Part III

Incorporating multiple outputs in Gaussian Process Regression

Chapter 6

Multi-task learning for extrapolation

Résumé

Dans la partie III, nous nous intéressons à l'apprentissage de relations entre plusieurs sorties. Une régression sur les sorties multiples peut être divisé en deux catégories, lorsque nous souhaitons découvrir des relations sur plusieurs sorties, et lorsque nous voulons appliquer une relation connue entre les sorties (en tant que biais) dans notre algorithme d'apprentissage.

Dans ce chapitre, nous verrons comment effectuer la régression GP en présence de sorties multiples. Une méthode simple d'apprentissage d'un modèle de régression pour les sorties multiples est de supposer le nombre de sorties comme une dimension supplémentaire (cette dimension supplémentaire est une variable catégorielle). Cette hypothèse est bénéfique pour deux raisons, nous avons effectivement augmenté le nombre de points de données (figure 6.2.1), puis des fonctions de covariances pour plusieurs dimensions (section 5.3) peuvent être également appliquées au cas des sorties multiples (section 6.2.1).

Si aucune information n'est disponible sur la relation entre les sorties, nous pouvons apprendre la relation entre les sorties en utilisant une covariance 'simple MTGP' (section 6.2.2), un 'Linear Model Of Coregionalization' (section 6.2.3), ou un 'convoluted GP' [Alvarez 2011]. Si nous savons *a-priori* que les sorties proviennent de modèles de fidélité multiples, nous pouvons utiliser la fonction de covariance jointe proposée par [Kennedy 2000] (section 6.2.4).

Dans la section 6.4, nous avons montré comment les modèles GP multi-fidélité peuvent être utilisés pour effectuer une interpolation ou une extrapolation des données expérimentales du modèle. Nous incluons un terme d'erreur et un terme de translation

dans un modèle multi-fidélité normal pour tenir compte des différences entre le modèle de simulation et les données expérimentales. Dans des travaux futurs, nous souhaitons quantifier la performance des différentes méthodes d'extrapolation et décider de la méthode à choisir dans un scénario particulier.

6.1 Introduction

In the current part (part III) we are interested in learning relationships between multiple outputs. Performing regression on multiple-outputs can be split into two categories, first when we wish to discover relationships across multiple outputs, and second when we wish to enforce a known relationship between outputs (as a bias) into our learning algorithm.

Multi-Task Learning

When no relationship is known between outputs, we are interested in discovering these relationships. Several real-world problems often exhibit strong correlations between output variables, for example, correlations across spatial coordinates (x, y, z) in an experiment. This setting is also called 'Multi-Task Learning' in the machine learning literature. Learning relationships across outputs can improve the efficiency and prediction accuracy of the models when compared to training the outputs individually. This is because learning together outputs which are related, effectively increases the number of data-points, thereby providing more data for the learning algorithm [Caruana 1998]. In the GP literature these models are called 'Multi-Task GP' (MTGP) [Alvarez 2011, Bonilla 2008, Boyle 2005], while in the kriging literature they are called as Cokriging [Helterbrand 1994, Chilès 1999, Ver Hoef 1998].

Enforcing relationships

When a relationship between the outputs is known, enforcing such a relationship in a learning algorithm will mean that we have implemented a correct bias. The outputs can be results from computer codes of varying accuracy [Kennedy 2000, Forrester 2007, Le Gratiet 2013], they can be related through a known equation[1] [Ginsbourger 2013, Särkkä 2011], or they can be related through a known computer code[2] [Constantinescu 2013].

We split the part III of this thesis into two chapters. The current chapter (chapter 6) will describe the basics of MTGP, present a model to interpolate a series of multi-fidelity simulations, and then propose a method to extrapolate experimental data in the presence of simulation models. The next chapter (7) will describe how to encode known relationships between outputs, the most popular form of this model is called as Gradient Enhanced Kriging (GEK) in the literature.

The main contribution of this chapter is two-fold we first compare the accuracy of

[1] For example if we know that $acceleration = d(velocity)/d(time)$ and we measure both velocity and acceleration. How can we enforce the known equation into our model?

[2] For example stress and forces in a CSM code

prediction of different MTGP models. Secondly, we expand the multi-fidelity formulation provided by [Kennedy 2000] to the case of extrapolating experimental data using a simulation model. This type of model is useful during aircraft certification when there is a need to extrapolate experimental data.

The current chapter unfolds as follows, section 6.2 describes the various methods available in the literature to perform Multi-Task GP (MTGP). Section 6.2.4 describes MTGP model in presence of outputs coming from simulations with different accuracy. Section 6.3 then compares the prediction capabilities of a normal MTGP with multi-fidelity GP. Finally, section 6.4 describes our model of extrapolation for certification. This chapter is influenced from the works of [Forrester 2007, Alvarez 2011, Bonilla 2008, Boyle 2005, Kennedy 2000, Le Gratiet 2013].

6.2 Multi-task Gaussian Process

Suppose we have a D_{inputs}-dimensional input space and a $D_{outputs}$-dimensional ($D_{outputs} > 1$) output space.

$$\boldsymbol{x_i^j} = [x_{i1}^j, x_{i2}^j, \ldots, x_{iD_{inputs}}^j] \tag{6.1}$$

Notations

Here, x_i^j is the i^{th} input point of the j^{th} output, which is a horizontal vector with D_{inputs} components, each component representing a scalar value. The training data-set for the j^{th} output[3] can be then written as $\{(x_i^j, y_i^j)\}$ for $i \in [1; N_j]$. Here N_j is the number of measurement points for the j^{th} output, while $x_i^j \in \mathbb{R}^{D_{inputs}}$ and $y_i^j \in \mathbb{R}$.

$$\boldsymbol{X^j} = \begin{bmatrix} \boldsymbol{x_1^j} \\ \boldsymbol{x_2^j} \\ \vdots \\ \boldsymbol{x_{N_j}^j} \end{bmatrix} ; \quad \boldsymbol{y^j} = \begin{bmatrix} y_1^j \\ y_2^j \\ \vdots \\ y_{N_j}^j \end{bmatrix} \tag{6.2}$$

Here, $\boldsymbol{X^j}$ is the matrix containing all input points of the j^{th} output such that $\boldsymbol{X^j} \in \mathbb{R}^{N_j \times D_{inputs}}$. While $\boldsymbol{y^j}$ is the vector containing all the output values for the j^{th} output such that $\boldsymbol{y^j} \in \mathbb{R}^{N_j}$. If $\Sigma N_j = N_{joint}$ for $\forall j \in [1, D_{outputs}]$. Then N_{joint} represents the total number of training points for all the outputs combined

[3]The superscript is used to denote the 'number of output' and does not denote the power.

6.2.1 An extra dimension

One simple way of building a joint model for multiple outputs is by treating them as a single-output GP but with an extra dimension, the extra dimension denoting the output number (equation 6.3) [Osborne 2010].

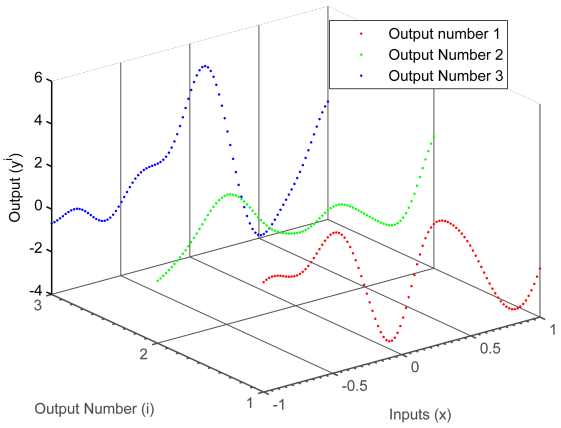


Figure 6.1: The outputs can also be represented as an extra dimension.

The figure 6.2.1 demonstrates how the different outputs can be treated as a single output coming from different dimensions. We can thus write the new input point as equation 6.3.

$$\boldsymbol{x_i^j} = [x_{i1}^j, x_{i2}^j, \ldots, x_{iD_{inputs}}^j, j] \tag{6.3}$$

Categorical variable

Note, the new input point has an extra dimension which denotes its output number. The extra dimension (j) is a categorical variable and hence has no measure of distance. Henceforth we define the joint output vector $\boldsymbol{y_{joint}}$ such that all the output values are stacked one after the other, while $\boldsymbol{y_{joint}} \in \mathbb{R}^{N_{joint}}$. Similarly, we define the joint input matrix as $\boldsymbol{X_{joint}}$, having one extra dimension representing the output number, such that $\boldsymbol{X_{joint}} \in \mathbb{R}^{N_{joint} \times (D_{inputs}+1)}$.

$$\boldsymbol{X_{joint}} = \begin{bmatrix} \boldsymbol{X^1}, \mathbb{1}_{\boldsymbol{N_1}} \times 1 \\ \boldsymbol{X^2}, \mathbb{1}_{\boldsymbol{N_2}} \times 2 \\ \vdots \\ \boldsymbol{X^{D_{outputs}}}, \mathbb{1}_{\boldsymbol{N_{D_{outputs}}}} \times \boldsymbol{D_{outputs}} \end{bmatrix}; \quad \boldsymbol{y_{joint}} = \begin{bmatrix} \boldsymbol{y^1} \\ \boldsymbol{y^2} \\ \vdots \\ \boldsymbol{y^{D_{outputs}}} \end{bmatrix} \tag{6.4}$$

Here, $\mathbb{1}_{\boldsymbol{N_j}}$ denotes a vector of ones to be multiplied with the 'number of output'. The size of $\mathbb{1}_{\boldsymbol{N_j}}$ is $1 \times N_j$, N_j is the number of data-points for the j^{th} output. Since the multi-output GP can be represented as a single output GP with an extra dimension, we can use all the kernel making techniques discussed in section 5.3 to the current problem. For a

case of 2 outputs we can write a zero mean GP prior for the full set of inputs and outputs $\{\boldsymbol{X_{joint}}, \boldsymbol{y_{joint}}\}$, as equation 6.5.

$$\begin{aligned}\Pr[\boldsymbol{y_{joint}}] &= \Pr\begin{bmatrix}\boldsymbol{y_{joint}}(x,1)\\ \boldsymbol{y_{joint}}(x,2)\end{bmatrix}\\ &= GP\left[\begin{pmatrix}0\\0\end{pmatrix}, \begin{pmatrix}Cov(\boldsymbol{y_{joint}}(x,1),\boldsymbol{y_{joint}}(x,1)) & Cov(\boldsymbol{y_{joint}}(x,1),\boldsymbol{y_{joint}}(x,2))\\ Cov(\boldsymbol{y_{joint}}(x,2),\boldsymbol{y_{joint}}(x,1)) & Cov(\boldsymbol{y_{joint}}(x,2),\boldsymbol{y_{joint}}(x,2))\end{pmatrix}\right]\end{aligned} \tag{6.5}$$

Here, $Cov(\boldsymbol{y_{joint}}(x,j),\boldsymbol{y_{joint}}(x,j))$ is called the auto-covariance between observations of the j^{th} output, while $Cov(\boldsymbol{y_{joint}}(x,2),\boldsymbol{y_{joint}}(x,1))$ is called cross-covariance between the 2^{nd} and the 1^{st} output. Once we have the functional forms of the covariance functions, we can equivalently calculate the Gram matrix ($\boldsymbol{K_{XX}}$), by replacing the values of individual input points. The upcoming sections describe different forms of covariance functions between outputs and their corresponding family of functions.

Equation 6.5

The sections 6.2.2 and 6.2.3 describe two simple covariance function forms when we are interested in discovering relationship between outputs, while section 6.2.4 describes how to impose prior information of multi-fidelity among outputs.

6.2.2 Simple Multi-task kernel

One simple method to calculate the auto and cross-covariance functions is by simply multiplying the covariance functions for the inputs dimensions with the covariance functions for the output dimension (equation 6.6), refer to [Bonilla 2007] for more clarity .

$$Cov(\boldsymbol{y_{joint}}(x_1,a),\boldsymbol{y_{joint}}(x_2,b)) = k_{output}(a,b) \times k_{input}(x_1,x_2) \tag{6.6}$$

There exists one issue though, the extra dimension is a categorical variable and hence has no measure of distance, i.e. the difference between two output numbers a and b is not defined (equation 6.6). [Bonilla 2007] define the covariance across outputs as equation 6.7, this makes the matrix $\boldsymbol{K_{output}}$ positive definite and hence a valid covariance matrix. $k_{output}(a,b)$ denotes the (a^{th}, b^{th}) coordinate of the matrix $\boldsymbol{K_{output}}$.

$$\boldsymbol{K_{output}} = \boldsymbol{K_{lower}}\boldsymbol{K_{lower}^T} \tag{6.7}$$

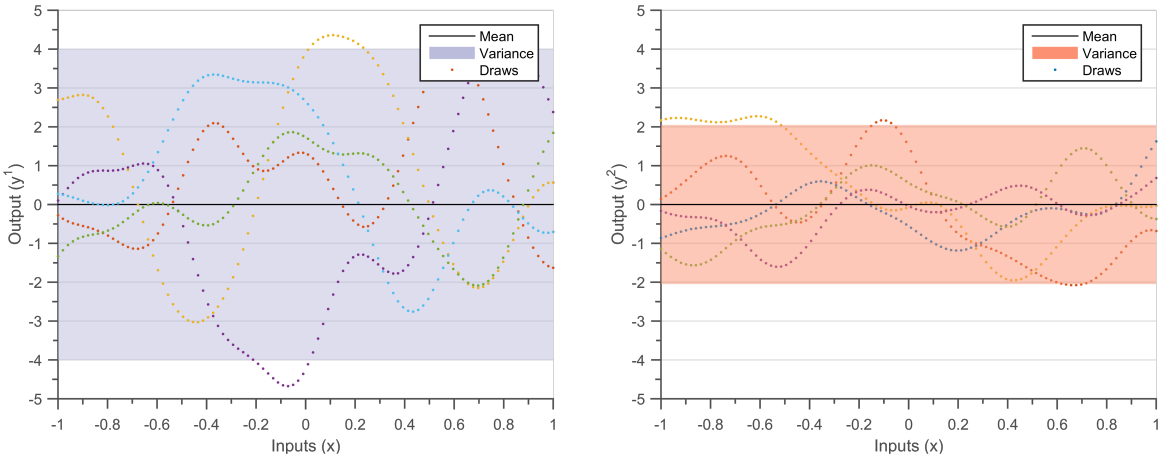
The hyper-parameters of above prior are the parameters of the lower triangular matrix L, and parameters of the input covariance function ($k_{input}(x_1^a, x_2^b)$). If $\boldsymbol{K_{output}}$ is a diagonal matrix, then there is no cross-covariance across outputs, this implies that the outputs are

independent of each other. In such a kernel design there is no transfer of information between outputs [Bonilla 2007, O'Hagan 1998].

If we measure the output 'a' at points $\boldsymbol{X^1}$ and output 'b' at not necessarily the same points $\boldsymbol{X^2}$, then we can write the Gram matrix for the above prior (equation 6.6) as equation 6.8.

$$\boldsymbol{K_{XX}} = \begin{bmatrix} k_{output}(a,a) \cdot \boldsymbol{K}_{input}(\boldsymbol{X^1}, \boldsymbol{X^1}) & k_{output}(a,b) \cdot \boldsymbol{K}_{input}(\boldsymbol{X^1}, \boldsymbol{X^2}) \\ k_{output}(b,a) \cdot \boldsymbol{K}_{input}(\boldsymbol{X^2}, \boldsymbol{X^1}) & k_{output}(b,b) \cdot \boldsymbol{K}_{input}(\boldsymbol{X^2}, \boldsymbol{X^2}) \end{bmatrix} \tag{6.8}$$

Here, '$\cdot$' denotes the multiplication between a scalar and a matrix.



(a) Five randomly drawn functions for output number 1, using the MTGP kernel proposed by [Bonilla 2007]

(b) Five randomly drawn functions for output number 2, using the MTGP kernel proposed by [Bonilla 2007]

Figure 6.2: Joint draws from a simple multi-task kernel function between outputs y^1 and y^2. The matrix $\boldsymbol{K_{output}} = [4, 0.2; 0.2, 1]$ while the covariance function between inputs $k_{input}(x_1, x_2)$ is a SE kernel with hyper-parameters ($\theta = [1, 0.2]$). The solid black line defines the mean function, shaded 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 6.3 varies faster when compared to figure 6.2 due to smaller length scale hyper-parameter in one of the latent functions.

Figure 6.2 is a joint draw from such a kernel function between outputs y^1 and y^2. The matrix $\boldsymbol{K_{output}} = \left(\begin{smallmatrix} 4 & 0.2 \\ 0.2 & 1 \end{smallmatrix}\right)$ while the covariance function between inputs $k_{input}(x_1, x_2)$ is a SE kernel with hyper-parameters ($\theta = [1, 0.2]$). We can observe that the variance of output y^1 is twice that of output y^2, this is because $\sqrt{k_{output}(1,1)/k_{output}(2,2)} = 2$.

6.2.3 Linear Model of Coregionalization

The next method considers the output variables as a linear combination of independent latent GPs. It is called the Linear Model of Coregionalization in kriging literature [Goovaerts 1997] or Semiparametric Latent Factor Model in MTGP literature [Seeger 2005]. Latent GPs are random variables which are not directly observed.

Suppose we have a set of L latent GPs $U(x) = \{u^1(x), u^2(x), \ldots, u^L(x)\}$, where $u^i(x)$ is a GP with covariance $k_u^i(x_1, x_2)$. Any linear combination of $u^i(x) \quad \forall i \in L$ is a viable GP (section 5.2.2), therefore we can define the GP of output y^j as equation 6.9.

$$y^j(x) = \sum_{i=1}^{L} \alpha^{ij} u^i(x) \tag{6.9}$$

The covariance function can thus be written as equation 6.10.

$$\begin{aligned} Cov(\boldsymbol{y_{joint}}(x_1, a), \boldsymbol{y_{joint}}(x_2, b)) &= Cov(\sum_{i=1}^{L} \alpha^{ia} u^i(x_1), \sum_{i=1}^{L} \alpha^{ib} u^i(x_2)) \\ &= \sum_{i=1}^{L} \alpha^{ia} \alpha^{ib} Cov(u^i(x_1), u^i(x_2)) \\ &= \sum_{i=1}^{L} \alpha^{ia} \alpha^{ib} k_u^i(x_1, x_2) \\ &= \sum_{i=1}^{L} \boldsymbol{K^i_{output}}(a, b) \cdot k_u^i(x_1, x_2) \end{aligned} \tag{6.10}$$

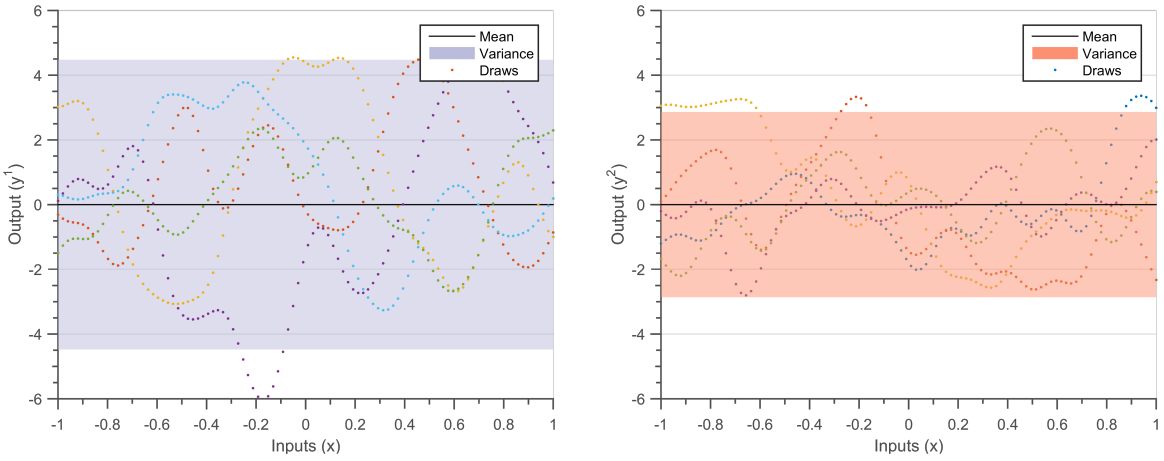
The covariance between the output dimension $\boldsymbol{K^i_{output}}$ is also called the coregionalization matrix, and is of size $D_{outputs} \times D_{outputs}$. It can be written as equation 6.11, making it a positive definite matrix [Mercer 1909].

$$\boldsymbol{K^i_{output}} = \begin{bmatrix} \alpha^{i1} \\ \alpha^{i2} \\ \vdots \\ \alpha^{iL} \end{bmatrix} \cdot \begin{bmatrix} \alpha^{i1} & \alpha^{i2} & \ldots & \alpha^{iL} \end{bmatrix} \tag{6.11}$$

Note, when $L = 1$ and $D_{outputs} > 1$ the Linear Model of Coregionalization is equivalent to the model proposed by [Bonilla 2007]. While, when $L > 1$ and $D_{outputs} = 1$ the Linear Model of Coregionalization model resembles an additive covariance of T individual covariances (section 5.2.2).

Figure 6.3

Figure 6.3 shows 3 joint draws from a 'Linear Model of Coregionalization' kernel function between outputs y^1 and y^2. We use two latent functions for this figure; the kernels between output dimensions are $\boldsymbol{K}^1_{\boldsymbol{output}} = \left(\begin{smallmatrix} 4 & 0.2 \\ 0.2 & 1 \end{smallmatrix}\right)$ and $\boldsymbol{K}^2_{\boldsymbol{output}} = \left(\begin{smallmatrix} 1 & 0.2 \\ 0.2 & 1 \end{smallmatrix}\right)$, while the covariance function between inputs are SE kernel with hyper-parameters $(\theta = [1, 0.2])$ for the first latent process and $(\theta = [1, 0.1])$ for the second latent process. The length-scale of the functions is determined by the smallest length-scale of the latent processes.



(a) Five randomly drawn functions for output number 1, using the MTGP kernel proposed by Linear Model of Coregionalization

(b) Five randomly drawn functions for output number 2, using the MTGP kernel proposed by Linear Model of Coregionalization

Figure 6.3: Joint draws from a 'Linear Model of Coregionalization' kernel function between outputs y^1 and y^2. We use two latent functions for figure; the kernels between output dimensions are $k^1_{output} = [4, 0.2; 0.2, 1]$ and $k^2_{output} = [1, 0.2; 0.2, 1]$, while the covariance function between inputs are SE kernel with hyper-parameters $(\theta = [1, 0.2])$ for the first latent process and $(\theta = [1, 0.1])$ for the second latent process. The solid black line defines the mean function, shaded 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 6.3 varies faster when compared to figure 6.2 due to smaller length scale hyper-parameter in one of the latent functions.

6.2.4 Multi-fidelity GP

Simulation models in engineering design often come with different orders of accuracy or fidelity. The increase in accuracy is generally associated to an increase in cost. This is because capturing the higher order effects means either more non-linear models or finer meshes. This creates an opportunity to reduce the overall cost of accurate predictions by reducing the number of runs of the costly, high-fidelity computation and launching several low-fidelity computations. We can treat the low-fidelity and high-fidelity predictions as

cases with multiple outputs. The prior information that we have here is that one output is more precise than another.

[Kennedy 2000, O'Hagan 1998] propose the first Cokriging model for multi-fidelity calculations. This method has been used extensively to make cheaper surrogate models and perform cheap optimization of a costly code [Forrester 2007, March 2012]. We will describe the methods proposed by [O'Hagan 1998] in this section.

Suppose we have $D_{outputs}$ simulation models $y^i(x)\ \forall\ i \in D_{outputs}$, ordered with increasing levels of accuracy. Since these are simulations of a computer code there is no noise in the outputs. [O'Hagan 1998] use the Markov property (equation 6.12) between a higher fidelity code $y^i(x)$ and lower fidelity code $y^{i-1}(x')$ for all $x \neq x'$. This assumption means that no further information about $y^i(x)$ can be extracted from lower level models if we have access to simulation results at the same input point.

$$Cov(y^i(x), y^{i-1}(x') \mid y^{i-1}(x)) = 0 \tag{6.12}$$

The Markov property assumption means that if there are two subsequent fidelity models $y^i(x)$ and $y^{i-1}(x)$. Then the high-fidelity model $(y^i(x))$ can be represented as a linear combination of the low-fidelity model $(y^{i-1}(x))$ and a model that represents the difference between the two $(\delta^i(x))$, thereby linearly seperating the effects of the two models. It can be expressed equivalently as the set of equations below.

Linearly separable

$$y^i(x) = r^{i-1}(x)y^{i-1}(x) + \delta^i(x) \tag{6.13}$$
$$y^{i-1}(x) \perp \delta^i(x) \tag{6.14}$$

Here, $r^{i-1}(x)$ is a continuous function in $x \in \mathbb{R}$, $\perp$ signifies independence between two GPs $(Cov(y^{i-1}(x), \delta^i(x)) = 0)$. If we have 2 sets of simulation results with varying fidelity $y^2(x)$ more accurate than $y^1(x)$, the above model gives rise to the following joint-covariance structure.

$$\begin{aligned} \boldsymbol{K_{XX}} &= \begin{bmatrix} k^1(x^1, x^1) & r(x^1)k^1(x^1, x^2) \\ r(x^2)k^1(x^2, x^1) & r(x^2)^2k^1(x^2, x^2) + k^\delta(x^2, x^2) \end{bmatrix} \\ &= \begin{bmatrix} k^1(x^1, x^1) & r(x^2)k^1(x^1, x^2) \\ r(x^1)k^1(x^2, x^1) & r(x^2)^2k^1(x^2, x^2) \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & k^\delta(x^2, x^2) \end{bmatrix} \end{aligned} \tag{6.15}$$

Here, x^1 are the input points for simulation y^1, and x^2 are the simulation points for output y^2, $k^1(x^1, x^2)$ is the covariance function for output y^1. [Kennedy 2000] have used a constant function for the value of $r(x)$ and an SE kernel for the covariances $k^1(x, x')$ and $k^\delta(x, x')$.

The model of multi-fidelity GPs in equation 6.15 is equivalent to a 'Linear Model of Coregionalization' model with two latent processes. The fist process has $\boldsymbol{K}^1_{\boldsymbol{output}} = \left(\begin{smallmatrix} 1 & r \\ r & r^2 \end{smallmatrix}\right)$ as covariance between outputs and $k^1_{inputs} = k^1$ as the covariance between inputs, while the second process has $\boldsymbol{K}^2_{\boldsymbol{output}} = \left(\begin{smallmatrix} 0 & 0 \\ 0 & 1 \end{smallmatrix}\right)$ as the covariance between outputs and $k^2_{inputs} = k^\delta$ as the covariance between inputs. We have thus linearly seperated the effects of both the models, such that the difference between the two is captured by the $k^\delta(x, x')$ covariance function.

Recursive model

Multi-fidelity GPs are expensive due to the cost of calculating the precision matrix. [Le Gratiet 2013] propose a recursive model for performing multi-fidelity, this model performs the learning for individual computer codes thereby breaking the Gram matrix into smaller sizes and reducing the computational complexity. They also extend the Cokriging model to several number of multiple outputs.

6.2.5 Posterior distribution

For simplicity let us take the case of two outputs y^1 and y^2. Suppose we measure the two outputs with some error (ϵ_{n1} and ϵ_{n2}), while the true physical process is defined by latent variables (f^1 and f^2). Then the relation between the output function, measurement error, and true physical process can be written as follows.

$$y^1 = f^1 + \epsilon_{n1} \tag{6.16}$$
$$y^2 = f^2 + \epsilon_{n2} \tag{6.17}$$

Here, ϵ_{n1} and ϵ_{n2} are measurement errors sampled from a white noise gaussian $\mathcal{N}(0, \sigma^2_{n1})$ and $\mathcal{N}(0, \sigma^2_{n2})$ respectively, then the joint error matrix can be denoted by $\boldsymbol{\Sigma}$;

$$\Sigma = \begin{bmatrix} \sigma^2_{n1} \times \mathbb{I}_{\boldsymbol{N_1}} & 0 \\ 0 & \sigma^2_{n2} \times \mathbb{I}_{\boldsymbol{N_2}} \end{bmatrix} \tag{6.18}$$

Noise model

Here, σ^2_{nj} is the variance of the measurement error sampled from a white noise Gaussian and $\mathbb{I}_{\boldsymbol{N_j}}$ is an identity matrix of size N_j (N_j are the number of data-points for j^{th} output). If we assume a zero mean GP for the above type of covariance functions (sections 6.2.2, 6.2.3 and 6.2.4). The prior for a noisy multi-task learning case can be written as equation 6.19.

$$\Pr[\boldsymbol{y_{joint}}(\boldsymbol{X_{joint}}))] = GP(0, \boldsymbol{K_{XX}} + \boldsymbol{\Sigma}) \tag{6.19}$$

If we want to make a prediction for the i^{th} output at the test point $\boldsymbol{x_*}$ ($\boldsymbol{X_*} = [\boldsymbol{x_*}, i]$).

Then the full joint prior can be expressed as equation 6.20.

$$\begin{bmatrix} \boldsymbol{y_{joint}}(\boldsymbol{X})) \\ \boldsymbol{f}(\boldsymbol{X_*})) \end{bmatrix} = GP\left[\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{bmatrix} \boldsymbol{K_{XX}} + \boldsymbol{\Sigma} & \boldsymbol{K_{XX_*}} \\ \boldsymbol{K_{X_*X}} & \boldsymbol{K_{X_*X_*}} \end{bmatrix}\right] \tag{6.20}$$

The posterior mean and variance, conditioned on the data-set based on the prior (equation 6.5) can then be derived as the set of equations below .

$$E[\boldsymbol{f}(\boldsymbol{X_*}))] = \boldsymbol{K_{X_*X}} \left(\boldsymbol{K_{XX}} + \boldsymbol{\Sigma}\right)^{-1} \boldsymbol{y_{joint}} \tag{6.21}$$

$$Cov(\boldsymbol{f}(\boldsymbol{X_*}))), \boldsymbol{f}(\boldsymbol{X_*}))) = \boldsymbol{K_{X_*X_*}} - \boldsymbol{K_{X_*X}} \left(\boldsymbol{K_{XX}} + \boldsymbol{\Sigma}\right)^{-1} \boldsymbol{K_{XX_*}} \tag{6.22}$$

Posterior

Here, the elements $\boldsymbol{K_{XX}}$, $\boldsymbol{K_{X_*X}}$ and $\boldsymbol{K_{X_*X_*}}$ are block covariances derived from equations 6.5. The joint-covariance matrix depends on several hyper-parameters $\boldsymbol{\theta}$. The MTGP prior has a greater number of hyper-parameters when compared to single output GP. This is due to the need to define hyper-parameters for the coregionalization matrix. We maximize the log-marginal likelihood to find a set of good hyper-parameters. This leads to an optimization problem where the objective function is given by equation 6.23

$$\log(\Pr[\boldsymbol{y_{joint}} \mid \boldsymbol{X}, \boldsymbol{\theta}]) = \log[GP(\boldsymbol{y_{joint}}|0, \boldsymbol{K_{XX}} + \boldsymbol{\Sigma})] \tag{6.23}$$

Scalability?

Calculating the posterior distribution and fine-tuning the hyper-parameters for an MTGP is similar to a Single Output GP (chapter 2). The only thing different is the assumption that the output can be expressed as an extra dimension, resulting in a change of structure of covariance matrix and increasing the number of hyper-parameters. By writing a joint-prior for multiple outputs we have effectively increased the size of the Gram matrix. This further increases the burden on the scalability of MTGP. We will discuss the scalability solutions to MTGP in chapter 7.

6.3 Experiments

Let us revisit the data-set $\mathcal{D}_2$ used to calculate the posterior in section 2.3 and 4.4.3, but here we will update the data-set to have two different outputs, let us call this data-set $\mathcal{D}_4$. We compare the accuracy of prediction on the data-set ($\mathcal{D}_4$), for independent kernels (chapter 2), simple multi-task kernels (section 6.2.2), and multi-fidelity kernels (section 6.2.4).

$$f^1(x) = \frac{sin(5\pi x)}{5\pi x} \tag{6.24}$$

$$f^2(x) = f^1(x)/2 + 0.1cos(f^1(x)) \tag{6.25}$$

The Matlab code 6.1 is how we have created the data-set $\mathcal{D}_4$ for our upcoming experiment.

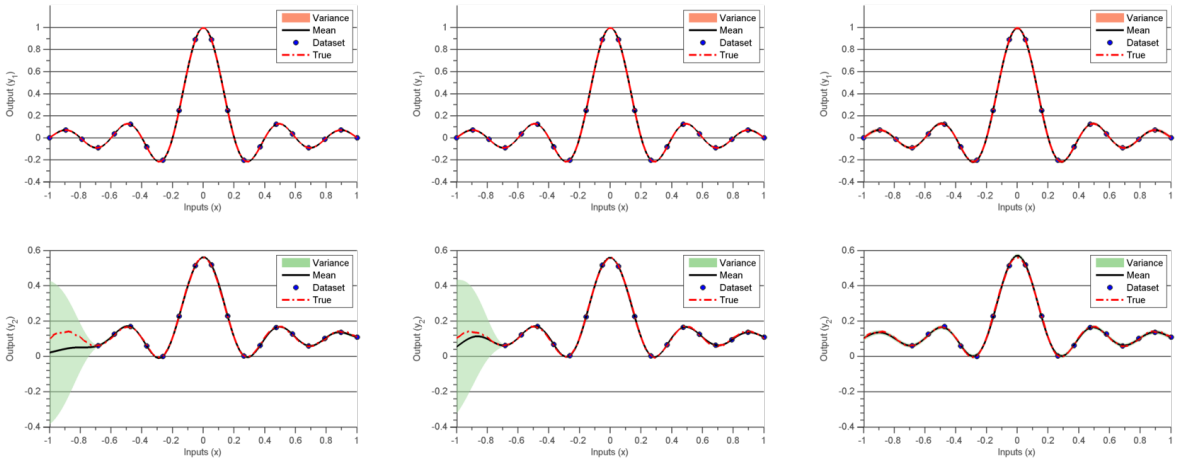
```
nData = 20; % number of data-points

f1 = @(x)sin(5*pi*x)./(5*pi*x); % Function
f2 = @(x) f1(x)/2 + 0.1*cos(f1(x));

noise = 0.1; % Noise in data-set
xData = linspace(-1, 1, nData)';
% full set of input points
xFull = [xData, ones(nData, 1); xData, 2*ones(nData, 1)];
% full set of output points
yFull = [f1(xData); f2(xData)];
```

Matlab Code 6.1: Code for data-set D4

To build the models we follow the standard work-flow of GP regression; we first define a zero mean prior using a preferred covariance function and define a hypothesis space, we then calculate the posterior distribution conditioned on data-set $\mathcal{D}_4$, and finally optimize the marginal likelihood to compare the final predictions of the three covariance functions.

Figure 6.4

Figure 6.4 shows the posterior distribution using the three covariance functions. The solid black line defines the mean function, while the shaded region defines 95% confidence interval (2σ) distance away from the mean. The output y^1 and y^2 are evaluated at $N_j = 20$ equidistant points, while the points in between $x^2 = [-1, -0.75]$ are removed from the second output data-set. Figure 6.4(a) shows the posterior when the two outputs are assumed to be independent of each other. Figure 6.4(b) shows the posterior when the two outputs are related through a multi-task covariance function, while the figure 6.4(c) shows the posterior when the outputs are related through a multi-fidelity covariance kernel. The mean prediction for the multi-fidelity covariance is best, followed by the simple multi-task kernel and then by the independent GP.



(a) GP posterior for **Independent outputs** conditioned on the data $\mathcal{D}_4$, we choose an SE kernel as the individual covariances of the individual outputs. The points where data is unavailable has high value of variance and bad mean prediction.

(b) GP posterior for a **simple multi-task** kernel conditioned on the data $\mathcal{D}_4$, we choose an SE kernel as the covariance across input points ($k_{inputs} = k_{SE}$). The points where data is unavailable has high value of variance but a better mean prediction when compared to figure 6.4(a)

(c) GP posterior for a **multi-fidelity** kernel conditioned on the data $\mathcal{D}_4$. We choose a SE kernel as the covariance across input points ($k_{inputs} = k_{SE}$) and covariance for δ process, the value of $r(x)$ is kept constant. We can observe that, the prediction at missing points is more accurate.

Figure 6.4: Joint Posterior distribution for two outputs with missing data at $x^2 = [-1 : -0.75]$. The solid black line defines the mean function, shaded region defines 95% confidence interval (2σ) distance away from the mean.

We now compare the accuracy of the three methods for an increasing number of input points. To measure the accuracy of the prediction, we again use the 10-fold Cross Validation (CV).

Figure 6.5 shows the accuracy of the three methods with an increasing number of data-points. The box-plots in red are cases when outputs are assumed independent, the box-plots in green are cases when covariance is assumed to be multi-task, and the box-plots in blue are cases when the output covariance is assumed to be multi-fidelity. As expected the error improves with the increasing number of data-points, the multi-fidelity covariance is best for all the three cases.

Linearly separable

The independence assumption cannot learn the relationship between the outputs hence has the highest value of error. The multi-task covariance has better accuracy than the independent case, although it still cannot match the performance of the multi-fidelity model. Multi-fidelity covariance model works so efficiently because the formula of the second output. The second output is a combination of a multiplication term to the first

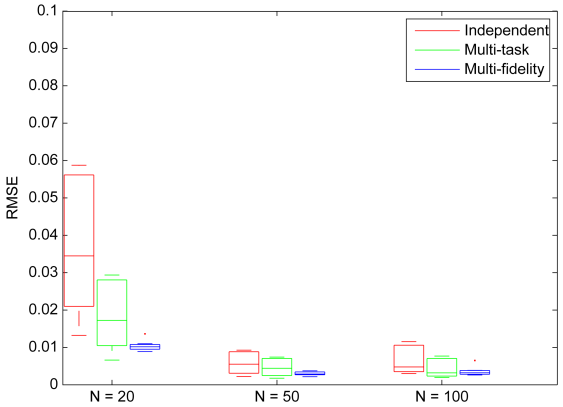


Figure 6.5: 10-fold RMSE box-plots for increasing number of input points. The box-plots in red are cases when outputs are assumed independent, the box-plots in green are cases when covariance is assumed to be multi-task, and the box-plots in blue are cases when the output covariance is assumed to be multi-fidelity. The error improves with increasing data-points,the multi-fidelity covariance is best fir for all the three cases.

output $f^2/2$, and a new term $0.1cos(f^1(x))$. This exactly matches the linearly separable assumptions in the multi-fidelity covariance function, on the other hand the multi-task model cannot effectively capture the new term $0.1cos(f^1(x))$.

6.4 Extrapolation of Flight Loads

Certification

Once an aircraft design is ready and a flight-test aircraft is manufactured, it becomes the task of the aircraft manufacturer to certify the airplane. This section is dedicated to using GP algorithms to assist in the task of flight-loads certification. For certification of flight-loads, it is necessary to show that the loads (aerodynamic forces) experienced by the aircraft are coherent with the loads predicted during the design phase. To achieve this we perform flight tests with varying aircraft parameters and get a proper understanding of how the loads are varying with respect to various parameters.

Figure 6.6

Figure 6.6 is an example of the prediction of Shear Load (Tz) on a section of the wing, for varying values of Mach and Load factor (Nz). The points in blue are the points where tests have been performed, the black line denotes the convex hull of the flight test points. The 3d surface is the prediction of a GP model built by using measurements on the input points and an SE kernel. The boundaries denoted by the red lines are the limit load cases, flying near the limit conditions will result in irreversible damage to costly flight-test aircraft. The data is normalized according to Airbus standards.

Verification and validation of the engineering model using experimental data of flight-test is a difficult exercise because:

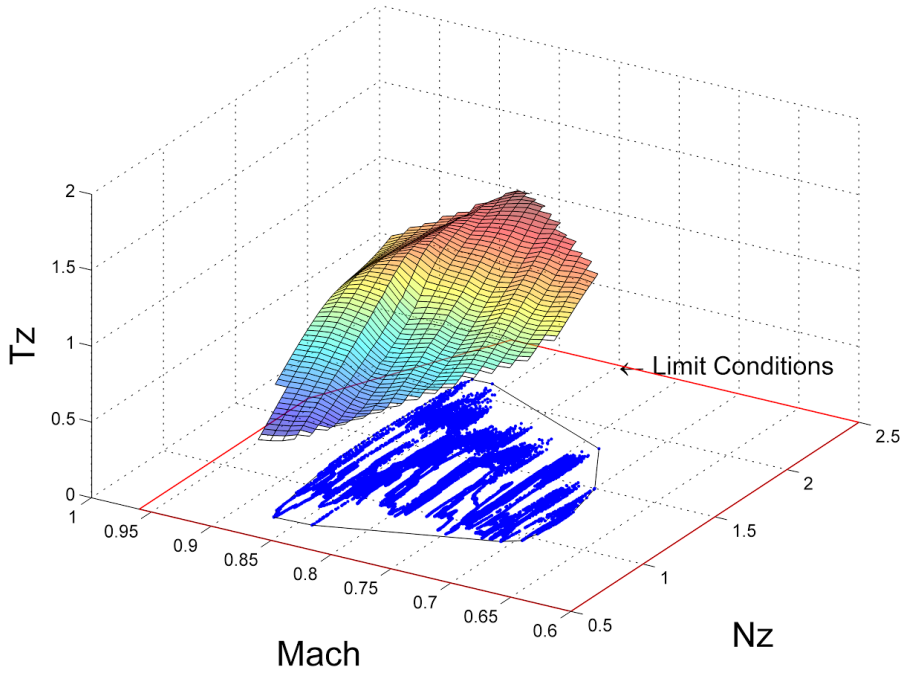


Figure 6.6: Flight-loads identification

1. Each flight-test campaign costs millions of euros, and hence flying the aircraft in all of the parameter combinations is very costly.

2. We cannot fly the aircraft at limit conditions because it will cause irreversible degradation to our single test-aircraft. But we still need to prove that the loads experienced at these limit conditions fall in the safety limit as prescribed by the certification authorities.

Hence we need to both interpolate the experimental results in the convex hull of flight-test points, while also being able to extrapolate until the red line for certifying the safety limits. In this section, we propose a method to perform extrapolation using the engineering model developed during the detailed design phase. We would like to extend the MTGP formulation for extrapolation cases.

6.4.1 Current Approach

Theoretical loads model

During the design phase of the aircraft, the flight loads department comes up with a theoretical model as shown in figure 6.7. The aerodynamic department starts from a set of inputs comprising of aircraft states (for example angle of attack, Mach etc) and aircraft geometry (for example deformation of control surfaces), and runs CFD simulations on these set of inputs. Since we cannot run CFD simulations on each and every set of parameters in

the flight domain, another aerodynamics team tries to build a parametric surrogate model which can map any set of aircraft states into rigid aerodynamic loads. This surrogate model is built by combining several types of aerodynamic computations, wind tunnel tests, and results in a lookup table model consisting of 100 aerodynamic parameters. In parallel the inertial loads department, uses the structural model to calculate the interial loads for same aircraft states. Finally, these two load models are combined to give the final load predictions.

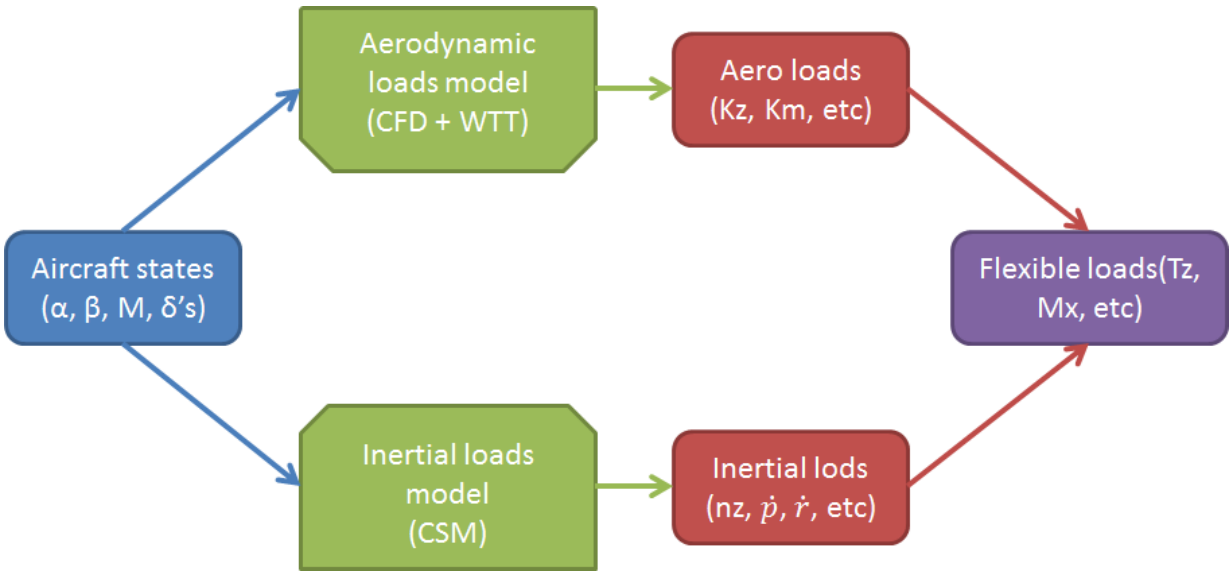


Figure 6.7: Current Flight-Loads theoretical model

Complexity

To update the simulation model, the measured data from flight-test is compared to the simulation predictions. The differences between the two models are minimized by minimizing the least square error. There are around 100 ($D_{outputs}$) load outputs (Tz, Mx, Kz etc), which are dependent on around 30 (D_{input}) flight parameters (Mach, the angle of attack etc). The parametric model has 100 variables and the flight tests are performed at millions of data-points. Updating this massively complex model is a highly complicated problem, requiring dozens of engineers and 6 months of certification time. Moreover, there are two main problems with this approach:

1. Due to the parametric nature of the surrogate model, the updated model performs perfectly in certain areas of the flight domain while giving bad results in other parts.

2. Due to the above issue many a times while updating the theoretical model, we add and remove certain parameters. CHoosing which parameters to update and which parameters to add, requires a team of 5-10 engineers working for 6 months to finally deliver the updated model.

6.4.2 Proposed Approach

Non-parametric

The main issue while using a parametric model is that, it uses parameters to express a function between input and outputs. This means to be able to express non-representable functions we iteratively add parameters in the model. We wish to replace this model updating phase with a MTGP model. We have access to both a relatively cheap simulation model produced during the design phase and costly flight-test experiments. Using the multi-fidelity formulation described in section 6.2 we can encode the simulation model as a prior information. This helps in providing the correct bias and using the years of hard work encoded into the simulation model as a prior information for learning the flight-test model.

The figure, 6.4.4 shows a common example of the difference between a simulated model and an experimental model. It shows predictions of a simulated model of the coefficient of lift, with respect to the angle of attack (in green). The points in red denote the measurements from flight-test.

6.4.3 Proposed method

There are currently three methods to perform extrapolation:

1. **Build a flight-test surrogate model**: We can build a GP model purely using data from flight-test. Unfortunately, the simple stationary covariance functions do not have any extrapolation capabilities (chapter 4). This means we would have to either use a Spectral Mixture to automatically detect patterns [Wilson 2014] or iteratively adjust the covariance function to detect patterns [Duvenaud 2014]. Unfortunately, flight-test data is very costly and thus we might not have access to a huge amount of data to learn detailed patterns. Moreover, by not using the information of a simulation model we are effectively putting years of hard-work into waste.

2. **Update the simulation model**: This is the standard methodology used by design team during the certification process. The results of flight-test are compared to the results of prediction of a simulation model. A simulation model has several parameters, which are fine-tuned to decrease the difference between flight-test data and simulation results. This is a costly and time-consuming process, requiring a team of several dozen engineers working on a tight certification schedule.

3. **Use multi-fidelity modelling**: The final option is using the multi-fidelity approach proposed in section 6.2.4. If we assume the simulation model as a low-fidelity model, and flight-test data as results from a high-fidelity model, we can use launch the simulations at limit conditions and use the multi-fidelity formulation to perform predictions. This is equivalent to encoding the simulation model as the prior information.

Suppose we have a D_{inputs}-dimensional input space, and two outputs (f^s, y^e) which represent the simulation model and flight-test data respectively. The experiments are performed at the input points x^e for $x^e \in \mathbb{R}^{N_e \times D_{inputs}}$, while the simulation model can be evaluated exactly at all points in the input domain.

$$\Pr[f^s(x)] = GP(\bar{f^s}(x), k^s = 0) \tag{6.26}$$

Here, $\bar{f^s}(x)$ signifies evaluation of the simulation model at input point x, since the simulation model is deterministic the covariance term (k^s) is to zero. We can also signify the experimental observations as:

$$y^e(x) = f^e(x) + \epsilon_{n_e} \tag{6.27}$$

Here, $f^e(x)$ is the latent physical process of the flight-test, while ϵ_{n_e} is experimental error. For the current case let's assume the experimental error is sampled from a white noise Gaussian , $\epsilon_{n_e} \sim \mathcal{N}(0, \sigma^2_{n_e})$. We can therefore, write the joint prior between the experimental measurement (y^e) and simulation model (f^s) as:

$$\Pr\begin{bmatrix} f^s \\ y^e \end{bmatrix} = GP\left(\begin{bmatrix} \bar{f^s}(x) \\ r(x^e) * \bar{f^s}(x) \end{bmatrix}, \begin{bmatrix} k^s(x^s, x^s) & r(x^s)k^s(x^s, x^e) \\ r(x^e)k^s(x^e, x^s) & r(x^e)^2 k^s(x^e, x^e) + k^\delta(x^e, x^e) + \sigma^2_{n_e} \times I_{N_e} \end{bmatrix}\right) \tag{6.28}$$

Since, the simulation model is deterministic, the covariance of the simulation model is zero ($k^s = 0$).

$$\Pr\begin{bmatrix} f^s \\ y^e \end{bmatrix} = GP\left(\begin{bmatrix} \bar{f^s}(x) \\ r(x^e) * \bar{f^s}(x) \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0 & k^\delta(x^e, x^e) + \sigma^2_{n_e} \times \mathbb{I}_{N_e} \end{bmatrix}\right) \tag{6.29}$$

Since the cross-covariance between simulation model and experimental data is zero, the two GPs are independent. We can thus write the GP prior for only the experimental data as equation 6.30.

$$\Pr[y^e] = GP\left(r(x^e) * \bar{f^s}(x), \quad k^\delta(x^e, x^e) + \sigma^2_{n_e} \times \mathbb{I}_{N_e}\right) \tag{6.30}$$

The above form of the GP prior means that, the modified form of the simulation model can be treated as the mean of the GP prior, while the k^δ learns the difference between the two models. This result is exactly similar to recursive multi-fidelity model [Gratiet 2012] since we have broken the joint GP formulation into smaller parts.

Latency

Often times when we are performing measurements we may have latency or translation among the measurements and predictions. This latency can be taken into account by introducing a delay hyper-parameter (Δx) [Osborne 2008]. This means that our proposed prior can be written as:

$$\Pr[y^e] = GP\left(r(x^e - \Delta x) * f^s(x \bar{-} \Delta x), \quad k^\delta(x^e, x^e) + \sigma_{n_e}^2 \times \mathbb{I}_{N_e}\right) \tag{6.31}$$

This is equivalent to assuming a Markov property as equation 6.32, $\forall x' \neq x - \Delta x$.

$$Cov(y^i(x), y^{i-1}(x') \mid y^{i-1}(x - \Delta x)) = 0 \tag{6.32}$$

If again we use a simple covariance function for k^δ then extrapolation of the covariance term will again tend to zero. This means that we still need to detect the pattern of the difference between experimental data (y^e), and simulation model ($r(x^e - \Delta x) * f^s(x \bar{-} \Delta x)$), but we have significantly reduced its need by applying the prior information of the simulation model.

6.4.4 Experiments on toy data-set

We now present a comparison of prediction accuracy between independent learning, multi-fidelity learning and proposed model in section 6.2.4. We have used a elsA solver to simulate a NACA 0012 airfoil, the model uses SST turbulence model and the Coefficient of lift (Cl) values are measured at steady points. The points in green in figure 6.4.4 represent the results of the simulation for varying angle of attack, all the other variables (Mach, Reynolds number etc) are kept constant for this exercise.

To represent the experimental data-set we make the following changes, equation 6.33 represents the case when there is no difference between experiment and simulation, equation 6.34 represents the case when there is a measurement noise, equation 6.35 represents the case when the simulation has not captured a sinusoidal component in the measurements, and equation 6.36 represents the case when there is a translation in the measurement.

$$y^e(x) = f^s(x) \tag{6.33}$$
$$y^e(x) = f^s(x) + \epsilon_n \tag{6.34}$$
$$y^e(x) = f^s(x) + sin(f^s) + \epsilon_n \tag{6.35}$$
$$y^e(x) = f^s(x + 0.2) + \epsilon_n \tag{6.36}$$

Here ϵ_n represents an independent noise, sampled from a white noise Gaussian $\epsilon_n \sim$

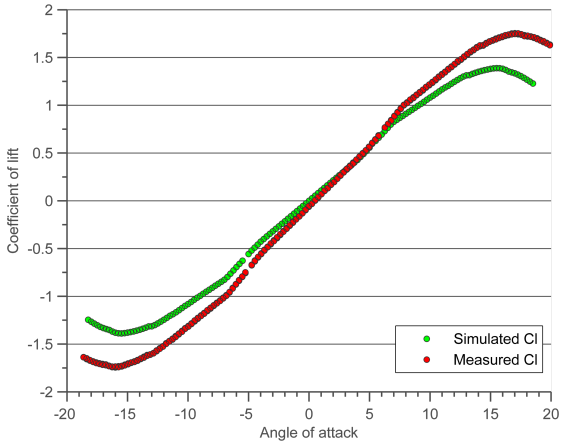


Figure 6.8: Example of difference between simulated Coefficient of lift (f^s) and experimental Coefficient of lift (y^e) (coming from equation 6.36), with respect to angle of attack.

$\mathcal{N}(0, 0.001)$. To measure the accuracy of predictions, we again launch a 10-Fold CV and calculate the RMSE. The 10-Fold CV has been performed for all the four types of changes between the measurement and experimental error. The hyper-parameters are fine-tuned for each of the three models (independent, multi-task and proposed model) by maximizing the marginal likelihood.

10-Fold CV

Figure 6.9 shows the box-plots of RMSE's for all the four cases. We see that the independent learning method always performs worst when compared to multi-task learning or proposed model with translations. When error starts appearing in the model then proposed model starts outperforming the multi-task method (there is no hyper-parameter for noise in the multi-task model). When there is a translation component present in the measurement then the proposed model clearly outperforms the two models.

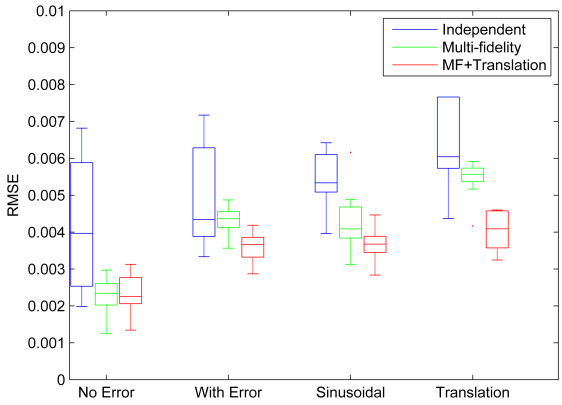


Figure 6.9: Boxplots of extrapolation for the four different cases; experimental set with no error (equations 6.33), with a white noise error (equations 6.35), with a sinusoidal error (equations 6.35) and with translation (equations 6.36).

The main aim of the current experiment is to present how the simulation model can be used for extrapolating experimental measurements. In our case, it is done by adding a translation hyper-parameter (Δx) to compensate for the lag in measurement and an error covariance to model the measurement error. We would also like to point out that the proposed model works better only because there is a translation term in the third case (equation 6.36).

Non-separable

Often times the different models cannot be linearly separable by the Markov property. This leaves us with three choices, we can try and identify the pattern in experimental data by using pattern detecting covariance functions [Wilson 2014]. Second, we can use a non-separable MTGP joint-covariance to learn the relation between the simulation model and the experimental model [Alvarez 2011], or third, we can learn the pattern of the k^{δ} automatically [Duvenaud 2013]. We wish to study the prediction properties of all the three proposed approaches and try to understand which is the better choice for tasks of model updating and extrapolation in the future.

6.5 Summary and discussion

In the current chapter, we have seen how to perform GP regression in the presence of multiple outputs. One simple method of learning a regression model for multiple outputs is by assuming the output number as an extra dimension (this extra dimension is a categorical variable). This observation is beneficial due to two reasons, first, we have effectively increased the number of data-points (figure 6.2.1). Second, all the tricks of making covariance functions for multiple dimensions can be applied to the case of multiple outputs as well (section 6.2.1).

If no information is available about the relation between outputs then we can learn the relationship between outputs by either using a simple MT covariance (section 6.2.2), a 'Linear Model of Coregionalization' (section 6.2.3), or a convoluted GP [Alvarez 2011]. If we know *a-priori* that the outputs are coming from models of multiple fidelity then we can use the joint covariance function proposed by [Kennedy 2000] (section 6.2.4).

In section 6.4 we have shown how multi-fidelity GP models can be used to perform interpolation or extrapolation of experimental data. We include an error term and a translation term into a normal multi-fidelity model to account for differences in the simulation model and experimental data. In the future, we would wish to quantify the performance of different methods of extrapolation and decide which method to choose in a particular scenario.

Chapter 7

Adding relationships in GP Regression

Résumé

Dans ce chapitre, nous considérons le problème de la régression par GPs à plusieurs sorties corrélées par les lois physiques d'un système. Ces lois physiques peuvent être représentées sous la forme d'un ensemble d'équations ou d'un code informatique (CFD ou CSM par exemple).

Un exemple usuel d'application des relations *a-priori* connues entre les sorties est appelée le 'Gradient Enhanced Kriging' dans la littérature sur le Krigeage [Chung 2002, Morris 1993, Forrester 2009] ou appelée 'GP regression with derivative observations' dans la littérature sur les GPs [Solak 2003]. La relation "dérivée" est appliquée en modifiant la fonction de covariance jointe (équation 6.5). Dans ce chapitre, nous étendons le modèle du 'Gradient Enhanced Kriging' aux intégrales, aux fonctions quadratiques ou à n'importe quelle fonctionnelle entre les sorties [Constantinescu 2013].

Le codage des lois physiques rend le modèle appris compatible avec la physique du système, ce qui constitue un avantage majeur par rapport aux modèles purement générés par les données [Jazwinski 2007]. La contribution de ce chapitre est double ; nous démontrons d'abord l'efficacité de l'ajout de relations dans un GP, à la fois sur un base des données synthétiques et sur un ensemble de données normalisées d'essais en vol. Les résultats ont été présentés lors d'une conférence AIAA en 2016 [Chiplunkar 2016a]. Après, nous démontrons comment appliquer la régression GPs avec plusieurs sorties à une grande quantité de données, en utilisant à la fois une inférence variationnelle et une 'Distributed GPs'. Les résultats de cette étude ont été publiés dans ICPRAM 2016 et LNCS 2017 [Chiplunkar 2016c, Chiplunkar 2017c].

7.1 Introduction

In this chapter, we consider the problem of modelling multiple output Gaussian Process (GP) regression correlated through physical laws of the system. Some example of physical laws can be 'Newtons Laws of Motion', 'Navier Stokes equation' or the 'Heat dissipation equation'. These physical laws have been developed after centuries of observations and iterative experiments. They can be represented in the form of a set of equations or through a computer code (CFD, CSM codes). This chapter discusses how to encode such relationships between outputs in an MTGP regression framework.

Contribution

Encoding physical laws makes the learned model consistent with the physics of the system, which is a major advantage when compared against pure data generated models [Jazwinski 2007]. The contribution of this chapter is two-fold, we first demonstrate the effectiveness of adding relationships in GP, on both a toy data-set and a normalized flight-test data-set. The results were presented in AIAA aviation conference 2016 [Chiplunkar 2016a]. Second, we demonstrate how to scale MTGP to a large amount of data, using both Variational inference and Distributed GPs. The results of this study were published in ICPRAM 2016 and LNCS 2017 [Chiplunkar 2016c, Chiplunkar 2017c].

Enhanced kriging

One popular example of enforcing the prior known relations between outputs is the "Gradient Enhanced Kriging" in Kriging literature [Chung 2002, Morris 1993, Forrester 2009] or "GP regression with derivative observations" in GP literature [Solak 2003]. The derivative relationship is enforced by changing the joint-covariance function (equation 6.5). In this chapter, we extend the framework of gradient enhanced kriging to integral enhanced kriging, quadratic enhanced kriging or any functional relationship between outputs [Constantinescu 2013].

The current chapter unfolds as follows. Section 7.2.1 describes how to enforce linear relationships between outputs, while section 7.2.2 describes how to encode non-linear relationships. In section 7.4 various methods of scaling multi-output GP are demonstrated. Finally, in section 7.5 we demonstrate the approach on both theoretical and flight-test data. This chapter has been motivated by the works of [Constantinescu 2013, Alvarez 2009, Särkkä 2011, Jidling 2017, Ginsbourger 2013, Särkkä 2011].

7.2 Encoding relationships in GP

For simplicity let us take the case of an explicit relationship between two outputs y^1 and y^2. Suppose we measure the two outputs with some error (ϵ_{n1} and ϵ_{n2}), while the true physical process is defined by latent variables (f^1 and f^2). Then the relation between the output function, measurement error, and true physical process can be written as follows[1].

[1]Reminder the superscript denotes the number of output and does not denote the power

$$y^1 = f^1 + \epsilon_{n1} \tag{7.1}$$
$$y^2 = f^2 + \epsilon_{n2} \tag{7.2}$$

Here, ϵ_{n1} and ϵ_{n2} are measurement errors sampled from a white noise Gaussian $\mathcal{N}(0, \sigma_{n1}^2)$ and $\mathcal{N}(0, \sigma_{n2}^2)$ respectively. While, the relation between the latent function can be expressed as follows:

$$f^1(z) = \mathcal{L}\left(f^2(x), z\right) \tag{7.3}$$

Here $\mathcal{L}(.) \in \mathcal{C}^2$ is an operator defining the relation between f^1 and f^2. We are thus, interested in evaluating the Gram matrix $\boldsymbol{K_{XX}}$ between the full data-set.

$$\boldsymbol{K_{XX}} = \begin{pmatrix} Cov(f^1(x), f^1(x)) & Cov(f^1(x), f^2(x)) \\ Cov(f^2(x), f^1(x)) & Cov(f^2(x), f^2(x)) \end{pmatrix} \tag{7.4}$$
$$= \begin{pmatrix} \boldsymbol{K_{11}} & \boldsymbol{K_{12}} \\ \boldsymbol{K_{21}} & \boldsymbol{K_{22}} \end{pmatrix} \tag{7.5}$$

We will denote the Gram matrices for auto-covariances as $\boldsymbol{K_{ii}}$ and cross-covariances as $\boldsymbol{K_{ij}}$ for $i \neq j$. In next two sections, we assume one output function to be an independent (f^2) random variable and evaluate the remaining auto-covariance ($Cov(f^i(x), f^i(x))$) and cross-covariance ($Cov(f^i(x), f^j(x))$ for $i \neq j$) functions exactly, if the physical relation between them is linear or use an approximate joint-covariance, if the relationship between them is non-linear[Constantinescu 2013]. $\boldsymbol{K_{XX}}$

7.2.1 Case of Linear Operators

Physical laws of the system are generally denoted by sets of differential equations which explain the basic behaviour of the system, these differential equations are linear in nature. One simple example of a linear operator is an integration operation, $f^1(z) = \int_\infty^z f^2(x)dx$.

The physical laws can be enforced by manipulating the covariance function between outputs ($Cov(f^i(x), f^j(x))$). [Alvarez 2009, Särkkä 2011] have used these forms of covariance functions to solve partial differential equations, [Jidling 2017] have used these covariances to impose linearity constraints in observations of magnetic experiments, while [Ginsbourger 2013] have used them to encode transformations in a random process. We give an overview of their work in this section.

If $\mathcal{L}(.)$ is a linear operator then it has the following property:

$$\mathcal{L}(a_1 g^1 + a_2 g^2) = a_1 \mathcal{L}(g^1) + a_2 \mathcal{L}(g^2) \tag{7.6}$$

Here, a_1 and a_2 are scalars and g^1 and g^2 are functions. If we assume a GP prior over the function f^2 such that its mean (m^2) and covariance functions (k^2) can be represented as the set of equations below. Note that m^2 is the mean of the output number 2 and not the square of its mean.

$$E[f^2(x)] = m^2(x) \tag{7.7}$$
$$Cov[f^2(x), f^2(x')] = k^2(x, x') \tag{7.8}$$

Then after applying the operator $\mathcal{L}(.)$, the mean of the function f^1 can be written as equation 7.9. From now on we will use a short-hand notation for operation $\mathcal{L}\left(f^2(x), z\right)$ as $\mathcal{L}\left(f^2, z\right)$.

$$\begin{aligned} E[f^1(z)] &= E[\mathcal{L}\left(f^2, z\right)] \\ &= \mathcal{L}(E[f^2], z) \\ &= \mathcal{L}(m^2, z) \end{aligned} \tag{7.9}$$

Similarly, the covariance of the function (f) can be written as $Cov[f^1(z), f^1(z')]$ (equation 7.10).

$$\begin{aligned} Cov[f^1(z), f^1(z')] &= E[[f^1(z) - \mathcal{L}(m^2, z)][f^1(z') - \mathcal{L}(m^2, z')] \\ &= E[[\mathcal{L}\left(f^2, z\right) - \mathcal{L}(m^2, z)][\mathcal{L}\left(f^2, z'\right) - \mathcal{L}(m^2, z')] \\ &= E[[\mathcal{L}\left(f^2 - m^2, z\right)][\mathcal{L}\left(f^2 - m^2, z'\right)]] \\ &= \mathcal{L}\left(\mathcal{L}\left(E[[f^2 - m^2][f^2 - m^2]], z'\right), z\right) \\ &= \mathcal{L}\left(\mathcal{L}\left(k^2(x, x'), z'\right), z\right) \end{aligned} \tag{7.10}$$

Since $\mathcal{L}$ is a linear operator, the resulting function is a positive semidefinite function [Ginsbourger 2013, Särkkä 2011]. We can thus write the joint-Gram matrix between the outputs f^1 and f^2 as:

$$\boldsymbol{K_{XX}} = \begin{pmatrix} \mathcal{L}(\mathcal{L}(\boldsymbol{K^2}, x^1), x^1) & \mathcal{L}(\boldsymbol{K^2}, x^1) \\ \mathcal{L}(\boldsymbol{K^2}, x^2) & \boldsymbol{K^2} \end{pmatrix} \tag{7.11}$$

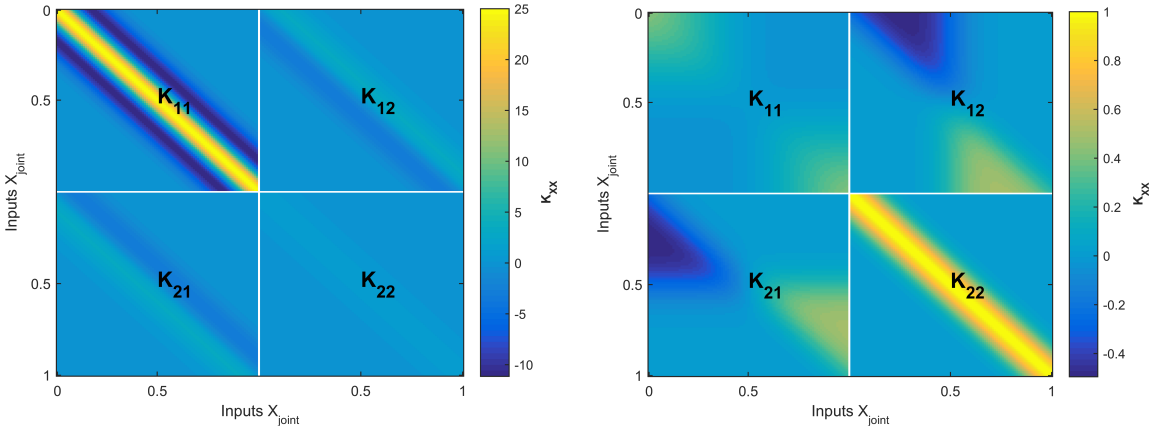
Here the matrix $\boldsymbol{K^2}$ is the Gram matrix evaluated using the covariance function $k^2(x, x')$. The hyper-parameters of the prior are $\theta = \{\theta_{amplitude}, \theta_{lengthScale}\}$. These correspond to the hyper-parameters of the independent covariance function k^2. We have effectively represented the covariance function $Cov[f^1(z), f^1(z')]$ and $Cov[f^1(z), f^2(z')]$ in terms of the covariance function $Cov[f^2(z), f^2(z')]$ using the known relation between outputs.

The sample code 7.1 shows one of the ways to calculate the covariance functions for the equation 7.11. For complex linear operators, the covariance functions start getting complicated pretty quickly, using the symbolic toolbox is a nice hack around manually calculating their functional forms.

```
function [covy1y1, covf1f2, covf2f1, covf2f2] =
   calculateJointCovarianceFunctions(linearOperator)

% calculating the covariance functions using symbolic toolbox
[x1, x2, sn22, sn11, sf2, ell] = deal([]);
syms x1 x2 sn22 sn11 sf2 ell sqDist dist

% ell = exp(hyp1)
% sf2 = exp(2*hyp2)

latentk22 = sf2.*(exp(-0.5.*(x1-x2)^2/ell^2));
covf1f2 = linearOperator(latentk22, x1);
covf2f1 = linearOperator(latentk22, x2);
latentk11 = g(covf1f2, x2);

covf2f2 = latentk22 + sn22;
covy1y1 = latentk11 + sn11;

end

%%
% Integral Linear operator
L = @(y, x)  int(y, x, 0, x);
[covy1y1, covf1f2, covf2f1, covf2f2] =
   calculateJointCovarianceFunctions(L);
```

Matlab Code 7.1: Create joint covariance functions using Matlab Symbolic Toolbox

Figure 7.1 represents the Gram matrix obtained by enforcing a relationship between two outputs y^1 and y^2. We use a SE kernel ($\theta_{amplitude} = 1$ and $\theta_{lengthScale} = 0.2$) as a covariance function for defining the prior distribution of the independent output y^2. The auto-covariance and cross-covariance functions are calculated using the code 7.1. The Gram matrix is evaluated at 100 equidistant points $X^j = [0 : 0.01 : 1]$ for both the outputs. Figure 7.1(a) is the Gram matrix when the outputs are related through a derivative relationship ($y^1 = \frac{\partial y^2}{\partial x}$), while figure 7.1(b) is the Gram matrix when the outputs are related through an integral relationship ($y^1 = \int_0^x y^2$). The $\boldsymbol{K_{ii}}$ represent the Gram matrices between same outputs, while the $\boldsymbol{K_{ij}}$ (for $i \neq j$) represents the Gram matrix between different outputs.



(a) Gram matrix between y^1 and y^2 such that $y^1 = \frac{\partial y^2}{\partial x}$. SE covariance function is used for the y^2 with $\theta = [1, 0.2]$, and $\boldsymbol{X}^j = [0 : 0.01 : 1]\ \forall\ j = [1, 2]$

(b) Gram matrix between y^1 and y^2 such that $y^1 = \int_0^x y^2$. SE covariance function is used for the y^2 with $\theta = [1, 0.2]$, and $\boldsymbol{X}^j = [0 : 0.01, 1]\ \forall\ j = [1, 2]$

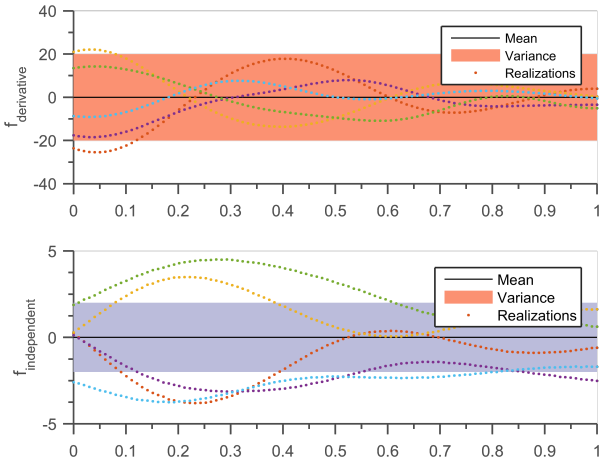
Figure 7.1: Gram matrices for the joint kernel, which encodes the relation between two outputs y^1 and y^2

The joint Gram matrix between f^1 and f^2, means that a random draw of independent function f^2 will result in a correlated draw of the function f^1. This effectively means that when we draw a random function f^2 it will result in a correlated draw of f^1, such that the two random functions will satisfy the enforced equation.

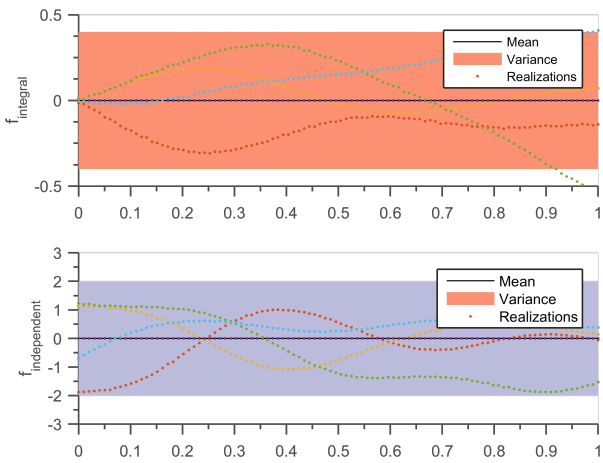
Figure 7.2

Figure 7.2 shows randomly drawn functions for two outputs related through an equation. The figure 7.2(a) shows random draws coming from a differential relationship between $f_{differential}$ (red) and $f_{independent}$ (blue) such that $f_{differential} = \frac{df_{independent}}{dx}$. We can see that the top figure is derivative of the bottom one since, $f_{derivative}$ goes to zero where $f_{independent}$ goes to maxima or minima. Similarly, the figure 7.2(b) shows random draws coming from an integral relationship between $f_{integral}$ (red) and $f_{independent}$ (blue) such that $f_{integral} = \int_0^x f_{independent}$. We can see that the top figure is integral of the bottom one since, $f_{independent}$ goes to zero where $f_{integral}$ goes to maxima or minima.

Note that, proper care should be taken before performing these operations, if an operation is not defined for the prior family of functions of f^2, then we cannot write its corresponding joint-GP. For example, if we define a GP prior for f^2 with an exponential covariance function (section 4.4.2), we cannot enforce a differential relationship on f^2. This is because the exponential covariance function defines a hypothesis space of non-differentiable functions, and hence a differential operator is not-defined for such functions.



(a) Joint draws between $f_{differential}$ (red) and $f_{independent}$ (blue) such that $f_{differential} = \frac{df_{independent}}{dx}$. We define a GP prior with zero mean and SE covariance function ($\theta = [1, 2]$) for the function f^2.



(b) Joint draws between $f_{integral}$ (red) and $f_{independent}$ (blue) such that $f_{integral} = \int_0^x f_{independent}$. We define a GP prior with zero mean and SE covariance function ($\theta = [1, 2]$) for the function f^2.

Figure 7.2: Joint draws defined by priors on related-outputs. 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.

7.2.2 Case of non-linear operators

Simulation codes

Physical laws often do not come in form of non-linear operators, but they are difficult to solve analytically[2] for complex geometries. For example solving the structural dynamics equation is difficult for continuous complex geometries of an aircraft. One popular method to solve these differential equations is to discretize the complex geometry into simple meshes. A complex geometry of an aircraft is broken down into a simpler point, beam and plate structures, these simpler geometries are easier to solve analytically. We can then apply the physical laws on these simplistic geometries, and finally, aggregate the result of individual meshes. For example, CFD and CSM are discretized techniques to evaluate fluid and structural laws on complex geometries. These discretized operations are often non-linear and are represented in form of computer code. In this section, we wish to evaluate auto and cross-covariances across outputs related through non-linear relationships.

A non-linear operation on a GP does not result in a GP, hence for the case of non-linear $\mathcal{L}(.)$ the joint Gram matrix, as derived in equation 7.4, is not positive semi-definite [Stein 1999]. Therefore, we will use an approximate joint-covariance as developed by

[2]There is a millennium prize for solving the Navier Stokes equation

[Constantinescu 2013] for imposing non-linear relations.

$$K_{XX} = \begin{bmatrix} L\boldsymbol{K}^2 L^T & L\boldsymbol{K}^2 \\ \boldsymbol{K}^2 L^T & \boldsymbol{K}^2 \end{bmatrix} + \mathcal{O}\left(\delta_2^3\right) \tag{7.12}$$

$$L = \frac{\partial \mathcal{L}}{\partial y}\Big|_{f^2=m^2} \tag{7.13}$$

Where L is the Jacobian matrix of $\mathcal{L}(.)$ evaluated at the mean (m^2) of independent output (f^2). δ_2 is the amplitude of small variations of f^2, introduced by the Taylor series expansion of $\mathcal{L}(f^2)$ with respect to $E[f^2]$.

$\mathcal{O}\left(\delta_2^3\right)$

The above covariance function takes a parametric form that depends on the mean value process of the independent variable $(E[f^2(x)])$. Equation 7.12 is basically a Taylor series expansion for approximating related kernels. Since a Taylor series expansion is constructed from derivatives of a function, which are linear operations, the resulting approximated joint kernel is a Gaussian kernel with the non-Gaussian part $(\mathcal{O}\left(\delta_2^3\right))$ as the error. Higher-order closures can be derived with higher order derivatives of the operator $\mathcal{L}(.)$. For simplicity we will restrict ourselves to first order approximation of the auto- and cross-covariance functions leading to an error of the order $\mathcal{O}\left(\delta_2^3\right)$. A more detailed derivation can be found at appendix C.

Below is a sample code (code 7.2) for calculating the joint-covariance functions automatically. The 'jacobianOfOperator' is the jacobian of the non-linear operator $\mathcal{L}$ evaluated at the mean values of y^2.

```
function [covy1y1, covf1f2, covf2f1, covy2y2] =
   calculateNonLinearJointCovarianceFunctions(jacobianOfOperator
   )

% calculating the covariance functions using symbolic toolbox
[x1, x2, sn22, sn11, sf2, ell] = deal([]);
syms x1 x2 sn22 sn11 sf2 ell sqDist dist

latentk22 = sf2.*(exp(-0.5.*(x1-x2).^2/ell.^2));
covf1f2 = jacobianOfOperator(x1).*(latentk22);
covf2f1 = jacobianOfOperator(x2).*(latentk22);
latentk11 = jacobianOfOperator(x2).^2.*(latentk22);

% adding noise
covy2y2 = latentk22 + sn22;
covy1y1 = latentk11 + sn11;
```

```
end
```

Matlab Code 7.2: Create joint covariance functions for non-linear operators using Matlab Symbolic Toolbox

7.2.3 Calculating posterior

The posterior distribution can be calculated by conditioning the prior distribution on the observation data-set. Since a prior distribution over functions defined over the joint data-set is also a GP, we can easily derive the posterior mean, posterior covariance and marginal likelihood (the exact derivations are described in section 6.2.5). We just need to insert the appropriate Gram matrix ($\boldsymbol{K_{XX}}$) in the equations of posterior mean (equation 7.14), posterior variance (equation 7.15) and marginal likelihood (equation 7.16).

Posterior

$$E[f(\boldsymbol{X_*}))] = \boldsymbol{K_{X_*X}}\left(\boldsymbol{K_{XX}} + \boldsymbol{\Sigma}\right)^{-1}\boldsymbol{y_{joint}} \tag{7.14}$$

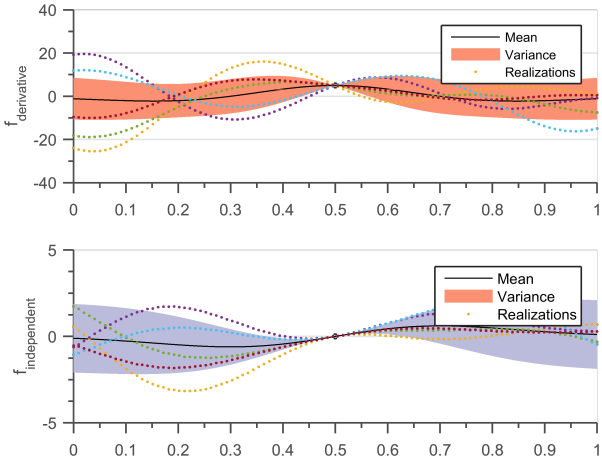
$$Cov(f(\boldsymbol{X_*})) = \boldsymbol{K_{X_*X_*}} - \boldsymbol{K_{X_*X}}\left(\boldsymbol{K_{XX}} + \boldsymbol{\Sigma}\right)^{-1}\boldsymbol{K_{XX_*}} \tag{7.15}$$

$$\log(\Pr[\boldsymbol{y_{joint}} \mid \boldsymbol{X}, \theta]) = \log[GP(\boldsymbol{y_{joint}}|0, \boldsymbol{K_{XX}} + \boldsymbol{\Sigma})] \tag{7.16}$$

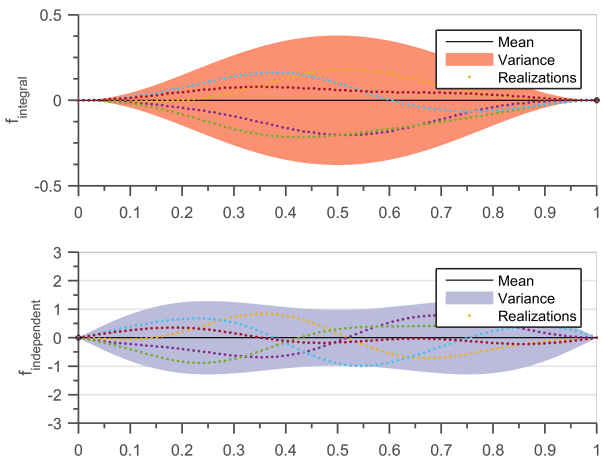
Example The figure 7.3(a) shows mean and variance for a differential relationship between independent function $f_{independent}$ (blue) and differential function $f_{derivative}$ (red), such that $f_{derivative} = \frac{\partial f_{independent}}{\partial x}$. We have conditioned the two functions such that $f_{derivative}|_{x=0} = 5$ and $f_{independent}|_{x=0} = 0$. This means that the function $f_{independent}$ passes through 0 and has a derivative equal to 5 at $x = 0$. It is easy to observe that all the functions of $f_{independent}$ that pass through $x = 0$ have the same value of derivatives. We have not maximized the marginal likelihood for this case and the hyper-parameters of k^2 are $\theta_{amplitude} = 1; \theta_{lengthScale} = 0.2$.

Figure 7.3(b) shows mean and variance for an integral relationship between independent function $f_{independent}$ (blue) and differential function $f_{integral}$ (red) and such that $f_{integral} = \int_0^x f_{independent} z.dz$. We have conditioned the two functions such that $f_{integral}|_{x=1} = 0$ and $f_{integral}|_{x=0} = 0$. This means that the function $f_{independent}$ has an integral 0 at the two points $[0, 1]$. Although it is difficult to verify from the figure, but all the randomly drawn functions of $f_{independent}$ indeed have their integral ($\int_0^1 f_{independent} = 0$) equal to zero. We have not maximized the marginal likelihood for this case and the hyper-parameters of k^2 are $\theta_{amplitude} = 1; \theta_{lengthScale} = 0.2$. All the corresponding draws from the posterior GP also follow the conditioning.

In the next section, we compare the prediction capabilities of independent GP, with that of multi-fidelity GP and joint relationship enforced GP on a synthetic data-set. We



(a) Joint draws between $f_{derivative}$ *red* and $f_{independent}$ *blue* such that $f_{derivative} = \frac{\partial f_{independent}}{\partial x}$. The posterior is conditioned such that $f_{derivative}|_{x=0} = 5$ and $f_{independent}|_{x=0} = 0$. The draws from the posterior follow the relationship between outputs and also the conditioning.



(b) Joint draws between $f_{integral}$ *red* and $f_{independent}$ *blue* such that $f_{integral} = \int f_{independent}$. The posterior is conditioned such that $f_{integral}|_{x=0} = 0$ and $f_{integral}|_{x=1} = 0$. The draws from the posterior follow the relationship between outputs and also the conditioning.

Figure 7.3: Multi-Output Gaussian Process Regression Predictions. The solid black line defines the mean function, shaded region defines 95% confidence interval (2σ) distance away from the mean. Dotted lines represent the functions randomly drawn from the posterior distributions.

then take a real flight-test data-set and study the effects of different prior k^2 on their predictions.

7.3 Experiments

In this section, we provide a numerical illustration to the theoretical derivations in the earlier sections. We start with a synthetic problem where we try to learn the model over quadratic relationships. We compare the cross-validation error values of independent GP regression with that of multi-fidelity GP and relationship enforced joint MTGP. Finally, we study the effect of choosing different priors for the independent function f^2 on a data-set for flight-loads estimation for the horizontal tail plane.

The basic toolbox used for this section is GPML provided with [Rasmussen 2005], we generate covariance functions to handle relationships as described in equations 7.4 using the “Symbolic Math Toolbox" in MATLAB 2014b (code 7.1 and 7.2).

7.3.1 Quadratic relation on Synthetic Data

We take the case of a quadratic operator $\mathcal{L}(.)$ equation 7.17.

$$f^1 = \left[f^2\right]^2 \tag{7.17}$$

Data-set

To generate the data, we randomly draw a single function of f^2 as described in equation 7.18 for 50 equally spaced inputs between [-1, 1]. The output f^1 is then calculated by applying the operation in equation 7.17. The latent functions $f's$ are then corrupted according to equation 7.18 which gives us the outputs $y's$. We finally hide the observations for y^1 in the domain $x = [-0.2, 0.2]$. We now have our data-set for testing the performance of quadratic relationship.

$$\begin{aligned} \Pr[f^2] &= GP[0, k_{SE}(0.2, 1)] \\ \Pr[\sigma_{n2}] &= \mathcal{N}[0, 0.1] \\ \Pr[\sigma_{n1}] &= \mathcal{N}[0, 1] \end{aligned} \tag{7.18}$$

$k_{SE}(1, 0.2)$ means squared exponential kernel with length scale 0.2 and variance as 1. σ_{n2} and σ_{n1} are the white noises added to the latent functions f^1 and f^2 respectively. Since the quadratic relationship $\mathcal{L}(.)$ is non-linear in nature we use equation 7.12 to calculate the auto- and cross-covariance functions as shown in equation 7.19.

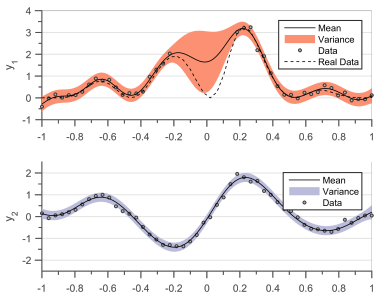
$$\begin{aligned} Cov(f^2(x), f^2(x')) &= k^2 \\ Cov(f^1(x), f^2(x')) &= 2m^2k^2 \\ Cov(f^1(x), f^1(x')) &= 4\left[m^2\right]^2 k^2 \end{aligned} \tag{7.19}$$

Here, m^2 is the mean value of function y^2. For the case of quadratic relationship the Jacobian L as described in equation 7.12 comes out to be $2{*}m^2(x)$. For non-linear operators $\mathcal{L}(.)$ the joint-covariance prior depends on the mean value of f^2 ($m^2(x)$). $Cov(f^1(x), f^1(x'))$ and $Cov(f^1(x), f^2(x'))$ are the covariance functions calculated at the input points x and x' as described in equation 7.12 and equation 6.5. We can observe that the joint-covariance has been expressed as a function of covariance k^2 using equation 7.17.

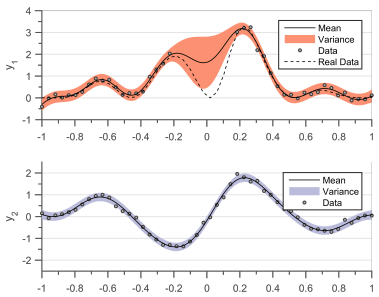
Figure 7.4

Figure 7.4(a) shows the independent fit of two GP regressions. For the case of y^1, there is a huge difference between the real data and predicted mean at points where data is unavailable. Figure 7.4(b) shows the fit using multi-fidelity GP regression model (section 6.2.4). We can observe that the multi-fidelity model cannot capture the quadratic nature of the relationship between outputs. Figure 7.4(c) shows the joint-GP regression, which gives better prediction even in the absence of data for y^1 because a transfer of information is happening from observations of y^2 present at those locations.

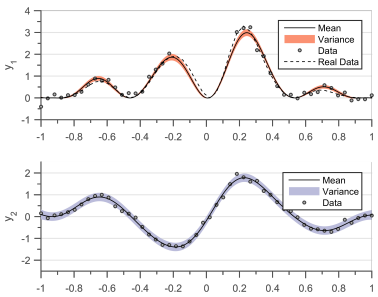
Table 7.1 shows comparison of Root Mean Square Error (RMSE). 10 sets of experiments were run for 85% of data as training set and 15% of data as the test set, test sets were



(a) Independent GP Regression for the two outputs y^1 and y^2. We can observe the huge difference between the real data and the predicted mean values at zone with no data.



(b) Multi-fidelity GP Regression for the two outputs y^1 and y^2 (section 6.2.4). We can observe the huge difference between the real data and the predicted mean values at zone with no data.



(c) Joint-GP Regression for the two outputs y^1 and y^2 related through equation 7.17. We can observe the improved prediction between zone with no data because information is being shared between the two outputs.

Figure 7.4: GP Regression on Quadratic relationship. The solid black line represents the predicted mean while the shaded area denotes 2σ uncertainty region. The dashed black line represents the real value of f^1 or f^2. For y^1 the data is hidden from section $x = [-0.2, 0.2]$.

Table 7.1: mean RMSE errors for quadratic relationship

	RMSE y^1	RMSE y^2
Independent Single-output GPR	1.74	0.14
Multi-fidelity GPR (section 6.2.4)	1.54	0.13
Joint Multi-output GPR	0.26	0.12

Table 7.1 removed uniformly from the output y^1 (similar to figure 7.4). We learn the optimal set of hyper-parameters for on training data all 10 sets of experiments. Finally, RMSE values are evaluated with respect to the test set. We are comparing here the accuracy of three model types, where the independent model gives the worst prediction, the multi-fidelity model is a little bit better but still worse than the model with the actual relationship enforced.

The current experiment demonstrates how to enforce a non-linear relationship between outputs. Multi-fidelity GP has a similar performance as that of an independent GP since the two outputs are not results of code for different fidelities. The current set of outputs do-not show the Markov property used in multi-fidelity GPs, if a similar type of information is missing in the simulation model then using the linearly separable covariance functions will obviously lead to bad extrapolation results. Enforcing the quadratic relationship between the outputs has the best accuracy of the three methods (figure 7.4(c)).

Adding inequality constraints Research on adding inequality constraints in GP models is slowly picking up pace in the GP community. [Da Veiga 2012] enforced inequality constraints by truncating probability distributions, while [Maatouk 2017] enforce inequality constraints by using functional decomposition. The quadratic relationship gives rise to an interesting property. If $f^1 = [f^2]^2$ then it means that $f^1 > 0 \; \forall x \in \mathcal{R}$, this property can also be used to indirectly enforce inequality constraints in GP models.

Suppose we have access to a N observed data-points $\mathcal{D} = \{\boldsymbol{X}, \boldsymbol{y}\}$ subject to the constraints $y_i > 0 \; \forall i \in N$. We can assume a second output $sqrtY$, such that $y = [sqrtY]^2$ and build a quadratic relationship enabled GP model between the two (y and $sqrtY$). GP models enforcing monotonocity or convexity can be similarly made by first, enforcing a derivative relationship and then a quadratic relationship.

One problem that can come up by enforcing inequality constraints using quadratic relationship, is that square root of y ($sqrtY$) can be both positive and negative. This can complicate matters when calculating the mean of the square root. It would be interesting to study the implications of choosing one sign of square roots over the another needs more investigation.

7.3.2 Flight Mechanics on Flight Test Data

In this section, we conduct experiments applying our approach on the flight loads data. We look at normalized data of a simple longitudinal maneuver. The maneuver is quasi-static which means that the airplane is in equilibrium at all times and there are no dynamic effects observed by the aircraft. The two outputs in our case are the coefficient of lift C_z on the Horizontal Tail Plane (HTP) and the spanwise lift k_z. We know that the integral of spanwise pressure will be equal to the coefficient of lift (equation 7.20).

$$C_z(\eta) = \int_{\eta_{edge}}^{\eta_{root}} k_Z(\eta) d\eta \tag{7.20}$$

.

Here, η_{edge} denotes the edge of horizontal tail span while η_{root} represents root of tail. The above equation is linear in nature and hence we will use equation 7.4 to calculate the auto- and cross-covariance functions.

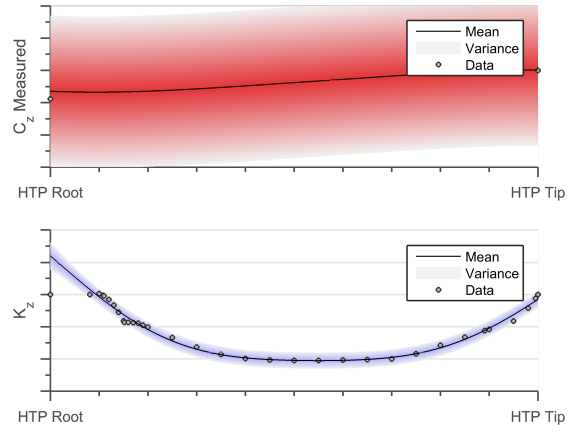
Merging domains

In practice, the coefficient of lift C_z is calculated using strain gauges on the HTP and the spanwise force is measured using pressure tappings. Chordwise pressures are measured at multiple η locations on the HTP, which are integrated chordwise to give spanwise lift distribution. We are trying to merge data coming from two different calculation chains flight mechanics (C_z) and aerodynamics (k_z), by using a basic physics based relationship between them. To the best of our knowledge this approach has not been proposed earlier in loads estimation.

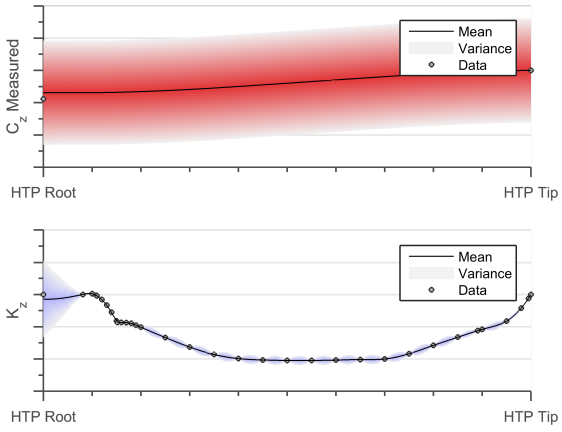
Figure 7.5

Figure 7.5 shows a comparison between two joint multi-output GP on K_z and C_z using different prior distributions. Figure 7.5(a) shows the effect of SE kernel (infinite differentiability) on the joint model. The SE kernel is a strong assumption for the type of lift distribution. To accommodate for the value of measured C_z, the error band increases and the boundary condition at HTP root is not satisfied. Moreover, we identify points in the figure 7.5(a) which fall out of the 95% confidence band. We can claim with 95% probability that if our lift distributions come from a family of infinitely differentiable functions these outliers do not satisfy the physics of the problem and can be termed as measurement errors.

Figure 7.5(b) shows the effect of twice differentiable Matérn kernel on the joint model. The twice differentiable assumption gives our family of functions enough flexibility to follow the spanwise lift data. We observe that error estimates for C_z improve considerably while there is an increased uncertainty at the HTP root for K_z. This loss in the order of differentiability can be explained due to fuselage HTP interaction near HTP root. We can conclude that even though we are working with very basic assumptions of differentiability the choice of kernel is important in performing the predictions.



(a) Joint multi-output GPR between K_z and C_z using a **SE kernel** and equation 7.20. Outlier points either do not follow the relationship between outputs or the SE kernel imposes a strict bias



(b) Joint multi-output GPR between K_z and C_z using a twice differentiable **Matérn kernel** and equation 7.20.

Figure 7.5: Joint multi-output GPR on K_z and C_z relationship. The solid black line represents the predicted mean while the shaded area denotes 2σ uncertainty region. The figure demonstrates the impact of the covariance function for relationship enabled regression.

In this section, we compare the predictive capabilities of using two different covariance functions for defining the prior of independent output f^2. For the current case, the SE kernel imposed a very strict bias for the pressure distribution K_z, this puts a few observation points outsize the 95% confidence interval. In fact, the twice differentiable Matérn kernel is a better covariance function for the problem. This is because due to the fuselage wing

interaction, the pressure distribution is not infinitely differentiable near the HTP root.

Identifying faulty sensors We also observe that the current formulation pushes observations that do not follow the physics of the system outside its confidence band, this is a very interesting property and can be used to automatically identify faulty sensors. Suppose we measure the loads on an aircraft using strain gauges, while we measure the acceleration of the aircraft using accelerometers. We can leverage the Newton's third law of motion ($\int loads = MassOfAircraft \times acceleration$) to build a robust loads model and identify faulty strain gauges automatically.

Calculating the log-marginal likelihood involves inverting the matrix $\boldsymbol{K_{XX}} + \boldsymbol{\Sigma}$. The size of the $\boldsymbol{K_{XX}} + \boldsymbol{\Sigma}$ matrix depends on a total number of input points N, hence inverting the matrix becomes intractable for a large number of input points. In the next section we describe how to solve the problem of inverting huge $\boldsymbol{K_{XX}} + \boldsymbol{\Sigma}$ matrices using approximate inference techniques.

7.4 Scaling up MTGP

As already discussed in chapter 3 on scaling GP regression, the GP approach is intractable for large data-sets. This problem is further aggravated when we are creating a joint Gram matrix between several outputs. For a multi-output GP as defined in section 6.2 the covariance matrix is of size N_{joint}, where $\mathcal{O}\left(N_{joint}^3\right)$ time is needed for inference and $\mathcal{O}\left(N_{joint}^2\right)$ memory for storage.

In this section, we extend the formulation of Variational GP approximation (section 3.2.2) and Distributed GPs approximation (section 3.3) for the case of MTGP's. The results of this study were published in ICPRAM 2016 and LNCS 2017 [Chiplunkar 2016c, Chiplunkar 2017c]. The remaining section details the two methods for approximating covariance matrix which can be later used to calculate the predictive mean, predictive covariance and the marginal likelihood (7.14, 7.15 and 7.16).

Publications

7.4.1 Variational Approximation on Multi-output GP

Sparse methods use a small set of M function points as support or inducing variables. Suppose we use M inducing variables to construct our sparse GP. The inducing variables are the latent function values evaluated at inputs $\boldsymbol{X_M}$. An approximation to the true log marginal likelihood in equation 6.23 can allow us to infer these quantities.

We try to approximate the joint-posterior distribution $p(\boldsymbol{X}|\boldsymbol{y})$ by introducing a Variational distribution $q(\boldsymbol{X})$. We use the same inducing points for all the outputs (equation

6.3) and extend the derivation of [Titsias 2009] to the multi-output case.

$$q(\boldsymbol{X}) = \mathcal{N}(\boldsymbol{X}|\boldsymbol{\mu}, \boldsymbol{A}) \tag{7.21}$$

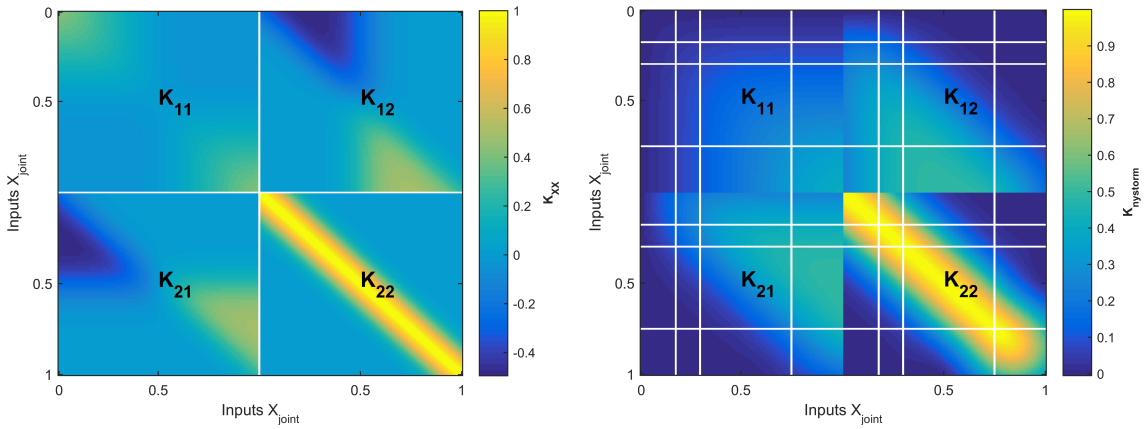
Here $\boldsymbol{\mu}$ and $\boldsymbol{A}$ are parameters of the Variational distribution. We follow the derivation provided in [Titsias 2009] and obtain the lower bound (Fv) of true marginal likelihood.

$$F_V = log(\mathcal{N}[\boldsymbol{y}|0, \sigma^2 \boldsymbol{I}_{N_{joint}} + \boldsymbol{K}_{Nystrom}]) - \frac{1}{2\sigma^2} Tr(\tilde{\boldsymbol{K}}) \tag{7.22}$$

where $\boldsymbol{K}_{Nystrom} = \boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}_M} \boldsymbol{K}_{\boldsymbol{X}_M\boldsymbol{X}_M}{}^{-1} \boldsymbol{K}_{\boldsymbol{X}_M\boldsymbol{X}}$ and $\tilde{\boldsymbol{K}} = \boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}} - \boldsymbol{K}_{Nystrom}$. $\boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}}$ is the joint-covariance matrix derived using equation 7.4 using the input vector $\boldsymbol{X}$. $\boldsymbol{K}_{\boldsymbol{X}_M\boldsymbol{X}_M}$ is the joint-covariance function on the inducing points $\boldsymbol{X}_M$, such that the inducing points are same for all the outputs. While $\boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}_M}$ is the cross-covariance matrix between $\boldsymbol{X}_{joint}$ and $\boldsymbol{X}_M$.

Figure 7.6(b)

Figure 7.6(b) shows the approximate Gram matrix by using the inducing points shown by white lines for the Gram matrix described in the left figure. We see how choosing three same random inducing points for both the outputs (y^1 and y^2) impacts the Gram matrix. As analyzed in the chapter 3 we can improve the approximate Gram matrix if we increase the number of inducing points.



(a) Gram matrix between y^1 and y^2 such that $y^1 = \int_0^x y^2$. SE covariance function is used for the y^2 with $\theta = [1, 0.2]$, and $\boldsymbol{X}^j = [0 : 0.01, 1]\ \forall\ j = [1, 2]$ (figure7.1(b))

(b) Approximated Gram matrix using Nyström approximation for Gram matrix of figure in left. Three inducing points were chosen randomly, the white lines denote the location of inducing points.

Figure 7.6: Approximate Gram matrix for a Joint MTGP kernel using Nyström approximation

Lower bound

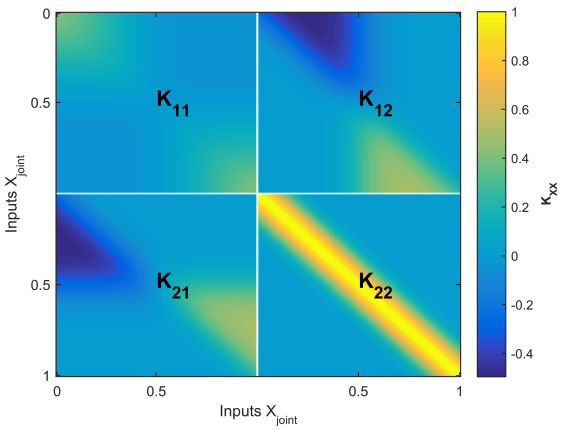
The bound (Fv) can be maximized with respect to all parameters of the covariance function; the model hyper-parameters and the Variational parameters. The optimization

parameters are the inducing inputs $\boldsymbol{X_M}$, the hyper-parameters θ of the independent covariance matrix $\boldsymbol{K_{22}}$ and the error while measuring the outputs σ. There is a trade-off between quality of the estimate and amount of time taken for the estimation process. On the one hand the number of inducing points determine the value of optimized log-marginal likelihood and hence the quality of the estimate. While, on the other hand there is a computational load of $\mathcal{O}\left(N_{joint}(MD_{outputs})^2\right)$ for inference. In upcoming experiments we increase the number of inducing points until the difference between two successive likelihoods is below a predefined quantity.

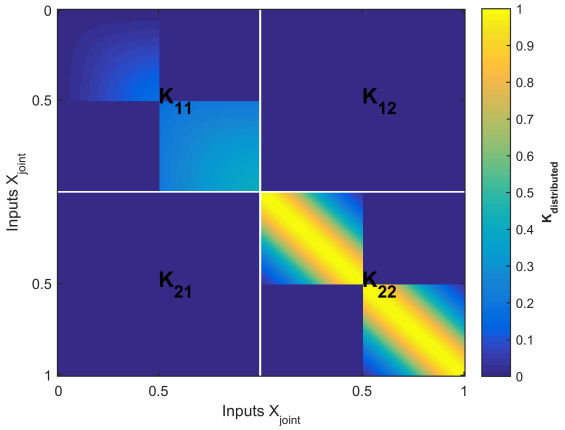
7.4.2 Distributed Inference on Multi-output GP

An alternative to sparse approximations is to learn local experts on a subset of data. We have described Distributed GPs in detail in section 3.3. We again distribute our data-set $\boldsymbol{X_{joint}}$ into smaller experts and use the independence assumption to approximate the Gram matrix.

Figure 7.7 shows the difference between an exact Gram matrix and a Gram matrix approximated using Distributed GPs. Figure 7.7(b) shows the Gram matrix after performing distributed approximation for the Gram matrix described in the left figure. We see how choosing two same uniform experts for both the outputs (y^1 and y^2) impact the Gram matrix. As analyzed in the chapter 3, we can improve the approximate Gram matrix if we increase the number of pointsin each expert.



(a) Gram matrix between y^1 and y^2 such that $y^1 = \int_0^x y^2$. SE covariance function is used for the y^2 with $\theta = [1, 0.2]$, and $\boldsymbol{X}^j = [0 : 0.01, 1]\ \forall\ j = [1, 2]$ (figure7.1(b))



(b) Approximated Gram matrix using Distributed GPs approximation for Gram matrix of figure in left. Points in the experts are distributed uniformly. We can observe that covariance across experts goes to zero

Figure 7.7: Approximate Gram matrix for a SE kernel using mixture of experts.

Equation 7.23 describes the formulation for marginal likelihood for Distributed GPs approximation. 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. After learning the hyper-parameters we can combine the predictions of local experts to give mean and variance predictions (more details in section 3.3.1).

$$\log p(\boldsymbol{y_{joint}}|\boldsymbol{X_{joint}}, \theta) \approx \sum_{k=1}^{M} \log p_k(\boldsymbol{y^{(i)}}|\boldsymbol{X^{(i)}}, \theta) \tag{7.23}$$

rBCM

The robust Bayesian Committee Machine (rBCM) model combines the various experts using their confidence on the prediction point [Deisenroth 2015]. In such manner experts which have high confidence at the prediction points get more weight when compared to experts with low confidence.

$$m(\boldsymbol{y_*}) = (Cov(\boldsymbol{X_*}))^{-2} \sum \beta_k \sigma_k^{-2} m_k(\boldsymbol{X_*}) \tag{7.24}$$

$$(Cov(\boldsymbol{y_*}))^{-2} = \sum_k \beta_k \sigma_k^{-2} + (1 - \sum_k \beta_k)\sigma_{**}^{-2} \tag{7.25}$$

In the above equations $m_k \boldsymbol{X_*}$ and σ_k are the mean and covariance predictions from expert k at point $\boldsymbol{X_*}$. σ_{**} is the auto-covariance of the prior at prediction points X_*. β_k determines the influence of experts on the final predictions [Cao 2014] and is given as $\beta_k = \frac{1}{2}(\log \sigma_{**}^{-2} - \log \sigma_k^{-2})$.

7.5 Experiments

In the current section, we first compare the predictions of independent GPs with respect to joint MTGPs both approximated through Variational approximation. We then empirically compare the performance of Distributed GPs and Variational Inference with respect to the training time and accuracy. Finally, we compare the predictions of Variational joint-GP on a real-world flight-test data-set.

7.5.1 Experiments on Theoretical Data

We consider a derivative relationship between two output functions as described in equation 7.3. Such that

$$f^1 = \frac{\partial f^2}{\partial x} \tag{7.26}$$

Data is generated from equations 7.27, a random function is drawn from GP to get f^2 whose derivative is then calculated to generate f^1. y^1 and y^2 are then calculated by adding noise according the equations 7.27. 10,000 equidistant points are generated between the locations $x \in [-1, 1]$, for both the outputs y^1 and y^2. Values of y^2 are masked in the region $x \in [0, 0.3]$, the remaining points now constitute our training data-set.

$$\begin{aligned} f^2 &\sim GP[0, k_{SE}(0.1, 1)] \\ \sigma_{n2} &\sim \mathcal{N}[0, 0.2] \\ \sigma_{n1} &\sim \mathcal{N}[0, 2] \end{aligned} \tag{7.27}$$

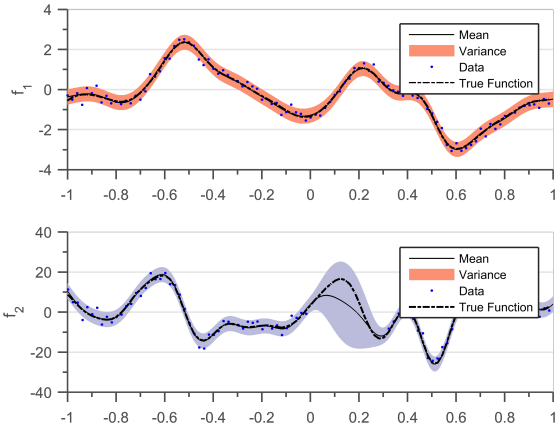
$k_{SE}(0.1, 1)$ means squared exponential kernel with $\theta_{lengthScale} = 0.1$ and $\theta_{amplitude} = 1$. Since the differential relationship ($\mathcal{L}(.)$) is linear in nature, we use the equation 7.4 to calculate the auto- and cross-covariance functions as shown in equation 7.28.

$$\begin{aligned} Cov(f^2(x), f^2(x')) &= \theta^2_{amplitude} exp\left[\frac{-1}{2}\frac{d^2}{\theta^2_{lengthScale}}\right] \\ Cov(f^1(x), f^2(x')) &= \theta^2_{amplitude}\frac{d}{\theta^2_{lengthScale}} exp\left[\frac{-1}{2}\frac{d^2}{\theta^2_{lengthScale}}\right] \\ Cov(f^1(x), f^1(x')) &= \theta^2_{amplitude}\frac{d^2 - \theta^2_{lengthScale}}{\theta^4_{lengthScale}} exp\left[\frac{-1}{2}\frac{d^2}{\theta^2_{lengthScale}}\right] \end{aligned} \tag{7.28}$$

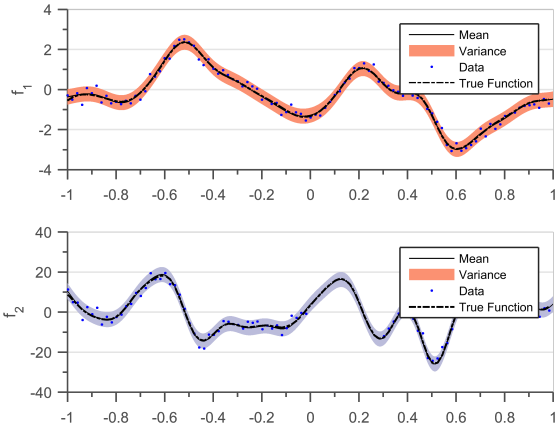
Figure 7.8 shows comparison between an independent fit GP and a joint multi-output GP whose outputs are related through a derivative relationship described in equation 7.26. For figure 7.8(a), using Variational inference algorithm, we optimize the lower bound of log marginal likelihood for independent GP's on y^1 and y^2. For figure 7.8(b), using Variational inference, we optimize the same lower bound but with a joint-covariance approach as described in section 7.4.1 using y^1, y^2 and $\mathcal{L}(.)$. We settled on using 100 equidistant inducing points for this exercise [Chiplunkar 2016c] and have only optimized the hyper-parameters to learn the model.

Figure 7.8(a) shows the independent fit of two GP for the differential relationship while, figure 7.8(b) shows the joint GP fit. The GP model with joint-covariance gives better prediction even in absence of data of y^2 for $x \in [0, 0.3]$.

Comparison between Distributed GPs and Variational GPs For the second experiment we compare the Root Mean Squared Error (RMSE) and run-times of Distributed GPs and Variational Inference algorithms while performing approximate inference. We progressively generate from 10^3 to 10^5 data-points according to the equations 7.27 and 7.26. We separated 75% of the data as the training set and 25% of the data as the test set,



(a) **Independent** fit for two GP's, experiment was run on 10,000 points but only 100 data-points are plotted to increase readability. Inference is performed using Variational inference algorithm and equidistant 100 inducing points. We can observe the huge difference between the real data and the predicted mean values at zone with no data.

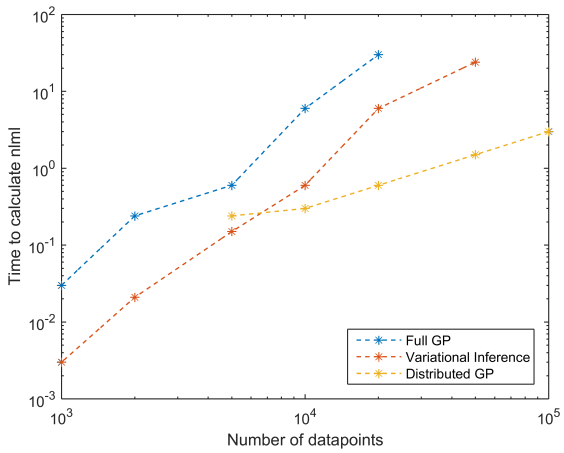


(b) **Joint multi-output GP's** for two outputs, experiment was run on 10,000 points but only 100 data-points are plotted to increase readability. Inference is performed using Variational inference algorithm and equidistant 100 inducing points. We can observe the improved prediction between zone with no data because information is being shared between the two outputs.

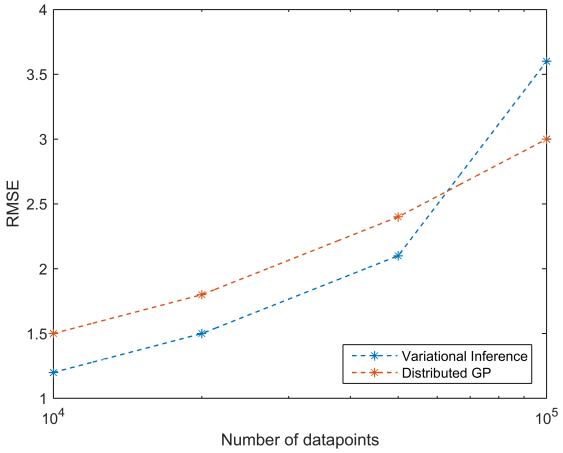
Figure 7.8: Experimental results for differential relationship while using Variational approximation. Predicted mean is represented by solid black line. 2σ confidence band is represented by light red for f^1 and light blue for f^2. The dashed black line represents the true latent function values

the training and the test sets were chosen randomly. The Variational inference relationship kernel as described in section 7.4.1 with 100 equidistant inducing points was used. We learn the optimal values of hyper-parameters for all the sets of training data. The Distributed GPs algorithm as described in section 7.4.2 was used with randomly chosen 100 points per expert. We learn the optimal values of hyper-parameters for all the sets of training data. The accuracy is plotted as RMSE values with respect to the test set. The run time is defined as time taken to calculate the approximate marginal likelihood equations 7.22 and 7.23. The RMSE values are calculated for only the dependent output y^1 and then plotted in the figure 7.9(a).

In figure 7.9(a) the time to calculate marginal likelihood with increasing number of training points is calculated. As expected the full GP takes more time when compared to Variational inference or Distributed GPs algorithms. The Variational inference algorithm has better run-time till $\mathcal{N} \sim 10^4$ data-points, after that Distributed GPs takes lesser time. In figure 7.9(b), the RMSE error with test set is compared between the Variational inference and Distributed GPs algorithm. Here too the Variational inference algorithm performs better for a lesser number of data-points, but Distributed GPs starts performing better when we reach more than $\mathcal{N} \sim 10^4$ data-points.



(a) Comparison of time to calculate log marginal likelihood for a Full GP, Variational inference and Distributed GPs with increasing number of data-points. We observe for data-points greater that 10^5 the Distributed GPs algorithm starts outperforming Variational inference



(b) Comparison of RMSE for Variational inference and Distributed GPs algorithm. We observe for data-points greater that 10^4 the Distributed GPs algorithm starts outperforming Variational inference.

Figure 7.9: Comparison of run time and RMSE between Distributed GPs and Variational Inference

As we keep on increasing the number of data-points, for a fixed set of inducing points Variational inference starts performing worse. This is because as we have already observed we need $M \sim \frac{N}{10}$ for good accuracy with inducing inputs approximation. If we keep the value of $M \sim \frac{N}{10}$ for $N \sim 10^5$ then the size of matrix $\boldsymbol{K_{MM}}$ will become very big to invert efficiently. Thus even with inducing inputs approximation we have an upper limit to the possible number of meaningful GP models. One thing to note is that we have fixed the number and position of inducing points while optimizing hyper-parameters for this experiment. While a more optimized set of inducing points will have better results, for data-sets of the order $\mathcal{N} \sim 10^5$ Distributed GPs algorithm starts outperforming Variational inference. For the case of Distributed GPs we have again fixed the number of points in an expert, but this does not have a significant impact on the trend on RMSE with increasing data-points.

7.5.2 Experiments on Flight Test Data

We perform experiments on flight loads data produced during flight test phase at Airbus. Loads are measured across the wingspan using strain gauges. Shear load T_z and bending moment M_x as described in figure 7.10 are used as two outputs for this exercise. η or point of action of forces and angle of attack α are the two inputs. The aircraft is in quasi-statis equilibrium in all conditions and there are no dynamic effects observed throughout this

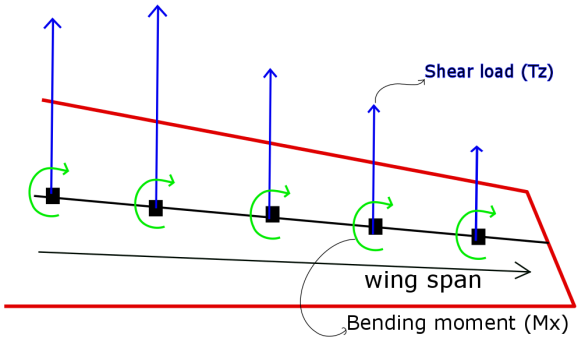


Figure 7.10: Wing Load Diagram

data-set. All data is normalized according to Airbus policy.

The relation between T_z and M_x can be written as:

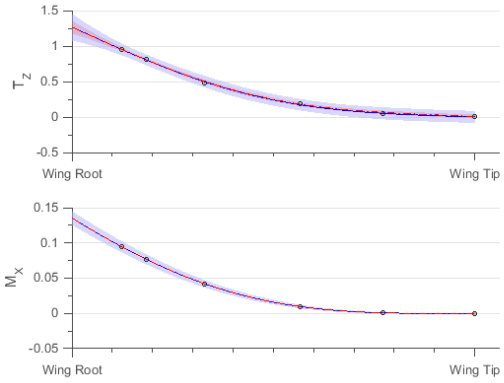
$$M_x(\eta, \alpha) = \int_{\eta}^{\eta_{edge}} T_Z(x, \alpha)(x - \eta)dx \tag{7.29}$$

.

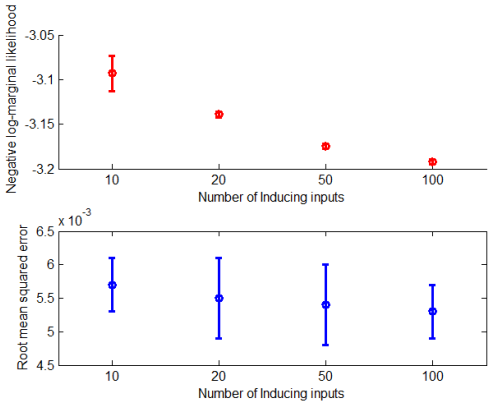
The integral operator in equation 7.29 is applied on the η dimension. Here, η_{edge} denotes the edge of the wingspan. The forces are measured at 5 points on the η dimension and at 8800 points on the α dimension. In this experiment we first compare plots of joint MTGP and independent GP, and then compare the measures of log marginal likelihood and RMSE for varying number of inducing points.

Figure 7.11(a) shows the independent (blue) and joint fit (red) of two GP. The top figure shows T_Z with the variance of dependent GP plotted in red and variance of independent GP plotted in blue. The bottom figure shows plots for M_X. 100 inducing points in the input space are used to learn and plot the figure. The variance of red is smaller than that of blue showing the improvement in confidence when imposing relationships in the GP architecture. The relationship between T_Z and M_X gives rise to better confidence during the loads prediction. This added information is very useful when identifying faulty sensor data since equation 7.29 will push data-points which do not satisfy the relationship out of the tight confidence interval.

Figure 7.11(b) shows improvement in the negative log-marginal likelihood and RMSE plots upon increasing number of inducing points. 10 sets of experiments were run on 75% of the data as training set and 25% of the data as the test set, the training and the test sets were chosen randomly. We learn the optimal values of hyper-parameters and inducing points for all the 10 sets of experiments of training data. Finally, RMSE values are evaluated with respect to the test set and log-marginal likelihood are evaluated for



(a) 2σ confidence interval and mean of the dependent GP are represented in red shade and solid red line. 2σ confidence interval and mean of the independent GP are represented in blue shade and solid blue line. Experiment was run on 8800 data-points Noisy data is denoted by circles only 1 α step is plotted. Confidence interval improves upon adding the relationship kernel.



(b) Progression of RMSE and log-likelihood upon increasing number of inducing points. Top plot shows the value of mean and variance of log-marginal likelihood. The bottom figure in blue shows the mean and variance of root mean squared error. 10 sets of experiments were run on 75% of the data as training set and 25% of the data as the test set, the training and test sets were chosen randomly.

Figure 7.11: Experimental results for aircraft flight loads

each learned model. The RMSE and log-likelihood improve upon increasing the number of inducing points.

Cost saving in MDO By enabling relationships in building models we can save significant costs while performing MDO. Earlier works have reduced the cost of optimization by incorporating Gradient Enhanced Kriging into the optimization process [Liem 2015], a similar cost saving can be obtained by using the proposed methods. Suppose we are designing a part of an aircraft by performing optimization over a fluid-structure interaction loop. The fluid part is being solved by a costly CFD code, while the structural part is being solved by a cheap CSM code. The CFD code results in *pressures* on the aerodynamic mesh, while the CSM part results in $deformations$ of the structural mesh. If we can leverage the relationship between pressure and deformation[3] then we can effectively reduce the number of required CFD runs. These type of cost savings can be made in several MDO problems.

[3] $\int pressures = Stiffness \times deformation$

7.6 Summary and discussion

The current chapter demonstrates how to incorporate physical laws into MTGPs regression framework and then eventually how to scale the framework to large data-sets. Section 7.2.1 demonstrates how to enforce physical laws, which are in the form of linear operators. Whereas section 7.2.2 demonstrates how to encode physical laws, which are in the form of non-linear operators. The improved accuracy of the model is then demonstrated both on a toy data-set and a data-set of real flight-test (section 7.3).

Section 7.4 demonstrates how to scale the MTGP to large number of data-points, both using a Variational approximation (section 7.4.1) and a Distributed GPs approximation (section 7.4.2). We then demonstrate the scalability on a toy data-set and a data-set from flight test and compare the accuracy of Distributed GPs and Variational inference on the toy-data-set.

Several future paths can be explored by leveraging this kind of relationship enabled MTGPs. Using a quadratic relationship enabled MTGP we can indirectly add inequality constraints. Leveraging the information of prior relationships can greatly reduce the cost of performing MDO. Finally, by enforcing physical laws of the system we can automatically identify faulty sensors. All these are interesting paths and should be explored further.