denotes the intercept and $\boldsymbol{w_1} \in \mathbb{R}^{D_{inputs}}$ 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[6] with variance σ_n^2. The above equations 1.1 can be combined to result

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

in the likelihood $\Pr[\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{w}]$

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

The notation $\Pr[\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{w}]$ symbolizes the probability distribution of the observations $\boldsymbol{y}$ at the inputs $\boldsymbol{X}$ given the parameter $\boldsymbol{w}$. The notation $\mathcal{N}[\boldsymbol{\mu}, \boldsymbol{\Sigma}]$ symbolizes a multi-variate Gaussian distribution for mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$. $\boldsymbol{I_N}$ is an identity matrix of size $N \times N$.

Prior

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.

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

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

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

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

Marginal Likelihood

The *marginal likelihood* $(\Pr[\boldsymbol{y} \mid \boldsymbol{X}])$ is a normalization constant, for more details please refer to section 2.4. After, using the equation 1.2, 1.3 and 1.4 we can get the posterior distribution of weights as:

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

Here, $\boldsymbol{A} = \sigma_n^{-2}\boldsymbol{X}\boldsymbol{X}^{\boldsymbol{T}} + \boldsymbol{\Sigma}_{\boldsymbol{Prior}}^{-\boldsymbol{2}}$. Thus the posterior distribution for function f at test point $\boldsymbol{x_*}$ becomes:

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

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

Posterior

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

Experiment

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

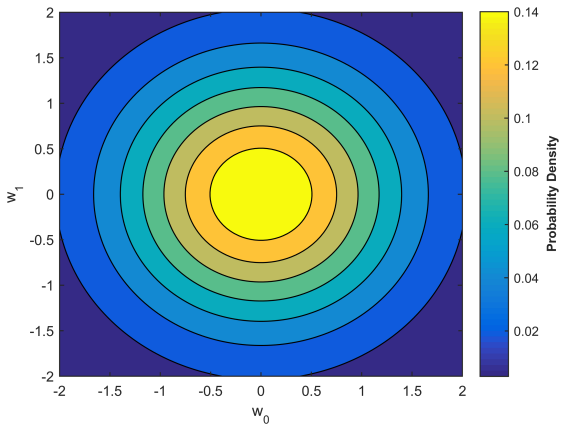
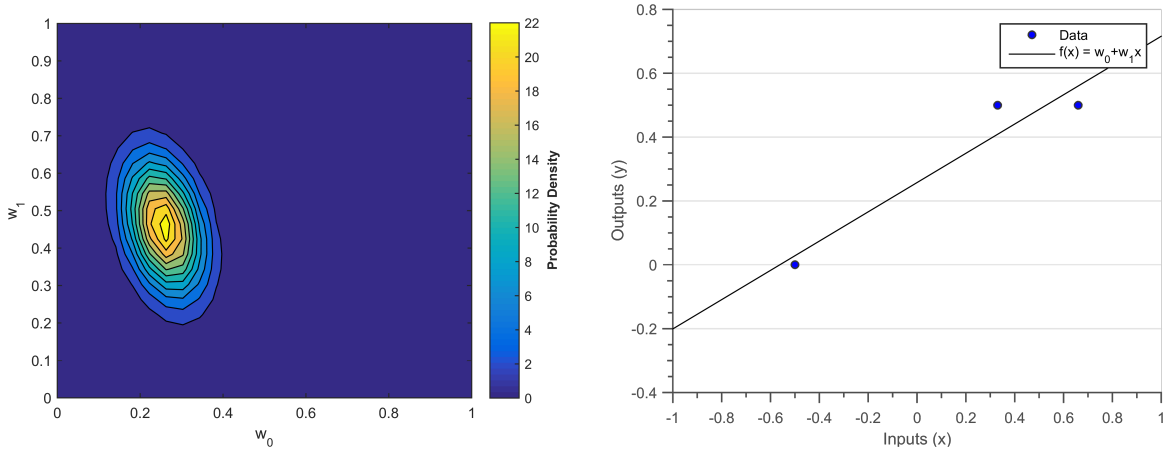


Figure 1.4: Prior: Contours of probability density for a prior on parameters w as defined by equation 1.7

Using equations 1.7 and 1.6 we can calculate the posterior distribution for the parameters. For a noise model of $\Pr[\epsilon] = \mathcal{N}(0, \sigma_n^2 = 0.1^2)$ the posterior distribution comes out as the figure 1.5(a)

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

This simple example demonstrates how to estimate the parameters w_0 and w_1 using the prior distribution on parameters w and data-set $\mathcal{D}_1$. 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 1.5(b)).



(a) 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

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

Figure 1.5: Posterior and Prediction in Bayesian linear regression

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 GPs to express prior assumptions. GPs 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, GP Regression is a type of non-parametric 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

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]. It has since been a popular method in engineering design to create surrogate models of costly computer simulations. Interest in GPs grew in the machine learning community from neural networks research. It was shown that a Bayesian neural network becomes a GP as the number of neurons tends to infinity [Neal 2012]. GPs are probabilistic distributions over functions, which provide a Bayesian non-parametric approach to smoothing and interpolation. Regression performed through GPs are gener-

alizations of the Kriging algorithm.

Informally, a GP is a method to probabilistically define a family of functions. A GP can be completely parametrized by its mean and covariance function. The mean function defines the trend-line of the family of functions, while the covariance defines the spatial dependency of input-points. Throughout this thesis we will manipulate the covariance function to enforce prior information into a GP Regression framework.

1.5 Layout

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 algorithm.

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 work-flow of how to perform regression using GPs. The chapter describes the key constituents of a GP, how to draw random functions from them, describes how to perform prediction in presence and absence of measurement noise, and finally introduces marginal likelihood maximization 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 increases to more than $\mathcal{O}(10^4)$. There exist two main methods to scale up a GP regression, one using a 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 data-set, giving directions to choosing optimal parameters and extracting the best possible result.

Part II

A GP prior can be fully parameterized by its mean and covariance functions. From part II on-wards we only concentrate on manipulating the covariance function to embed the desired *a-priori* information into a GP prior. Part II demonstrates how to incorporate an *a-priori* pattern into a regression framework. It consists of two chapters, chapter 4 describes

basic covariance functions, while chapter 5 explains how to make new covariance functions by combining the basic covariance functions. This part has been inspired from the works of [Bishop 2006, MacKay 2003, Duvenaud 2014, Wilson 2014, Lloyd 2014, Durrande 2001, Durrande 2013a].

Chapter 4

Chapter 4 describes different types of covariance functions and their corresponding families of functions. Patterns such as smoothness, linearity, differentiability, etc. can be easily encoded using simple covariance functions. We list the properties of basic stationary and non-stationary covariance functions, while also giving an example to demonstrate the effects of choosing different covariance functions on the final regression model. Stationary kernels have an interesting property that their Fourier transforms exist and are more interpretable. We exploit this property to learn modal parameters of a structure from their dynamic excitations.

Chapter 5

Chapter 5 provides intuition on what happens when we combine different covariance functions. We first look at the effects of simply adding or multiplying covariances in one-dimensional inputs (section 5.2), and then investigate how to create covariance functions for multi-dimensional inputs (section 5.3). We use these kind of covariance functions to detect onset of plasticity in Tensile experiments on an aluminum alloy (section 5.2.4) and interpolate pressures in the Transonic regime (section 5.3.5).

Part III

Several real-world problems often exhibit strong correlations between output variables, for example, correlations across spatial coordinates (x, y, z) in an experiment. Part III of this thesis demonstrates how to incorporate an *a-priori* information among multiple outputs, this is also called as 'Multi-Task Gaussian Process' (MTGP) in GP literature. This part is split into two chapters, chapter 6 describes the MTGP for the case of multi-fidelity simulations, while chapter 7 describes the MTGP for the case when we have *a-priori* information of relationships between outputs. This part has been inspired from the works of [Forrester 2007, Alvarez 2011, Bonilla 2008, Boyle 2005, Kennedy 2000, Le Gratiet 2013, Constantinescu 2013, Alvarez 2009, Jidling 2017, Ginsbourger 2013, Särkkä 2011]

Chapter 6

Chapter 6 describes how to build GP regression models in presence of multi-fidelity simulations. The *a-priori* information in this case is that one simulation is more accurate than another. The chapter starts by presenting MTGP models when no information of dependency between outputs exists, we then present the multi-fidelity model. Finally, we expand the multi-fidelity model to the case of extrapolating experimental data using a simulation model.

Chapter 7

Finally, chapter 7 demonstrates how to build an MTGP model if we know a relationship in the form of an operation between two outputs. We first tackle the case of linear operators and then expand the case to non-linear operators. We then give methods to scale up MTGP models to large numbers of data-points, mainly by applying techniques already seen in chapter 3.

1.6 Contributions

Before highlighting the contributions of this thesis it is important to understand the research work in context of a CIFRE thesis. A CIFRE thesis is different from a traditional PhD thesis since the doctoral candidate is an employee of the host company, while also being linked to the research university. Thus the research performed in this thesis is application oriented, mostly trying to solve some industry specific problem.

Developments

The main theme of this thesis is applying the developments from the supervised machine learning community to solving problems in engineering design. We choose GP regression as our supervised Machine Learning algorithm, this choice is driven because GP regression performs best when data is costly or sparse and the covariance functions acts as a good mechanism for encoding functional relationships. The developments proposed in this thesis can be further divided into two subgroups. First, where we have made contributions to the GP regression community. Second, where we have applied a formalism from GP regression literature to problems in aircraft design. Table 1.6 highlights the main contributions of this thesis.

GP + Aircraft Design	
Chapter 4	Identifying Structural dynamic parameters [Chiplunkar 2017b]
Chapter 5	Identifying onset of non-linearity [Chiplunkar 2016b]
Chapter 5	Interpolating Shock position [Chiplunkar 2017a]
Chapter 6	Extrapolating using a simulation model
Chapter 7	Adding flight mechanics to GP [Chiplunkar 2016a]

Additions to GP Regression	
Chapter 7	Scaling multi-task GP [Chiplunkar 2016c, Chiplunkar 2017c]

Table 1.1: List of contributions

When we are applying GP regression to Aircraft design problems, we are effectively combining the prior knowledge coming from decades of experimentation and adding 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. A major roadblock in using GP regression is that building GPs for large number of data-points is a costly task, we will also propose solutions for scaling GPs in this thesis. We answer the following questions throughout our thesis:

1. **Pattern**: How to add *a-priori* information of a pattern in a learning algorithm? For example, given that shock is a discontinuous change in pressure, how to predict the position of shock on an airfoil (part II).

2. **Simulation model**: How to merge *a-priori* information of simulations with experiments? For example, given a simulation model and experimental data how to perform extrapolations on experimental data (chapter 6)?

3. **Relationships**: How to add *a-priori* 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 7)?

4. **Scaling**: How to scale GP regression to large data-sets? How to scale single output and multiple output GPs to large number of outputs (chapter 3 and section 7.4).

The Matlab codes available in this manuscript are only for understanding. The basic toolbox used for this thesis is GPML provided with [Rasmussen 2005], all new functionalities have been grafted into this basic toolbox. The code of this manuscript is available at `https://github.com/ankitchiplunkar/thesis_isae`.

Chapter 2

Gaussian Process Regression

Résumé

Dans ce chapitre nous fournissons une introduction à la Régression avec GPs. Les GPs sont les candidats idéaux à la régression grâce à de leur propriété de marginalisation qui les rend résolubles informatiquement. Ce qui rends les réalisations des fonctions aléatoires, le calcul de distribution *a posteriori* et la sélection automatique d'hyper-paramètres faisables informatiquement aussi. Ceci fait des GPs des candidats idéaux pour définir une distribution *a-priori* dans un cadre de Régression Bayésien.

La section 2.2 détaille les composants clés du GPs. Un GP peut être complètement paramétré par sa moyenne et sa fonction de covariance. La tendance d'un GP est définie par sa fonction moyenne, tandis que la structure de ses fonctions constitutives est définie par la fonction de covariance. La moyenne d'un GP peut être supposée nulle, car un terme supplémentaire dans la fonction de covariance peut représenter la fonction moyenne. Par conséquent, le problème de l'apprentissage dans un GP consiste à trouver les propriétés appropriées de la fonction de covariance (section 2.2.3). Une fois qu'une forme de fonction de covariance est choisie, nous pouvons calculer la matrice de Gram aux points désirés et l'utiliser pour tirer des fonctions aléatoires (section 2.2.4).

La section 2.3 décrit comment calculer la distribution *a posteriori*. Celle-ci est la distribution conditionnelle ($\Pr[f(\boldsymbol{x_*}) \mid \boldsymbol{y}, \boldsymbol{X}, \boldsymbol{\theta}]$) pour une distribution *a-priori* supposée ($\Pr[f] = GP(0, k(\boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{\theta})$) et les ensembles des points de données observés ($\mathcal{D} = \boldsymbol{X}, \boldsymbol{y}$). En raison de l'hypothèse Gaussienne, les probabilités conditionnelles sont résolubles informatiquement et peuvent être calculées en utilisant quelques opérations matricielles. Le calcul de la distribution *a posteriori* est facile à la fois en l'absence (section 2.3.1) ou en l'presence (section 2.3.2) de bruit dans les observations.

La section 2.4 montre l'importance de choisir les hyper-paramètres correctement. Une pratique courante dans la communauté est de maximiser la probabilité marginale pour choisir automatiquement les hyper-paramètres. La probabilité marginale est la probabilité d'une distribution *a-priori* $\Pr[f] = GP(0, k(\boldsymbol{x_1}, \boldsymbol{x_2}, \theta)$ générant les observations $\mathcal{D}$. La maximisation de la probabilité marginale donne les hyper-paramètres optimaux.

2.1 Introduction

Data-set

Suppose we perform a simulation or experiment on an input point $\boldsymbol{x_j} \in \mathbb{R}^{D_{inputs}}$ 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, $\{\mathcal{D} = (\boldsymbol{x_j}, y_j) | j \in [1; N]\}$. The full input and output vectors can be denoted as $\boldsymbol{X} = (\boldsymbol{x_1}, \ldots, \boldsymbol{x_j}, \ldots, \boldsymbol{x_N})^T$ and $\boldsymbol{y} = \{\boldsymbol{y_1}, \ldots \boldsymbol{y_j}, \ldots, \boldsymbol{y_N}\}^T$ such that $\boldsymbol{X} \in \mathbb{R}^{N \times D_{inputs}}$ and $\boldsymbol{y} \in \mathbb{R}^N$. Given this data, we are interested in making predictions on a target point[1] $\boldsymbol{x_*}$ that may not be present in our series of experiments. This means that we need to use our training data and learn the true physical process $f(\boldsymbol{x})$ 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 the family of functions. More formally, a GP is a distribution over functions such that any finite set of function values $[f(\boldsymbol{x_1}), \ldots, f(\boldsymbol{x_j}), \ldots, f(\boldsymbol{x_N})]$ have a joint Gaussian distribution [Rasmussen 2005].

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 $\boldsymbol{x} \sim \mathcal{N}(\binom{0}{0}, \left(\begin{smallmatrix}1 & 0\\ 0 & 1\end{smallmatrix}\right))$ defines a Gaussian vector with mean $\binom{0}{0}$ and covariance $\left(\begin{smallmatrix}1 & 0\\ 0 & 1\end{smallmatrix}\right)$). 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 $f(\boldsymbol{x})$ at a particular point $\boldsymbol{x}$.

Mean

A GP model before conditioning on data can be completely parameterized by its mean $(m(\boldsymbol{x}))$ and its covariance function $(k(\boldsymbol{x_1}, \boldsymbol{x_2}))$ also called a kernel.

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

Covariance

In the context of GPs a kernel is a measure of similarity between pairs of functional values $(f(\boldsymbol{x}))$ evaluated at input points, often involving an inner product of basis functions $\phi(\boldsymbol{x})$ [Bishop 2006]. Please refer to Part II for a more detailed insight into kernels[2].

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

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

$$Cov(f(\boldsymbol{x_1}), f(\boldsymbol{x_2})) = \mathbf{E}[f(\boldsymbol{x_1}) - m(\boldsymbol{x_1}), f(\boldsymbol{x_2}) - m(\boldsymbol{x_2})] \tag{2.2}$$
$$= k(\boldsymbol{x_1}, \boldsymbol{x_2}) \tag{2.3}$$

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

$$\Pr[f(\boldsymbol{x})] = GP(m(\boldsymbol{x}), k(\boldsymbol{x_1}, \boldsymbol{x_2})) \tag{2.4}$$

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

Tractable

Performing inference on an infinite dimensional vector (function) can be a computationally intensive task. Thankfully, due to the marginalization property of Gaussians (appendix A.4), 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 $[f(\boldsymbol{x_1}), \ldots, f(\boldsymbol{x_j}), \ldots, f(\boldsymbol{x_N})]$ have a joint Gaussian distribution in GP (this is also the formal definition of GP). This property means that GP specified in equation 2.4 also specifies equation 2.5. This makes GPs computationally tractable, which is one of the major benefits of GP.

$$\Pr\begin{bmatrix} f(\boldsymbol{x_1}) \\ f(\boldsymbol{x_2}) \end{bmatrix} = \mathcal{N}\left(\begin{bmatrix} m(\boldsymbol{x_1}) \\ m(\boldsymbol{x_2}) \end{bmatrix}, \begin{bmatrix} k(\boldsymbol{x_1}, \boldsymbol{x_1}) & k(\boldsymbol{x_1}, \boldsymbol{x_2}) \\ k(\boldsymbol{x_2}, \boldsymbol{x_1}) & k(\boldsymbol{x_2}, \boldsymbol{x_2}) \end{bmatrix}\right) \tag{2.5}$$

GP regression work-flow

While performing regression in a GP framework, we first define a family of functions also called prior (section 2.2). The next step involves looking at the data-set and eliminating all the functions in our prior hypothesis space which do not pass through the data-set, this step gives us the posterior mean and variance (section 2.3). Finally, we can further improve our predictions by fine-tuning our hyper-parameters (section 2.4).

2.2 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.

2.2.1 Hyper-parameters

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 [Rasmussen 2005]. We will look at how to automatically fine-tune hyper-parameters in section 2.4, whereas part II will describe how to choose the functional form of covariance functions.

2.2.2 Mean function

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

Zero mean

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 (more details in section 5.2.2).

We assume a zero mean prior throughout this thesis. 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 2005] (equation 2.6).

$$\Pr[f(\boldsymbol{x})] = GP(0, k(\boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{\theta})) \tag{2.6}$$

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

2.2.3 Covariance function

The covariance function is a positive definite kernel, such that for any $a_i \in \mathbb{R}$ equation 2.7 is valid [Stein 1999].

$$\sum_{i=1}^{N}\sum_{j=1}^{N} a_i a_j k(\boldsymbol{x_1}, \boldsymbol{x_2}) \geq 0 \tag{2.7}$$

A popular choice of covariance function is a Standard Exponential (SE) function (equation 2.8), because it defines a family of highly smooth (infinitely differentiable) non-linear

functions as shown in figure 2.2.

$$k_{SE}(\boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{\theta}) = \theta^2_{amplitude} exp\left[-\frac{\boldsymbol{d^2}}{2\theta^2_{lengthScale}}\right] \tag{2.8}$$

SE kernel

For the case of the SE kernel the hyper-parameters ($\boldsymbol{\theta} = [\theta_{amplitude}, \theta_{lengthScale}]$) are amplitude ($\theta_{amplitude}$), which defines average distance from mean, and the length scale ($\theta_{lengthScale}$), which defines the smoothness of functions. Here, $\boldsymbol{d}$ defines the absolute distance between points ($\boldsymbol{d} = |\boldsymbol{x_1} - \boldsymbol{x_2}|$). Covariance functions which are purely a function of distance $\boldsymbol{d}$ are called isotropic stationary functions. These covariance functions remain unchanged if the points $\boldsymbol{x_1}, \boldsymbol{x_2}$ 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 $\boldsymbol{x_1}$ tends to $\boldsymbol{x_2}$, then $k(\boldsymbol{x_1}, \boldsymbol{x_2})$ approaches $\theta^2_{amplitude}$ this means that $f(\boldsymbol{x_1})$ is highly correlated with $f(\boldsymbol{x_2})$. This is a good characteristic for smooth functions since points in the neighbourhood must be alike. If $\boldsymbol{x_1}$ is far away from $\boldsymbol{x_2}$, then $k(\boldsymbol{x_1}, \boldsymbol{x_2})$ 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
SEKernel = @(theta, x1, x2)(theta(1).^2*exp(-(x1 - x2).^2/(2*
   theta(2).^2)));
% theta(1): is the amplitude hyper-parameter
% theta(2): is the length-scale hyper-parameter
```

Matlab Code 2.2: A SE covariance function

2.2.4 Sampling functions from GP priors

Gram matrix

To have a look at the constituent functions in a prior we can randomly sample functions from the GP. To draw random functions from a GP we choose $N*$ input points $\boldsymbol{X_*} = \{\boldsymbol{x_{1*}}, \boldsymbol{x_{2*}}, \ldots, \boldsymbol{x_{N*}}\}^T$ and write corresponding mean vector $\boldsymbol{m(X_*)}$ and covariance matrix $\boldsymbol{K(X_*, X_*)}$ using equation 2.5 and 2.8, the covariance matrix (equation 2.9) is also called the Gram matrix. We then generate a random Gaussian vector $\boldsymbol{f(X_*)}$ for this multi-variate

Gaussian (equation 2.6) and plot the generated values as a function of inputs $\boldsymbol{X_*}$.

$$\boldsymbol{K(X_*, X_*)} = \begin{bmatrix} k(\boldsymbol{x_{1*}}, \boldsymbol{x_{1*}}) & k(\boldsymbol{x_{1*}}, \boldsymbol{x_{2*}}) & \dots & k(\boldsymbol{x_{1*}}, \boldsymbol{x_{N*}}) \\ k(\boldsymbol{x_{2*}}, \boldsymbol{x_{1*}}) & k(\boldsymbol{x_{2*}}, \boldsymbol{x_{2*}}) & \dots & k(\boldsymbol{x_{2*}}, \boldsymbol{x_{N*}}) \\ \vdots & \vdots & \ddots & \vdots \\ k(\boldsymbol{x_{N*}}, \boldsymbol{x_{1*}}) & k(\boldsymbol{x_{N*}}, \boldsymbol{x_{2*}}) & \dots & k(\boldsymbol{x_{N*}}, \boldsymbol{x_{N*}}) \end{bmatrix} \tag{2.9}$$

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 2.8).

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

    [X1, X2] = meshgrid(x1, x2);
    % evaluating element wise gram matrix
    gramMatrix = covarianceFunction(theta, X1, X2);

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(SEKernel, theta, xStar, xStar);
imagesc(gramMatrix); % Plotting the Gram Matrix
```

Matlab Code 2.3: Plotting the Gram Matrix

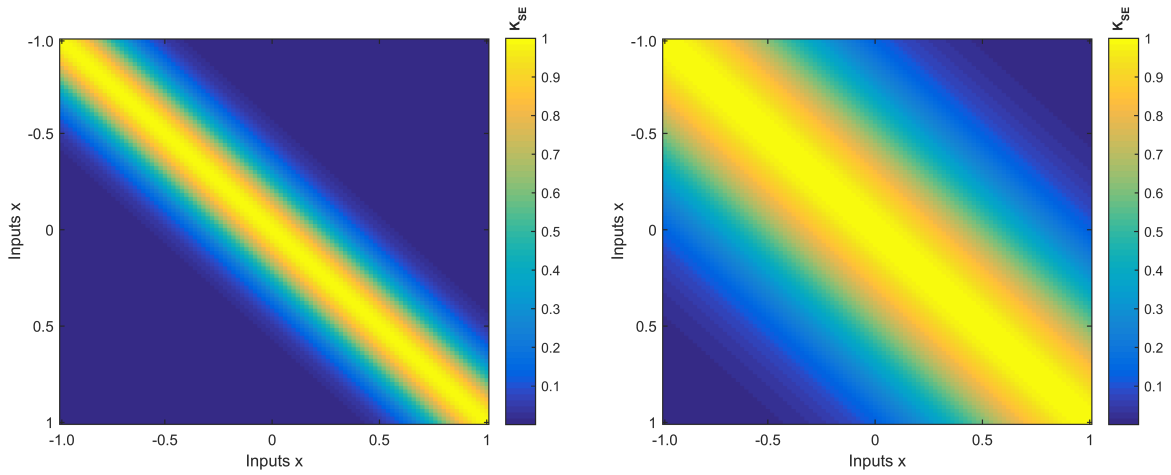
Figure 2.1

Figure 2.1 shows the covariance matrix for SE kernel with different hyper-parameters at the input points $\boldsymbol{X^*} = \{[0 : 0.02 : 1]\}$. 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).

Random Draw

To generate a random Gaussian vector $\boldsymbol{f(X_*)}$ of length N_*, we first calculate the Cholesky decomposition[3] of the covariance matrix $\boldsymbol{K(X_*, X_*)} = \boldsymbol{K_{Lower} K_{Lower}^T}$, where

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



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

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

Figure 2.1: Covariance matrix for a SE kernel with different hyper-parameters at the input points $\boldsymbol{X}^* = \{[0 : 0.02 : 1]\}$. 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).

$\boldsymbol{K_{Lower}}$ is a lower triangular matrix. We then generate a random vector $\boldsymbol{u}$, such that $\boldsymbol{u} = \mathcal{N}(0, \boldsymbol{I_{N_*}})$ and $\boldsymbol{I_{N_*}}$ is an identity matrix of size $N_* \times N_*$. The random vector can then be computed as $\boldsymbol{f}(\boldsymbol{X_*}) = \boldsymbol{m}(\boldsymbol{X_*}) + \boldsymbol{K_{Lower}}\boldsymbol{u}$, and when plotted with the inputs $\boldsymbol{X_*}$, gives a randomly drawn function. The Matlab Code 2.4 shows how we can draw functions randomly from a GP prior.

```
function [randomFunction] = drawRandomFunctions(meanVector,
   gramMatrix)

N = length(meanVector);

% 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);

randomFunction = meanVector + L'*rand(N, 1);

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

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

% Plotting the variance of Prior
gramMatrix = evaluateGramMatrix(SEKernel, theta, xStar, xStar);
f = [meanVector+2*sqrt(diag(gramMatrix)); flipdim(meanVector-2*
   sqrt(diag(gramMatrix)),1)];
hold on; fill([xStar; flipdim(xStar,1)], f, [7 7 7]/8);

% Plotting the randomly drawn sample function
[randomFunction] = drawRandomFunctions(meanVector, gramMatrix);
hold on; plot(xStar, randomFunction, 'b');
```

Matlab Code 2.4: Sampling a random function from the prior

Figure 2.2

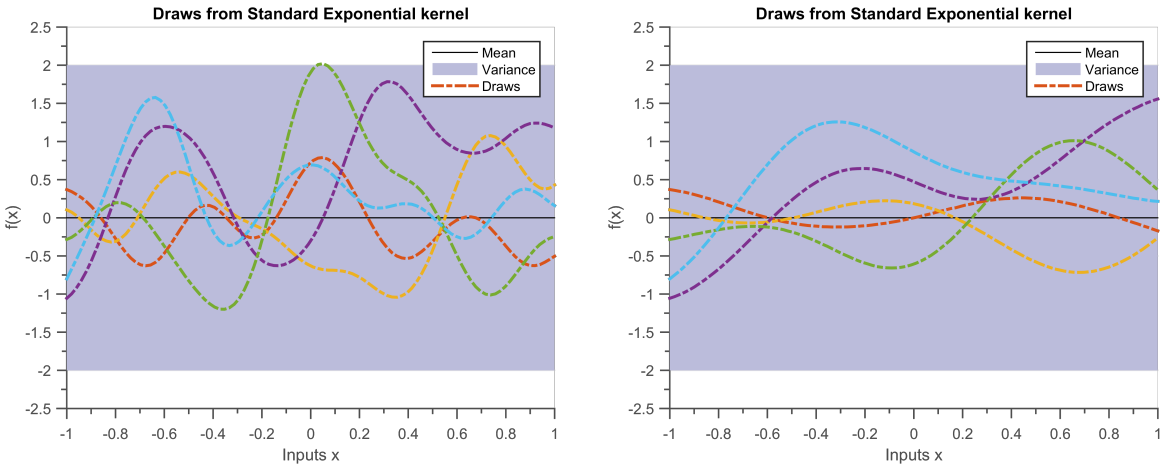
Figure 2.2 shows 5 random functions drawn for a zero mean GP with the covariance matrices of figure 2.1. The solid black line defines the mean function, shaded blue region defines 95% confidence interval (2σ) distance away from the mean. The dashed lines represent five functions drawn at random from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

2.3 Posterior

Once we have defined an appropriate prior we wish to incorporate the information of the 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 distribution.

2.3.1 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 while creating surrogate models of computer simulations since their outputs can



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

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

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.

be treated as having no noise [Sacks 1989]. If we desire to interpolate at test points $\boldsymbol{X}_*$, then the joint distribution of the training outputs $\boldsymbol{f}(\boldsymbol{X})$ and test outputs $\boldsymbol{f}(\boldsymbol{X}_*)$ is given by equation 2.10.

$$\Pr\begin{bmatrix}\boldsymbol{f}(\boldsymbol{X})\\ \boldsymbol{f}(\boldsymbol{X}_*)\end{bmatrix} = \mathcal{N}\left(\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}) & \boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}_*)\\ \boldsymbol{K}(\boldsymbol{X}_*,\boldsymbol{X}) & \boldsymbol{K}(\boldsymbol{X}_*,\boldsymbol{X}_*)\end{bmatrix}\right) \tag{2.10}$$

$\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}_*)$ is $N \times N_*$ covariance matrix between the training points $\boldsymbol{X}$ and test points $\boldsymbol{X}_*$ (equation 2.9). The other covariance matrices $\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X})$, $\boldsymbol{K}(\boldsymbol{X}_*,\boldsymbol{X})$ and $\boldsymbol{K}(\boldsymbol{X}_*,\boldsymbol{X}_*)$ can be computed similarly.

Conditioned distribution

The posterior will be the conditional probability of $\boldsymbol{f}(\boldsymbol{X}_*)$ given the prior and data-set. For a multi-variate Gaussian prior the conditional posterior distribution is also a multi-variate Gaussian and can be calculated tractably. For a more detailed derivation refer to appendix A.5. 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:

$$\Pr(\boldsymbol{f}(\boldsymbol{X}_*) \mid \boldsymbol{X}_*, \boldsymbol{X}, \boldsymbol{f}(\boldsymbol{X})) = GP(\boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X})(\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}))^{-1}\boldsymbol{f}(\boldsymbol{X}), \\ \boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X}_*) - \boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X})(\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}))^{-1}\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}_*)) \tag{2.11}$$

The term $\boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X})(\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}))^{-1}\boldsymbol{f}(\boldsymbol{X})$ is the predicted mean of the posterior at the test points $\boldsymbol{X}_*$. The term $\boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X}_*) - \boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X})(\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}))^{-1}\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}_*)$ is the predicted covariance.

We again use the data-set $\mathcal{D}_1$ (earlier used in section 1.4) 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 require only a few matrix operations.

.

```
function [meanPosterior, gramMatrixPosterior] =
    evaluatePosterior(meanFunction, covarianceFunction, theta, x,
     xStar)

% Evaluating the mean vector
meanVector = meanFunction(xStar);

% Evaluating the neccesary matrices
gramMatrixXstarX = evaluateGramMatrix(covarianceFunction, theta,
     xStar, x);
gramMatrixXX = evaluateGramMatrix(covarianceFunction, theta, x,
    x);
gramMatrixXstarXstar = evaluateGramMatrix(covarianceFunction,
    theta, xStar, xStar);

+% Calculating the posterior mean and covariance
meanPosterior = meanVector + gramMatrixXstarX*(gramMatrixXX\y);
gramMatrixPosterior = gramMatrixXstarXstar - gramMatrixXstarX*(
    gramMatrixXX\gramMatrixXstarX');

end

%% Dataset D_1
x = [-0.5, 1/3, 2/3]';
y = [0, 0.5, 0.5]';
hold on; plot(x, y, '*');
```

```
[meanPosterior, gramMatrixPosterior] = evaluatePosterior(
   meanFunction, SEKernel, theta, x, y, xStar);
% Plotting the variance and mean of Posterior
hold on; plot(xStar, meanPosterior, 'k');
f = [meanPosterior+2*sqrt(diag(gramMatrixPosterior)); flipdim(
   meanPosterior-2*sqrt(diag(gramMatrixPosterior)),1)];
hold on; fill([xStar; flipdim(xStar,1)], f, [7 7 7]/8)

% Plotting a randomly drawn sample
[randomFunction] = drawRandomFunctions(meanPosterior,
   gramMatrixPosterior);
hold on; plot(xStar, randomFunction, 'b');
```

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 thanks to the Bayes theorem all the functions that do not pass through the data, are eliminated from the hypothesis space. The solid black line defines the mean function, blue region defines 95% confidence interval (2σ) distance away from the mean. The mean 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.

Figure 2.3

2.3.2 Posterior with Noisy observations

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

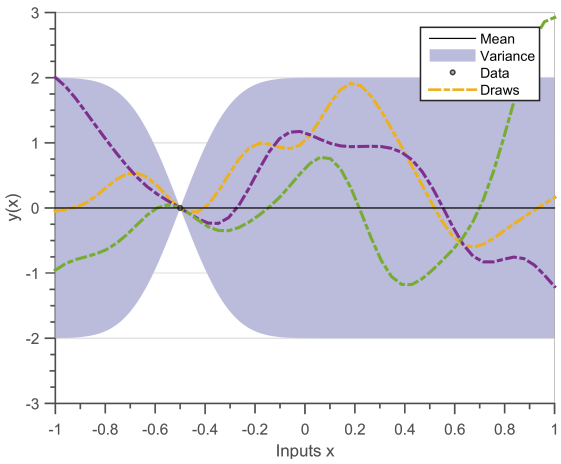
$$y(\boldsymbol{x}) = f(\boldsymbol{x}) + \epsilon \tag{2.12}$$

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:

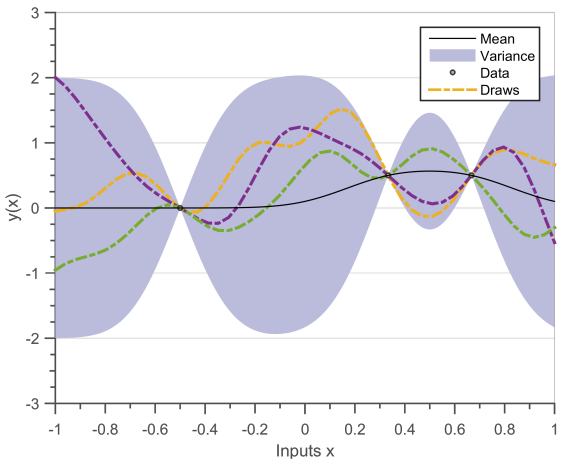
$$\Pr[y(\boldsymbol{x})] = GP(0, k(\boldsymbol{x_1}, \boldsymbol{x_2}) + \sigma_n^2 \delta_{x_1 x_2}) \tag{2.13}$$

Here, $\delta_{x_1 x_2}$ is a Kronecker delta function, which is one if $x_1 = x_2$ and zero otherwise. Since the noise is independent for each observation, there is no noise term for covariances across inputs. The Matlab code 2.6 shows a method to add independent noise into a SE covariance function.

Noise model



(a) Posterior distribution for the case of noiseless observations. Prior is a GP with mean zero and covariance as SE Kernel with ($\boldsymbol{\theta} = [1, 0.2]$), 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 ($\boldsymbol{\theta} = [1, 0.2]$), 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.

```
function covariance = noisySEKernel(theta, x1, x2)
% theta(1): is the amplitude hyperparameter
% theta(2): is the length-scale hyperparameter
% theta(3): is the noise hyperparameter

if x1 == x2 % addition of white noise
    covariance = SEKernel(theta(1, 2), x1, x2) + theta(3)^2;
else % normal SE kernel
    covariance = SEKernel(theta(1, 2), x1, x2);
end

end
```

Matlab Code 2.6: Matlab code for a noisy SE covariance function

The joint distribution of the training outputs $\boldsymbol{y}(\boldsymbol{X})$ and true physical process $\boldsymbol{f}(\boldsymbol{X}_*)$

according to the above prior becomes equation 2.14.

$$\Pr\begin{bmatrix}\boldsymbol{y}(\boldsymbol{X})\\ \boldsymbol{f}(\boldsymbol{X}_*)\end{bmatrix} = \mathcal{N}\left(\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}) + \sigma_n^2\boldsymbol{I}_{\boldsymbol{N}} & \boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}_*)\\ \boldsymbol{K}(\boldsymbol{X}_*,\boldsymbol{X}) & \boldsymbol{K}(\boldsymbol{X}_*,\boldsymbol{X}_*)\end{bmatrix}\right) \tag{2.14}$$

The difference between equation 2.14 and 2.10 is the addition of noise term $\sigma_n^2\boldsymbol{I}_{\boldsymbol{N}}$. Since the noise at each measurement point is assumed to be independent, it is multiplied to an identity matrix. To know how to add more complex noise models please refer to section 5.2.2. The posterior distribution of $\boldsymbol{f}(\boldsymbol{X}_*)$ can be calculated as follows:

$$\begin{aligned}\Pr(\boldsymbol{f}(\boldsymbol{X}_*) \mid \boldsymbol{X}_*, \boldsymbol{X}, \boldsymbol{y}(\boldsymbol{X})) = GP(&\boldsymbol{K}(\boldsymbol{X}_*,\boldsymbol{X})(\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}) + \sigma_n^2\boldsymbol{I}_{\boldsymbol{N}})^{-1}\boldsymbol{f}(\boldsymbol{X}),\\ &\boldsymbol{K}(\boldsymbol{X}_*,\boldsymbol{X}_*) - \boldsymbol{K}(\boldsymbol{X}_*,\boldsymbol{X})(\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}) + \sigma_n^2\boldsymbol{I}_{\boldsymbol{N}})^{-1}\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}_*))\end{aligned} \tag{2.15}$$

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 section 2.2. Due to the inclusion of noise in the prior, we see that the draws from posterior are not necessarily passing through the observed point.

Figure 2.4

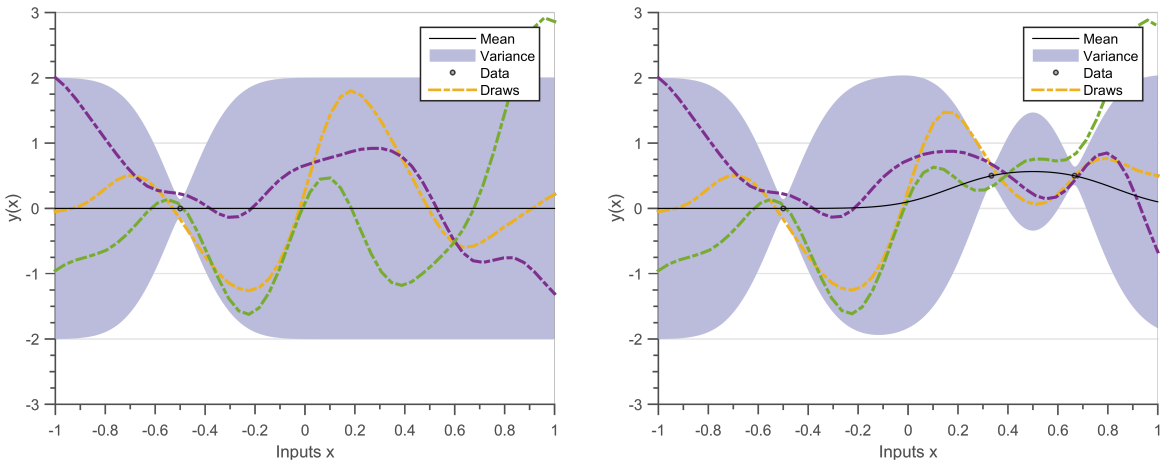
2.3.3 Interpretation of posterior

We will now introduce a short hand notation and replace the lengthy notation $\boldsymbol{K}(\boldsymbol{X_1},\boldsymbol{X_2})$ with $\boldsymbol{K}_{\boldsymbol{X_1X_2}}$ if it is a matrix, $\boldsymbol{k}_{\boldsymbol{X_1X_2}}$ if it is a vector, and $k_{X_1X_2}$ if it is a scalar. For the case where we have only one test point $\boldsymbol{x_*}$, we can write the predictive mean and variance in short-hand as:

$$\mathbf{E}[f(\boldsymbol{x}_*)] = \boldsymbol{k}_{\boldsymbol{Xx}_*}^T(\boldsymbol{K}_{\boldsymbol{XX}} + \sigma_n^2\boldsymbol{I})^{-1}\boldsymbol{y} \tag{2.16}$$

$$Cov[f(\boldsymbol{x}_*)] = k_{x_*x_*} - \boldsymbol{k}_{\boldsymbol{Xx}_*}^T(\boldsymbol{K}_{\boldsymbol{XX}} + \sigma_n^2\boldsymbol{I})^{-1}\boldsymbol{k}_{\boldsymbol{Xx}_*} \tag{2.17}$$

Precision Matrix Both the predictive mean (equation 2.16) and predictive covariance (equation 2.17) need the inverse of the covariance matrix $(\boldsymbol{K}_{\boldsymbol{XX}} + \sigma_n^2\boldsymbol{I})^{-1}$. The inverse of a covariance matrix is also known as the 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.



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

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 points.

$\mathcal{O}(N^3)$

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 invert the covariance matrix and perform predictions, more details are available in chapter 3.

Predicted mean The predictive mean is a linear combination of the observations y_i, and factors of $\boldsymbol{k}_{\boldsymbol{X}\boldsymbol{x}_*}^T(\boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}} + \sigma_n^2\boldsymbol{I})^{-1}$. Since a SE kernel $\boldsymbol{k}_{\boldsymbol{X}\boldsymbol{x}_*}^T$ decreases exponentially with distance, observations closer to x_* have more impact on the final prediction (equation 2.18).

$$\mathbf{E}[f(\boldsymbol{x}_*)] = \sum_{i=1}^{N} \boldsymbol{k}_{\boldsymbol{X}\boldsymbol{x}_*}^T(\boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}} + \sigma_n^2\boldsymbol{I})^{-1} y_i \tag{2.18}$$

The predictive mean can also be interpreted as a linear combination of the basis functions $k_{x_i x_*} \in \mathbb{R}$, and participation factors $(\boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}} + \sigma_n^2\boldsymbol{I})^{-1}Y$ (equation 2.19).

$$\mathbf{E}[f(\boldsymbol{x}_*)] = \sum_{i=1}^{N} k_{x_i x_*}(\boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}} + \sigma_n^2\boldsymbol{I})^{-1}\boldsymbol{y} \tag{2.19}$$

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 2.14). If the precision matrix is cached, then calculating the mean is an $\mathcal{O}(N)$ operation. $\mathcal{O}(N)$

Predicted variance The predicted variance is a combination of two terms, $k_{x_*x_*}$ which is the variance due to prior assumptions, and $-\boldsymbol{k}_{\boldsymbol{X}\boldsymbol{x}_*}^T(\boldsymbol{K}_{\boldsymbol{XX}} + \sigma_n^2\boldsymbol{I})^{-1}\boldsymbol{k}_{\boldsymbol{X}\boldsymbol{x}_*}$, which denotes the decrease in variance due to the observations. The predictive covariance of test targets $y(x_*)$ can be calculated by adding a noise term σ_n^2 in predictive covariance equation 2.17.

$$Cov[y(\boldsymbol{x}_*)] = k_{x_*x_*} - \boldsymbol{k}_{\boldsymbol{X}\boldsymbol{x}_*}^T(\boldsymbol{K}_{\boldsymbol{XX}} + \sigma_n^2\boldsymbol{I})^{-1}\boldsymbol{k}_{\boldsymbol{X}\boldsymbol{x}_*} + \sigma_n^2 \tag{2.20}$$

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 data-set ($\mathcal{D}$) 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 hyper-parameters. In order to automatically learn the hyper-parameters, we must perform model selection. Section 2.4 details how to fine-tune hyper-parameters to find an optimal prediction.

2.4 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 $\boldsymbol{\theta}$ 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 part II for discussion on how to choose covariance functions.

We define a new data-set $\mathcal{D}_2$ which will be used to compare predictions using different hyper-parameters. The function $f(x)$ (equation 2.21) is evaluated at 20 equidistant points between $x \in [-1, 1]$ and is corrupted by an independent white noise having variance $\sigma_{noise}^2 = 0.1^2$. Matlab code 2.7 is a sample code to generate the data-set $\mathcal{D}_2$. *data-set* $\mathcal{D}_2$

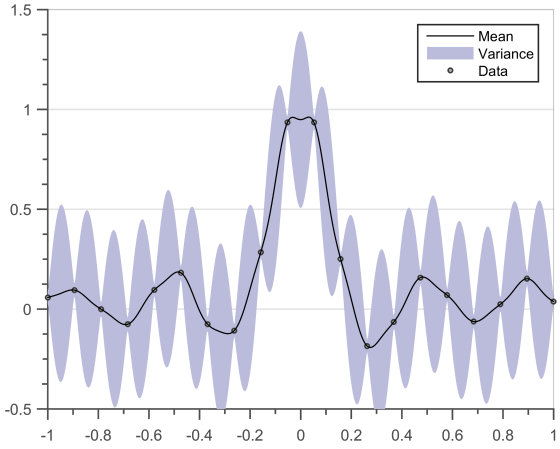
$$f(x) = \frac{sin(5\pi x)}{5\pi x} \tag{2.21}$$

```
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);
```

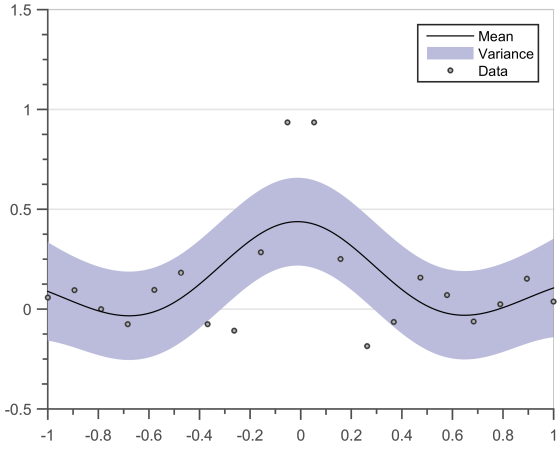
Matlab Code 2.7: Code for data-set D2

Figure 2.5

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.



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



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

To estimate the hyper-parameters in a pure Bayesian framework, we should put a prior over our hyper-parameters $\Pr[\boldsymbol{\theta}]$ and use Bayes Rule to estimate the posterior $\Pr[\boldsymbol{\theta} \mid \boldsymbol{X}, \boldsymbol{y}]$ over our data-set (just like we did in section 1.4).

$$\Pr[\boldsymbol{\theta} \mid \boldsymbol{X}, \boldsymbol{y}] = \frac{\Pr[\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta}] \times \Pr[\boldsymbol{\theta}]}{\Pr[\boldsymbol{y} \mid \boldsymbol{X}]} \tag{2.22}$$

The normalizing constant in the denominator is given by the following integral.

$$\Pr[\boldsymbol{y} \mid \boldsymbol{X}] = \int \Pr[\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta}] \Pr[\boldsymbol{\theta}] d\boldsymbol{\theta} \tag{2.23}$$

Depending on the functional form of the covariance function the normalizing constant may or may not be analytically integrable. We are effectively integrating (marginalizing) out the hyper-parameters from our probability distribution (appendix A.4). Hence, this approach becomes intractable and several sampling schemes have been proposed to calculate the posterior of hyper-parameters [Osborne 2010, Neal 2011].

LOO-CV

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 2005, 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 integrate out the latent functions f to calculate the marginal likelihood, and maximize it to find optimal hyper-parameters [MacKay 2003]. The probability of generating the observations ($\boldsymbol{y}$) at the points ($\boldsymbol{X}$) from a prior (defined by $k(\boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{\theta})$) is called the marginal likelihood $\Pr[\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta}]$. 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 2.13 and 2.14 we get:

$$\begin{aligned} \Pr[\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta}] &= \mathcal{N}(0, \boldsymbol{K(X, X')} + \sigma_n^2 \boldsymbol{I}) \\ &= \frac{1}{\sqrt{(2\pi)^N \boldsymbol{K_{XX}}}} exp^{-\frac{1}{2}\boldsymbol{y}^T \boldsymbol{K_{XX}} \boldsymbol{y}} \end{aligned} \tag{2.24}$$

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 [Rasmussen 2005].

$$\log(\Pr[\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta}]) = -\frac{1}{2}\boldsymbol{y}^T[\boldsymbol{K_{XX}} + \sigma_n^2 \boldsymbol{I}]^{-1}\boldsymbol{y} - \log\left|\boldsymbol{K_{XX}} + \sigma_n^2 \boldsymbol{I}\right| - \frac{N}{2}\log(2\pi) \tag{2.25}$$

Trade-off

The log marginal likelihood is composed of three terms a data-fit term ($-\frac{1}{2}\boldsymbol{y}^T[\boldsymbol{K_{XX}} + \sigma_n^2 \boldsymbol{I}]^{-1}\boldsymbol{y}$), a model complexity term ($\log|\boldsymbol{K_{XX}} + \sigma_n^2 \boldsymbol{I}|$), and a normalization term ($\frac{N}{2}\log(2\pi)$). It performs a trade-off between the data-fit term and the complexity

term. 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 2.8 defines the function 'logMarginalLikelihood' (equation 2.25), which is maximized later in the same code.

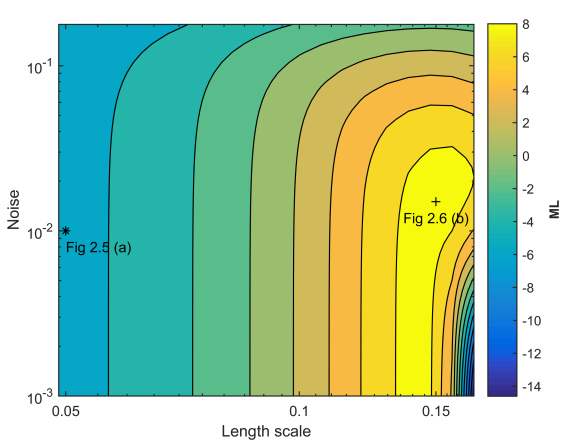
```
function logMarginalLikelihood = logMarginalLikelihood(theta,
    covarianceFunction, x, y)
N = length(x);

% Gram Matrix
gramMatrixXX = evaluateGramMatrix(covarianceFunction, theta, x,
    x);

% 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];
% Amplitude hyp = 1
% Length Scale = 0.2

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

Matlab Code 2.8: Optimizing the Log Marginal Likelihood

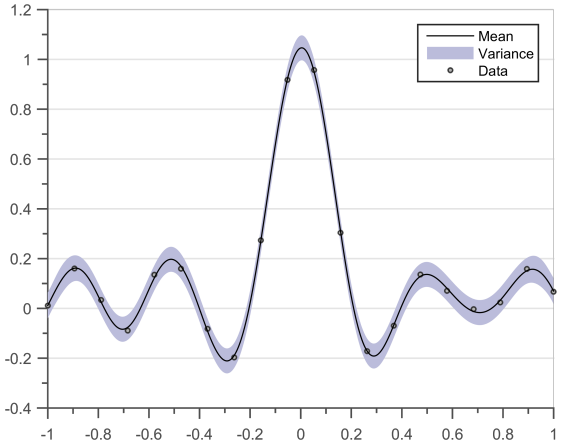
Figure 2.6(a)

Figure 2.6(a) shows the contours of the marginal likelihood with respect to length-scale $\theta_{lengthScale}$ and noise σ_n hyper-parameters. The data-set ($\mathcal{D}_2$) 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 the

same data-set as used in figure 2.5 but for the hyper-parameters where marginal likelihood is maximum.



(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

The marginal likelihood could have multiple maxima in the space of hyper-parameters. Since we do not know *a-priori* the multi-modality of log marginal likelihood, proper care should be taken while optimizing the hyper-parameters. [Forrester 2008] propose the use of a global optimizer (genetic algorithm) to find the global optimum, while [Le Gratiet 2013, Bouhlel 2016a] propose the use of cross validation to find the global optimum.

Multi modality

2.5 Summary and 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.

Section 2.2 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

Section 2.2

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 2.2.3). 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 2.2.4).

Section 2.3

Section 2.3 describes how to calculate the posterior distribution. The posterior is the conditional distribution ($\Pr[f(\boldsymbol{x_*}) \mid \boldsymbol{y}, \boldsymbol{X}, \boldsymbol{\theta}]$) for an assumed Prior distribution ($\Pr[f] = GP(0, k(\boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{\theta}))$ and a set of observed data-points ($\mathcal{D} = \{\boldsymbol{X}, \boldsymbol{y}\}$). 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 2.3.1)and presence (subsection 2.3.2) of noise in observations.

Section 2.4

Given a functional form of the covariance, section 2.4 shows the importance of choosing the correct hyper-parameters. In a pure Bayesian framework the posterior distribution of the hyper-parameters should be calculated ($\Pr[\boldsymbol{\theta} \mid \boldsymbol{X}, \boldsymbol{y}]$), but this is formally non-computable, 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 $\Pr[f] = GP(0, k(\boldsymbol{x_1}, \boldsymbol{x_2}, \theta)$ 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 $[\boldsymbol{K_{XX}} + \sigma^2_{noise}\boldsymbol{I}]^{-1}$ is an important task in calculating the marginal likelihood, posterior mean and posterior covariance. 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 [4].

In the certification phase of aircraft design, we have access to millions ($N \sim 10^6$) of training data-points which depend on several different parameters. [Bouhlel 2016b] tackle the problem of building models across high-dimensional input spaces. The next chapter describes few methods of performing approximate inference which scales GPs to $N \sim 10^6$ or more data-points.

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

Chapter 3

Scaling up Gaussian Process Regression

Résumé

Le calcul de lois a posteriori dans les GPs devient difficile pour les ensembles de données de grande taille. Le calcul de la matrice de précision est une opération de complexité $\mathcal{O}(N^3)$, mettant une limite de $N \sim 10^4$ points de données pour la construction de modèles. Ce chapitre décrit l'état de l'art pour l'étendue des GPs pour les tâches de régression. Il existe deux méthodes pour étendre des GPs, la première méthode 'Sparse GPs' utilisent un ensemble des points inductifs pour réduire le coût de calcul de la matrice de précision. La seconde est appelée 'Distributed GPs' elle divise l'ensemble de données en sous-ensembles plus petits, en distribuant le modèle en plusieurs lots.

Les méthodes 'Sparse GPs' utilisent l'approximation de Nyström, récrivant la matrice de Gram, diminuant ainsi la complexité de calcul à $\mathcal{O}(NM^2)$ (M $\ll$ N) , M étant le nombre de points inductifs. Grâce à des expériences sur les données synthétiques, on peut montrer que on peut fixer $M \sim N/10$ quand les points inductifs sont distribués aléatoirement et $M \sim N/50$ quand les emplacements des points inductifs sont optimisés. Cette approximation pousse la limite de régression GP à $N \sim 10^6$ points de données.

"Distributed GPs" distribue les tâches de régression GP dans plusieurs lots, diminuant ainsi la complexité informatique à $\mathcal{O}(NP^3)$ (P $\ll$ N), P étant le nombre des points dans un lot. Grâce à des expériences sur les données synthétiques, nous démontrons que $P \sim N/100$ n'affecte pas beaucoup la précision de la régression. En fait, nous pouvons réduire davantage P si nous permettons la répétition de points entre les lots. Cela permet d'étendre le GP à n'importe quel nombre de points de données.

3.1 Introduction

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 $\boldsymbol{K_{XX}}$ is of size $N \times N$ and $\mathcal{O}(N^3)$ 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 $\boldsymbol{K_{XX}}$.

Low-rank approximation

Let us take the example of a SE kernel, for a high value of length-scale, the Gram matrix ($\boldsymbol{K_{XX}}$) 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 3.2) use a set of inducing points to compress the information of the several observations through the low-rank approximation.

Block-diagonal 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 3.3) 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 3.2 describes the Sparse Approximations detailing several methods of choosing inducing points and then performing experiments on a toy data-set. Section 3.3 describes the Distributed GPs methodology detailing several methods for merging of experts and then performing experiments on the same toy data-set.

3.2 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 $\boldsymbol{X}^{\boldsymbol{m}} = \{\boldsymbol{x_1^m}, \boldsymbol{x_2^m}, \ldots, \boldsymbol{x_M^m}\}^T$, such that $M < N$. The points $\boldsymbol{X}^{\boldsymbol{m}}$ can be a subset of training inputs in the input space.

3.2.1 Nyström Approximation

Using Nyström approximation the Gram matrix can be approximated as equation 3.1 [Quiñonero-Candela 2005, Seeger 2003], for more detailed derivation refer to

[Williams 2001].

$$\boldsymbol{K}_{Nystorm}(\boldsymbol{X},\boldsymbol{X}) = \boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}^m)\boldsymbol{K}(\boldsymbol{X}^m,\boldsymbol{X}^m)^{-1}\boldsymbol{K}(\boldsymbol{X}^m,\boldsymbol{X}) \tag{3.1}$$

Here, $\boldsymbol{K}(\boldsymbol{X}^m,\boldsymbol{X}^m)$ is a $M \times M$ Gram matrix evaluated at inducing points $\boldsymbol{X}^m$, $\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}^m)$ is an $N \times M$ Gram matrix between training points and inducing points. The inversion of approximate Gram 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 Gram matrix.

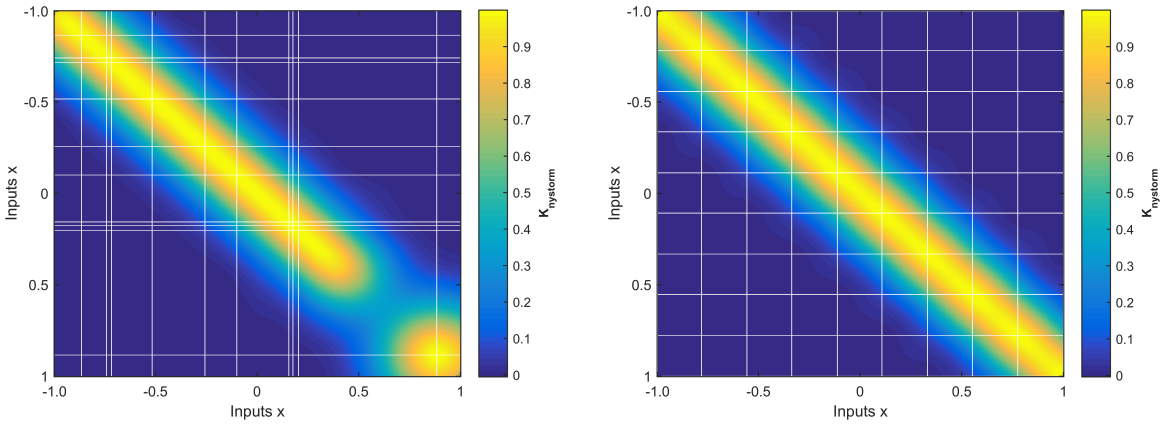
```
% Function to evaluate the approximate gram matrix
function gramMatrix = evaluateNystromGramMatrix(
   covarianceFunction, theta, xm, x1, x2)

        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 computation using Nyström approximation of the matrix in figure 2.1(a) at the input points $\boldsymbol{X}^* = \{[0 : 0.02 : 1]\}$. The inducing points $\boldsymbol{X}^m$ 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 the Gram matrix degrades (eg. at $x \sim 0.5$). Figure 3.1(b) is again an

Figure 3.1(a)

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 $\boldsymbol{X}^*$. 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 ($\boldsymbol{\theta} = [1, 0.2]$) (figure 2.1(a)) at the input points $\boldsymbol{X}^* = \{[0 : 0.02 : 1]\}$. 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 ($\boldsymbol{\theta} = [1, 0.2]$) (figure 2.1(a)) at the input points $\boldsymbol{X}^* = \{[0 : 0.02 : 1]\}$. 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 Fully Independent Training Conditional (FITC) approach which corrects the diagonal terms of the Gram matrix and improves the prediction capabilities (equation 3.2).

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

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

Posterior

The posterior distribution for the approximate prior can be derived similarly to section 2.3 [Williams 2001] and is a Gaussian. The predictive mean, and predictive variance are written as equation 3.3 and equation 3.4. Here, $\boldsymbol{K_{Approximate}}(\boldsymbol{X}, \boldsymbol{X'})$ can be the approximated Gram matrix either from the Nyström approximation (equation 3.1) or the FITC approximation (equation 3.2).

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

$$Cov[f_{approximate}(\boldsymbol{x_*})] = k_{x_*x_*} - \boldsymbol{k}_{\boldsymbol{X}\boldsymbol{x_*}}^T(\boldsymbol{K_{Approximate}}(\boldsymbol{X}, \boldsymbol{X}) + \sigma_n^2\boldsymbol{I})^{-1}\boldsymbol{k}_{\boldsymbol{X}\boldsymbol{x_*}} \tag{3.4}$$

By approximating the $\boldsymbol{K(X, X)}$ using inducing points, we have effectively changed the GP prior. This means that $\boldsymbol{X^m}$ have also become the hyper-parameters of our GP. Hence we should fine-tune locations of $\boldsymbol{X^m}$ and the hyper-parameters $\boldsymbol{\theta}$ to obtain a good prediction of our data. The marginal likelihood for the approximate prior (equation 3.5) can be written similarly as equation 2.24.

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

The approximate marginal likelihood (equation 3.5) has more parameters ($\boldsymbol{X^m}$ and $\boldsymbol{\theta}$) to fine-tune when compared to the old, exact marginal likelihood (equation 2.25). Hence, the maximization of the marginal likelihood in equation 3.5 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, due to over-fitting. The variational approximation (detailed next) approach overcomes this issue of over-fitting by adding a regularization term penalizing over-fitting.

Over-fitting

3.2.2 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 ($\boldsymbol{X^m}$) 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 ($\boldsymbol{X^m}$) and $\boldsymbol{\theta}$. This minimization of KL divergence is equivalently expressed as the maximization of equation 3.6

KL divergence

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

Notice the similarity between equation 3.6 and 3.5. The novelty of the above objective function is that it contains a regularization term: $\frac{1}{2\sigma_n^2} Tr(\boldsymbol{K_{XX}} - \boldsymbol{K_{Nystorm}(X, X)})$. 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 ($\boldsymbol{K_{XX}} = \boldsymbol{K_{Nystorm}(X, X)}$), it means that the inducing variables can exactly reproduce the true Gram Matrix.

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

Posterior

The posterior distribution for variational inference approximation is the same as the one derived for Nyström approximation (equations 3.3 and 3.4), for a detailed derivation refer to [Titsias 2009]. The difference between Nyström approximation and variational approximation is the addition of regularization parameter ($\frac{1}{2\sigma_n^2} Tr(\boldsymbol{K_{XX}} - \boldsymbol{K_{Nystorm}(X, X)})$) while optimizing $\boldsymbol{X^m}$ and $\boldsymbol{\theta}$, this additional term reduces over-fitting.

3.2.3 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 section is GPML provided with [Rasmussen 2005] on MATLAB 2014b. All experiments were performed on an Intel quad-core processor with 4Gb RAM.

10 fold CV

10-fold Cross Validation (CV) will be used to assess the performance of the prediction. CV is a technique where the data-set 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 data-set 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 data-set, 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 $\boldsymbol{X} = \{[-1 : 0.002 : 1]\}$ by sampling a random function from a GP[2] with zero mean, SE covariance function ($\boldsymbol{\theta} = [1, 0.1]$) and noise $\sigma_n = 0.3$. Code 3.2 generates the data-set $\mathcal{D}_3$.

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

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);%
```

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

```
plot(xData, yData, '.')
```

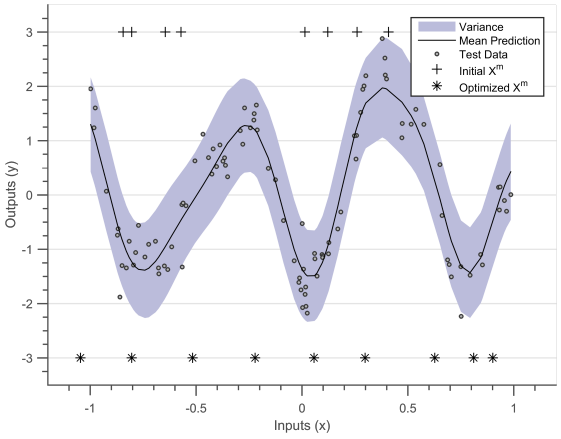
Matlab Code 3.2: Code for toy data-set 3

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. *Figure 3.2(a)*

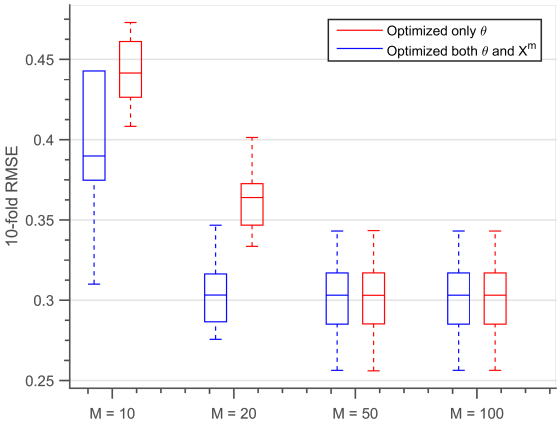
Notice the inducing points, while initially randomly distributed are later uniformly distributed due to optimization of marginal likelihood. In the case of dataset $\mathcal{D}_3$, since the training points are randomly distributed, uniformly distributed inducing inputs are better approximations of the 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 2.2.3) [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-set is $\sigma_n = 0.3$, it is the best achievable RMSE value. Models constructed when optimizing both $\boldsymbol{\theta}, \boldsymbol{X^m}$ 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 $\boldsymbol{\theta}$ will be sufficient to give a good prediction [Cao 2013], while if $M = \frac{N}{50}$ then both inducing points and hyper-parameters should be optimized. *Figure 3.2(b)*

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



(a) Posterior between a Nyström 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 $\boldsymbol{\theta}$ 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 $\boldsymbol{\theta}$ are optimized.

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

than $N > \mathcal{O}(10^6)$ by exploiting distributed architecture.

3.3 Distributed Gaussian Process

Netflix prize

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 $\mathcal{D}^{(i)} = \boldsymbol{X}^{(i)}, \boldsymbol{y}^{(i)}, i \in 1, \ldots N_{experts}$. Then each subset of data $\mathcal{D}^{(i)}$ learns an individual GP model, which are to 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 can also tie all the different experts using one single set of hyper-parameters. This is equivalent to assuming one single GP for the whole data-set, thus the experts are same but they each have access to different data subsets. 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 $\boldsymbol{\theta}$ and σ_n are same for all experts.

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

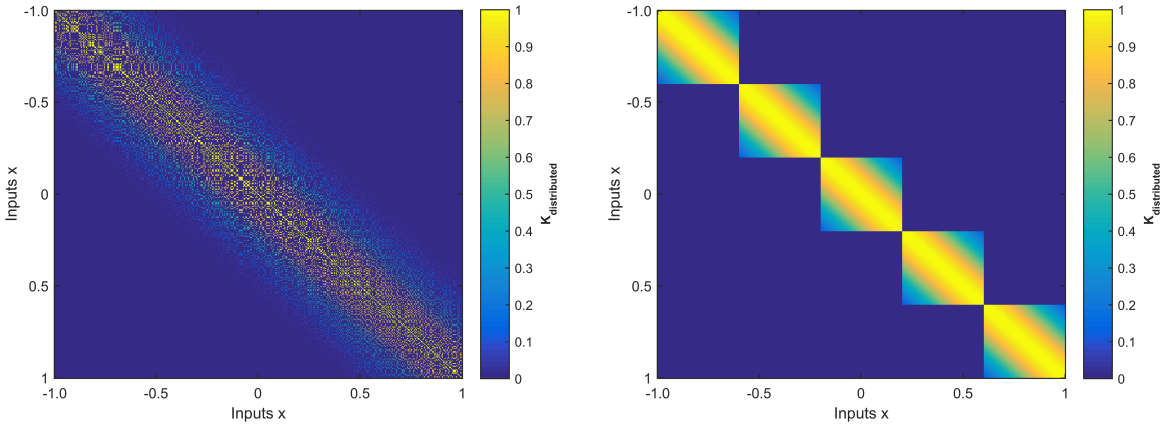
The sample code 3.3 presents a method to randomly clusterize data-set across 6 experts.

```
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 the above approximation, for a SE Kernel with ($\boldsymbol{\theta} = [1, 0.2]$) at 500 input points $\boldsymbol{X}^* = \{[0 : 0.02 : 1]\}$. 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 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 GPs approximation for a SE Kernel with ($\boldsymbol{\theta} = [1, 0.2]$) (figure 2.1(a)) at the input points $\boldsymbol{X}^* = \{[0 : 0.02 : 1]\}$. Points in the experts are distributed randomly, this gives the approximate Gram matrix scattered shape.

(b) Approximated Gram matrix using Distributed GPs approximation for a SE Kernel with ($\boldsymbol{\theta} = [1, 0.2]$) (figure 2.1(a)) at the input points $\boldsymbol{X}^* = \{[0 : 0.02 : 1]\}$. 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 have different data subsets, we can construct a posterior distribution for each expert (equations 3.8 and 3.9). In the following equations $m^{(i)}(\boldsymbol{x}_*)$ and $\sigma^{(i)}(\boldsymbol{x}_*)$ are the mean and covariance predictions from expert i at point $\boldsymbol{x}_*$.

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

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

3.3.1 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 (POE) 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].

rBCM

This thesis will use robust Bayesian Committee Machine (rBCM) model to combine the posterior distributions of experts [Deisenroth 2015]. The rBCM model is an aggregation 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.

$$m(y(\boldsymbol{x_*})) = (Cov(y(\boldsymbol{x_*})))^{-2} \sum_i \beta_i (\sigma^{(i)})^{-2} m^{(i)}(\boldsymbol{x_*}) \tag{3.10}$$

$$Cov(y(\boldsymbol{x_*}))^{-2} = \sum_i \beta_i \sigma^{(i)}(y(\boldsymbol{x_*})) + (1 - \sum_i \beta_i)(k_{x_* x_*})^{-2} \tag{3.11}$$

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

Marginal Likelihood

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 hyper-parameters. By approximating the $\boldsymbol{K(X, X)}$ 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 the prediction capabilities of GP. The below equation 3.12 describes the formulation for marginal likelihood.

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

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.

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

```
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 GPs

k-means

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 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.

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

3.3.2 Experiments

We again conduct experiments on a toy data-set, this time to observe the accuracy of the Distributed GPs approximation for a 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 ($\mathcal{D}_3$) as used in section 3.2.3 was used to perform the experiments in this section (1000 data-points from $\Pr[\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta}, \sigma_n] = GP(0, \boldsymbol{K_{SE}}(\boldsymbol{X}, \boldsymbol{X'}, \boldsymbol{\theta} = [\boldsymbol{1}, \boldsymbol{0.1}]) + (0.3)^2\boldsymbol{I})$.

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 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.

Figure 3.4(a)

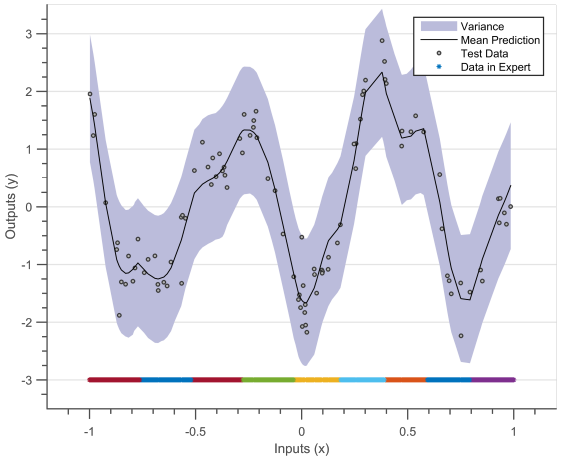
The points across experts are uniformly distributed, as can be observed by the coloring scheme. Since the training points are almost uniformly distributed, the k-means algorithm will cluster the points uniformly. The training and the test data-set for figures 3.4(a) and 3.2(a) are kept intentionally same. We can thus compare the interpolating properties of both Sparse GPs approximation (figure 3.4(a)) and Distributed GPs (figure 3.2(a)). The Nyström approximation has a global smooth shape while the Distributed GPs approximation retains the local features of the data-set.

Figure 3.4(b) are 10-fold RMSE box-plots for different values 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 toy data-set 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 $P = \frac{N}{100}$, and optimizing the hyper-parameters ($\boldsymbol{\theta}$) is a good enough approximation.

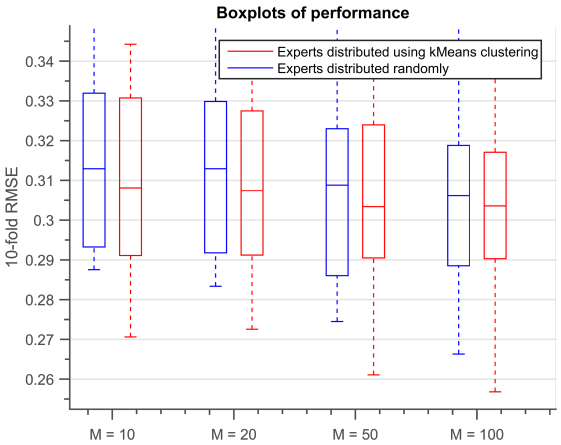
Figure 3.4(b)

3.4 Summary and 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-the-art for scaling up GPs for regression tasks. There exist two methods to scale up GPs, the first called Sparse GPs, they use a set of inducing inputs to reduce the computational cost of calculating the precision matrix. The second called Distributed GPs they divide the



(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 GPs approximation on a toy data-set of size $N = 1000$

data-set into smaller subsets called experts, distributing the model building into several batches.

Conclusions

Sparse methods use Nyström approximation rewriting the Gram matrix, thereby reducing the computational complexity to $\mathcal{O}(NM^2)$ (M $\ll$ N), M being the number of inducing points. Through experiments on a toy data-set, 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 $\ll$ N), P being the number of points in an expert. Through experiments on a toy data-set 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.

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 hyper-parameters. Although GPs suffer in presence of large data-sets, there exist several approximate methods to scale GPs to millions of data-points.

During the detailed design phase of an aircraft design cycle, high-fidelity code is used to explore the design space. This code is costly, mostly due to fine granularity of the meshes used. If we wish to use a surrogate model to explore the design space, then that surrogate model should be able to handle the large number of mesh points. Thankfully, due to the methods described in the current chapter we can scale GP regression to $N \sim 10^6$. We will demonstrate this capability by running a GP regression on millions of data-points in a CFD mesh (section 5.3.5). This surrogate model was used in a recent Airbus flight test campaign to compare pressures predicted from a high-fidelity CFD computation to pressures measured on the wing, in real time.