Fast emulation and calibration of large computer experiments with multivariate output

by

Grace G. Hsu

BSc., University of British Columbia, 2016

Project Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in the Department of Statistics and Actuarial Science Faculty of Science

© Grace G. Hsu 2019
SIMON FRASER UNIVERSITY
Spring 2019

Copyright in this work rests with the author. Please ensure that any reproduction or re-use is done in accordance with the relevant national copyright legislation.
Approval

Name: Grace G. Hsu

Degree: Master of Science (Statistics)

Title: Fast emulation and calibration of large computer experiments with multivariate output

Examiner Committee: Chair: Jinko Graham
Professor

Derek Bingham
Senior Supervisor
Professor

Richard Lockhart
Supervisor
Professor

Dave Campbell
Internal Examiner
Associate Professor

Date Defended: 2019 April 17
Abstract

Scientific investigations are often expensive and the ability to quickly perform analysis of data on-location at experimental facilities can save valuable resources. Further, computer models that leverage scientific knowledge can be used to gain insight into complex processes and reduce the need for costly physical experiments, but in turn may be computationally expensive to run. We compare multiple statistical surrogates or emulators based on Gaussian processes for expensive computer models, with the goal of producing predictions quickly given large training sets. We then present a modularised approach for finding the values of inputs that allow for the surrogate model to match reality, or field observations. This process is model calibration. We then extend the emulator of choice and calibration procedure for use with multivariate response and demonstrate the speed and efficacy of such emulators on datasets from a series of transmission impact experiments.

Keywords: Gaussian process regression; fast model emulation; multivariate response; computer experiments; model calibration; modularisation
Acknowledgements

Thank you to my supervisor Dr. Derek Bingham for taking a chance on me, for teaching me many additional skills useful for life and industry work, and of course all his understanding and patience. I’d also like to thank SFU’s Statistics and Actuarial Science department. I feel so lucky to have had courses with such a wonderful instructor (Rachel) that they induced interest in a subject where feelings were rather lukewarm. I’m sure I would have passed through the program with at least 80% less joy without my fellow master’s students–to the awesome ladies Ran, Michelle, Lillian, Yue, (and Richard of course), I’m so glad you were all there! Thank you also to my fellow MTEK Sciences colleagues, who have so graciously supported me and been incredibly understanding through this project. Special thanks to my academic brother Ofir and sister Shirin for their help and advice when I was drowning in confusion, and Ellie for holding my hand when difficulties arose and refused to die quickly. To my parents, I can never convey how much I appreciate their encouragement, and every other variation of the word “support” that exists. I cannot be grateful enough for all they’ve done. Finally, a resounding thanks to all others who have supported me–I’m humbled that I can say there are more than I can properly name here.
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Approval</td>
<td>ii</td>
</tr>
<tr>
<td>Abstract</td>
<td>iii</td>
</tr>
<tr>
<td>Acknowledgements</td>
<td>iv</td>
</tr>
<tr>
<td>Table of Contents</td>
<td>v</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>2 Motivation</td>
<td>2</td>
</tr>
<tr>
<td>2.1 Dynamic compression experiments</td>
<td>3</td>
</tr>
<tr>
<td>2.2 The Data</td>
<td>4</td>
</tr>
<tr>
<td>3 Model Emulation</td>
<td>9</td>
</tr>
<tr>
<td>3.1 Notation and Setup</td>
<td>10</td>
</tr>
<tr>
<td>3.2 Bayesian Treed Gaussian Process Models (TGP)</td>
<td>12</td>
</tr>
<tr>
<td>3.3 GP Models Using Compactly Supported Correlation Functions (CSC)</td>
<td>14</td>
</tr>
<tr>
<td>3.4 Locally Approximate Gaussian Process Models (LAGP)</td>
<td>14</td>
</tr>
<tr>
<td>3.5 Generalised Polynomial Chaos Models (GPC)</td>
<td>16</td>
</tr>
<tr>
<td>4 Comparison</td>
<td>18</td>
</tr>
<tr>
<td>4.1 Training and Testing Procedure</td>
<td>19</td>
</tr>
<tr>
<td>4.2 Results</td>
<td>21</td>
</tr>
<tr>
<td>4.3 Recommendations</td>
<td>31</td>
</tr>
<tr>
<td>5 Model Calibration</td>
<td>33</td>
</tr>
<tr>
<td>5.1 Methodology and procedure</td>
<td>34</td>
</tr>
<tr>
<td>5.2 Examples</td>
<td>36</td>
</tr>
<tr>
<td>6 Return to the materials science problem</td>
<td>42</td>
</tr>
<tr>
<td>6.1 Modelling data with multivariate output</td>
<td>42</td>
</tr>
<tr>
<td>6.2 Emulation</td>
<td>43</td>
</tr>
<tr>
<td>6.3 Calibration using multivariate output</td>
<td>52</td>
</tr>
</tbody>
</table>
6.3.1 Methodology and procedure ........................................... 52
6.3.2 Results ............................................................................ 54

7 Discussion ............................................................................ 61

Bibliography ........................................................................... 63
Chapter 1

Introduction

Scientific investigations are often expensive. Whether the cost is in computation, labour, or otherwise, the ability to quickly perform analysis of data on-location at experimental facilities can save valuable resources. Modern experiments often require time at a facility where scientists operate equipment with errors that may be difficult to detect until the data are analysed. Further, computer models that leverage scientific knowledge can be used to gain insight into complex processes. These models may defray the cost of physical labour required to conduct physical experiments, but in turn may be computationally expensive to run. This project compares multiple statistical surrogates or emulators for expensive computer models, with the goal of producing predictions quickly given large training sets. We then present an approach for finding the values of inputs that allow for the surrogate model to match reality, or field observations. This process is model calibration, for which a well-known statistical framework was proposed by Kennedy and O’Hagan (2001). Specifically, to further increase the speed of the calibration process, in addition to using a fast emulator we demonstrate the efficacy of modularised calibration. Altogether, the result is a surrogate model that can handle large datasets, make predictions quickly, and together with modularisation perform fast model calibration. Such surrogate models fit into and accelerate the current and future workflow pattern in experimental science.

The project is organised as follows: We begin in Chapter 2 by presenting the application and motivating problem. An introduction to Gaussian processes (GPs) and candidate emulation methods are given in Chapter 3. After comparing multiple fast emulators in Chapter 4, we demonstrate modularised calibration in Chapter 5. Up to that point all test functions considered have scalar responses. In Chapter 6, we return to our motivating problem by extending both the emulator of choice and calibration procedure for use with a multivariate response and demonstrate both on data from experiments in materials science.
Chapter 2

Motivation

Scientists conducting experiments at experimental facilities are often limited in their ability to perform data analysis on-location. In areas such as materials science, dynamic compression experiments require time at a facility where scientists operate equipment that can have erratic errors. For example, lasers may misfire but there is no way of catching such errors until the data are analysed. However current data analysis tools for materials simulation and analysis are so time-consuming, their results cannot be used to improve experiments while the scientists have access to the experimental facilities. Aside from inefficiencies, errors can be disastrous as they mean more time at facilities is needed, and thus more funding is required. Slow analyses and extraction of knowledge from the data are bottle-necks during the scientific process. These problems will only become more pronounced as the rate of data generation continues to increase.

The Los Alamos National Laboratory (LANL) aims to combine computer and statistical science to develop a toolset that can perform real-time analysis. Specifically, emulators can be used to stand in for complex computer models. A fast emulator would play a key role in allowing for real-time feedback to the scientists in the form of experimental design choices to reduce predictive and parameter uncertainty. LANL is developing this toolset using data from dynamic compression experiments, an area of importance for LANL, however many results can be reused for experiments in other scientific domains (Ahrens, 2017).

This project has three main parts, the first goal being a methodic comparison of several fast statistical emulators. The comparison is done using the simulated output from multiple computer experiments where the true models are known. In addition to having low computation time for training the emulator and prediction of the simulator response at unsampled points, their performance in terms of predictive accuracy will also be assessed. Thus, the results from the first part are generally applicable to experiments where fast emulation is needed or desired. Chapter 3 provides an overview of the candidate methods chosen, followed by Chapter 4, which describes the methodology used for comparison and results.
The next part of the project concerns a statistical method called model calibration (Kennedy and O’Hagan, 2001). Model calibration combines field data and computer model output to (i) build a predictive model for the physical system, and (ii) estimate the parameters that govern the behaviour of the simulator. Part (i) can be seen as a type of regression problem that uses two sources of information, whereas part (ii) can be viewed as solving an inverse problem. Chapter 5 introduces an approach where calibration is “modularised” into separate parts and calibration parameter estimation is treated as an optimisation problem. This results in a thrifty calibration method compared to the traditional framework given by Kennedy and O’Hagan (2001). We describe this fast calibration method and give examples using data from known models. Up until this point, all methodology is given for data with a scalar response.

The final goal is to assemble and extend the previous two parts by emulating and calibrating using multivariate responses. A complete demonstration is presented using data from a set of dynamic compression experiments. Altogether we present a fast model emulation and calibration framework for multivariate output that scales well with large datasets. This is a major component of real-time analysis at experimental facilities, which is key for achieving higher quality experimentation in a shorter amount of time and thus lowering funding.

### 2.1 Dynamic compression experiments

In the fields of materials science and shock physics, dynamic compression experiments subject materials to various conditions such as compression, high temperature, and deformation, on very short time scales (picosecond to microsecond). The result is some set of physical and chemical responses, which is intended to be used to aid in understanding of material behaviour and transformations under usually extreme conditions. The aerospace, automotive, and defence industries are among those that aim to predict the behaviour of materials when subjected to such stresses. The parameters in dynamic compression experiments may be a mix of user-set or those that require estimation, whereas the measured responses may be velocimetry, diffraction, and/or imaging data.
Examples of dynamic compression experiments are plate impact experiments, which study the effects of peak shock stress and pulse duration on the spall response of materials such as aluminum. The setup is shown in Figure 2.1.1. In this illustration, a flyer plate is shot via the barrel and impacts the target plate sample. The responses collected may include the velocity of the back surface of the flyer plate measured over time, the velocity at impact, variables relating to compression, etc. This means that each observation is a “shot” resulting in a response in the form of a time series.

Note that often in experimental science, aside from settable input parameters, there are many parameters that may not be known, such as material strength, deformation mechanisms, shock pressure, etc. For example, it is reasonable to expect that material strength would be an input when modelling a plate impact experiment and indeed a key part of understanding the physical process as a whole. However at the same time, material strength is unknown. When a model input requires estimation, it is called a calibration variable; calibration variables will be discussed in Chapter 5. The data used for this project originate from a series of dynamic compression experiments and are discussed in the following section.

### 2.2 The Data

This project uses data generated by Walters et al. (2018). They simulated the data using a computer, incorporating aluminum plate impact data from a series of transmission impact experiments (Boteler and Dandekar, 2006). Boteler and Dandekar (2006) investigate the mechanical response of an aluminium alloy with magnesium and traces of manganese and chromium (Al-5083). The response values of interest are velocities measured over time,
resulting in a time series or *velocimetry profile*. These were recorded using VISAR interferometry, which has a reported 1% precision over the entire time range of interest. From a total of eight experiments done by Boteler and Dandekar (2006), Walters et al. (2018) chose three shots for their work. Specifically, all three shots were symmetric plate impact experiments, meaning that the flyer and target plate were the same material, thus reducing the uncertainty as only one material requires characterisation.

The shot characteristics, such as flyer and target thickness, for the three chosen experiments shot 104, shot 105, and shot 106 (abbreviated as 104S, 105S, and 106S respectively), can be found in Table I of Walters et al. (2018). As is common for expensive experiments, a limited number of runs can be done. Indeed, for each shot there is only a single velocimetry profile available.

In order to generate training data for a statistical emulator, Walters et al. (2018) simulate more velocimetry profiles across a range of input settings. The computer model simulator, represented by the black box in Figure 2.2.2, takes a variety of input variables and outputs the velocity of the velocity of the flyer plate as a function of time. The two main components of the model are the empirical Johnson-Cook plasticity model and the Mie-Grüneisen equation of state. The Mie-Grüneisen equation of state is a thermodynamic model that describes changes in material volume and we omit its details as none of its variables are relevant for this project. We focus mainly on the Johnson-Cook model, which describes plasticity and dynamic strength via the flow stress $Y_f$, and uses a relatively small number of parameters:

$$Y_f = (a + b\varepsilon_p x_n) \left(1 + c \log \frac{\dot{\varepsilon}_p}{\dot{\varepsilon}_0}\right) \left(1 - (T^*)^{x_m}\right).$$

(2.2.1)

The inputs, called material constants, are $a, b, c, x_n,$ and $x_m$. They describe the physical properties of the flyer plate material. The initial yield stress is $a$. The equivalent strain is commonly used to describe strain/deformation in solids and is represented by $\varepsilon_p$. Strain rate is the change in strain/deformation of a material with respect to time and is represented by $\dot{\varepsilon}_p$, whereas the nominal strain rate is $\dot{\varepsilon}_0$. Here, $\dot{\varepsilon}_0$ is fixed at $1.0s^{-1}$. The contribution of the plastic strain rate $\dot{\varepsilon}_p$ and nominal strain rate $\dot{\varepsilon}_0$ on the response are both affected by parameter $c$. From the equation we can see that $b$ and $x_n$ control strain hardening effects (i.e. plastic deformation) related to $\varepsilon_p$. Lastly, the temperature $T^*$ and exponent $x_m$ describes thermal softening behaviour. Those variables in Equation 2.2.1 other than $a, b, c, x_n,$ and $x_m$ are either fixed or calculated in other parts of the computer model.

Walters et al. (2018) chose the Johnson-Cook model because of its simplicity and common use in materials science. The simulations were done using a multiphysics continuum hydrodynamics code, FLAG, developed by LANL for simulating high strain-rate and large deformation response of materials. The settings chosen for the simulations are such that the input values are spread over each of their plausible ranges.
Apart from parameters from the Johnson-Cook model, Walters et al. (2018) include additional known features of the experiment in order to improve the overall computer model. One additional input is the shear modulus, \( G \), which is a measure of a material’s stiffness and physically depends on the density and temperature of the material. The shear modulus value depends on the temperature at which the value is measured. Harder materials have a higher shear modulus value, which implies that greater force or stress is needed to strain or deform the material.

The second additional input is the velocity of the flyer plate, \( v_{el} \), as it passes through the velocity pins depicted in Figure 2.1.1. The simulation model includes \( v_{el} \) because Boteler and Dandekar (2006) describe a \( \pm 2\% \) uncertainty of the flyer velocity at impact. This uncertainty in the impact velocity significantly affects the response, which is the final velocity of the target free surface. A way to incorporate the flyer velocity as a source of uncertainty in the computer model is to include it as an input and vary it about the corresponding nominal values.

We explore three shots, each under different experimental conditions. For each shot, Walters et al. (2018) generate 1000 simulations from a Latin hypercube design. The covariates are \( a, b, c, x_n, x_m \) (parameters for the Johnson-Cook materials model), \( v_{el_i} \) (the velocity of the flyer from the \( i \)-th experiment), and \( G_i \) (the average shear modulus of the target plate in the \( i \)-th experiment). Inputs \( a, b \) and \( G_i \) are all measured in megabars (Mbar). The flyer velocity \( v_{el_i} \) is measured in cm/\( \mu s^{-1} \), and the remaining variables, \( c, x_n \) and \( x_m \) do not have associated units.

Parameters \( a, b, c, x_n, x_m \) are those that require estimation. That is, recall that statistical model calibration includes estimating parameters that produce results that best match the measured data. This means we can see calibration as an inverse problem. Regardless of the specific approach taken, calibration requires evaluating a computer model repeatedly, which means that building a fast emulator instead of directly using the complex physics simulator for the model allows for faster calibration.

The experimental and simulated velocimetry profiles appear continuous and non-decreasing. Following Walters et al. (2018), we assume no uncertainty in computer simulations. For each time series, out of the thousands of measurements or simulated points there are four points of interest, \( (t_i, v_i) \), \( i \in \{1, 2, 3, 4\} \), for materials scientists. The scientists can usually identify these four points by visual inspection.
Figure 2.2.1: The velocimetry profile of the flyer plate. The four points of interest for each response are in red and scientists can usually identify these points by eye (Walters et al. (2018)).

In Figure 2.2.1 the first point is the elastic-plateau \((t_1, v_1)\), the second and third together give the elastic-plastic slope \((v_3 - v_2)/(t_3 - t_2)\), and the last point is the plastic-plateau \((t_4, v_4)\). Thus, the full response vector for each observation is 4-dimensional, which requires a different modelling procedure than that for a scalar response. Note that we can easily imagine alternate problems where the times at which the velocities of interest occur are part of the response, making the response vector \((v_1, v_2, v_3, v_4, t_1, t_2, t_3, t_4)\). For this project, the response is the 4-dimensional \((v_1, v_2, v_3, v_4)\) in order to facilitate comparison with Walters et al. (2018). Thus, the response matrix for each experiment has 1000 rows and 4 columns. It is scaled such that each response variable has mean 0 and standard deviation of 1 across the 1000 observations.

In summary, the original experimental response data are three sets of velocity measurements of the target free surface upon impact of the flyer plate under different physical conditions recorded by Boteler and Dandekar (2006), shown as the \(y_F\) symbol in Figure 2.2.2. This limited data, along with the Johnson-Cook model and the Mie-Grüneisen equation of space together are used for the hydrocode simulations done by Walters et al. (2018). These simulations are the datasets used for this project and are represented by the \(y_M\) symbol in Figure 2.2.2.
Figure 2.2.2: An experimental science workflow pattern. The grey box covers the work already done that this project uses. The computer model/simulator is often highly complex and expensive to run, incorporating multiple physics models and equations, making it a black box. The field experiment too is often expensive in terms of labour or access to equipment/facilities. The two blue boxes represent the parts of the workflow this projects aims to perform as fast as possible. This allows the green diamonds to occur in an efficient manner, which ultimately feeds back to the beginning of the workflow and resulting in the highest quality experimentation possible.

Figure 2.2.2 is a birds-eye overview of how this project fits into an experimental science workflow. The blue boxes cover two main steps that we seek to make as fast as possible. This in turn allows for lower cost, both computational and monetary, and improve scientists’ ability to make optimal experimental design decisions. Since understanding complicated processes is typically an iterative process, speeding up the workflow results in higher quality and more efficient experimentation and analysis in modern science experiments.
Chapter 3

Model Emulation

When the data consists of output from a deterministic computer model/simulator or experiment, a statistical surrogate/emulator for the computer model trained using this output is a common approach alleviating computational expense or numerical difficulties associated with the computer model (e.g. Gramacy and Lee (2008); Jones et al. (1998), etc.). GP regression, also called kriging, originated from geostatistics (see Diggle and Ribeiro Jr. (2010) for an overview). Savitsky et al. (2011) and Shang and Chan (2013) demonstrate GP regression with non-Gaussian responses.

GPs are highly flexible regression models, making them popular in general as well as for analysing and designing computer experiments (Sacks et al., 1989). Fitting an emulator also allows for difficult optimisation problems to be solved (e.g. Gramacy and Lee, 2008).

GP models are used as emulators for two main reasons: the ability to interpolate a deterministic simulator, and because they provide a foundation for quantifying uncertainty in a deterministic setting. Interpolation is desirable since we often assume that computer model output is noise-free. GPs are highly flexible in the sense that it can capture highly complicated response surfaces. The covariance structure of a GP also allows correlation to be included in a model.

Fitting a GP requires evaluation of the likelihood. To do so requires the inversion of an $n \times n$ covariance matrix (see Section 3.1), where $n$ is the total number of observations. For large $n$, matrix inversion is slow, which in turn makes inference via maximum likelihood and Bayesian approaches slow as well. Thus, for computer experiments with large $n$ such as the transmission impact experiments in this project, there is a need for alternatives to the traditional full GP. The candidate methods are: Bayesian treed GPs (TGP), GP models using compactly supported covariance functions (CSC), locally approximate GPs (LAGP), and generalised polynomial chaos (GPC). After setting up the notation, the rest of this chapter introduces the four methods and motivation for their inclusion as candidate models. We compare these model emulation methods via a simulation study in Chapter 4.
3.1 Notation and Setup

Suppose we have inputs \( x \in D \subset \mathbb{R}^d \) corresponding to univariate outputs \( Y(x) \), where \( D \) is the input space. A design matrix \( X \) of \( n \) inputs is defined as follows:

\[
X = \begin{bmatrix} x_1^T \\ \vdots \\ x_n^T \end{bmatrix} = \begin{bmatrix} x_{11} & \cdots & x_{1d} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nd} \end{bmatrix},
\]

(3.1.1)

where \( x_{ij} \) is the \( j \)th dimension of the \( i \)th input \((i \in \{1,\ldots,n\}, j \in \{1,\ldots,d\})\). Let \( y = Y(X) = [Y(x_1),\ldots,Y(x_n)]^T \) be the observed response vector.

The GP model

We assume that the response \( Y(x) \) is a Gaussian process (GP), which is completely specified by a mean function \( \mu(\cdot) \) and a covariance (or kernel) function \( K(\cdot) \). Given \( x, x' \in D \),

\[
E[Y(x)] = \mu(x) \\
Var[Y(x)] = \sigma^2 \\
Cov(Y(x), Y(x')) = K(x, x'; \sigma^2, l) = \sigma^2 R(x, x'; l).
\]

(3.1.2) \hspace{1cm} (3.1.3) \hspace{1cm} (3.1.4)

The correlation function \( R(\cdot) \) is positive-definite and is equal to the covariance function divided by \( \sigma^2 \), the marginal variance of \( Y \). The \( d \)-dimensional \( l \) is a vector of positive characteristic length-scale parameters that controls the covariance structure of the process (Rasmussen and Williams, 2006).

For all finite \( n \), \([Y(x_1),\ldots,Y(x_n)]^T\) follows a multivariate normal distribution, where the mean is \( \mu = [\mu(x_1),\ldots,\mu(x_n)]^T \), and the covariance matrix is \( K = \sigma^2 R \), where \( K_{ij} = K(x_i, x_j; \sigma^2, l) \). That is, for \( n \) samples

\[
Y(x) \sim N\left( \mu, \sigma^2 R \right),
\]

(3.1.5)

and so the loglikelihood is

\[
\log l(y|x) = -\frac{1}{2}(y - \mu)^T (\sigma^2 R)^{-1} (y - \mu) - \frac{1}{2} \log |\sigma^2 R| - \frac{n}{2} \log (2\pi).
\]

(3.1.6)

Note that evaluating the likelihood requires the inversion of the covariance matrix, which is written in terms of the correlation matrix \( R \). Since \( R \) is \( n \) by \( n \) and matrix inversion has complexity \( O(n^3) \), fitting a GP for data with large \( n \) means that likelihood evaluation is slow. Later we will see that matrix inversion is required for prediction as well.

The choice of a GP’s covariance function determines other properties of a GP. Some terms that often appear in GP-related literature are (Grimmett and Stirzaker, 2001):
• **Stationarity**: if a GP is stationary, then the process’ behaviour does not depend on the actual position of the points, only their vector difference \((\mathbf{x} - \mathbf{x}')\).

• **Isotropy**: If a GP is isotropic, then the process only depends on the Euclidean distance \(|\mathbf{x} - \mathbf{x}'|\) between points.

• **Homogeneity**: If a GP is homogeneous, then it is stationary and isotropic.

A common choice for the covariance function is one from the isotropic power family, more specifically the *squared exponential (SE)* covariance function:

\[
K(\mathbf{x}, \mathbf{x}'; \sigma^2, l) = \sigma^2 \exp\left\{ -\frac{1}{2l^2} |\mathbf{x} - \mathbf{x}'|^2 \right\}.
\]  
(3.1.7)

There is a single length-scale parameter \(l > 0\) and \(K\) is infinitely differentiable with respect to all its arguments. However, anisotropic covariance functions are typical for computer experiments because of the different input scales and their possibly very different effect on the response.

When a different length-scale parameter \(l_i\) is allowed for each dimension \((i \in \{1, \ldots, d\})\), the result is a *separable* covariance function that is anisotropic (though still stationary). Assuming separability and using isotropic covariance functions in a product form is a simple way of getting an anisotropic covariance function. Thus we have the separable squared exponential covariance function:

\[
K(\mathbf{x}, \mathbf{x}'; \sigma^2, l) = \sigma^2 \prod_{j=1}^{d} \exp\left\{ -\frac{1}{2l_j^2} |x_j - x'_j|^2 \right\}.
\]  
(3.1.8)

Increasing \(l_j\), the length-scale in the \(j\)th dimension, increases the correlation between the response in the \(j\)th direction. This means that as inputs become farther away from each other, the correlation between points drops slowly. The ability to specify a different length-scale for each input allows for more flexibility.

There are many other covariance functions that can be chosen based on a problem’s needs. For example, the Matérn class of covariance functions tend to be more numerically stable than the SE function, but are not infinitely differentiable (e.g. Rasmussen and Williams (2006); Golchi et al. (2013), etc.). If there are multiple inputs, instead of using a separable covariance function, Gramacy and Lian (2012) propose the use of a single-index model, where the simulator response surface is modelled as a 1-dimensional function using a linear combination of the inputs.

**Prediction**

Using a statistical surrogate for emulation does not mean that the output of the simulator is seen as stochastic. The mean and variance allow for uncertainty quantification associated with the output for unsampled inputs of the simulator. At the design/training points the GP
interpolates, which is desirable in a deterministic setting. At unsampled inputs the GP lends itself to prediction and because we do not know which realisation of the GP is the truth. Since assuming that the response is a GP means that the observations $y$ have a multivariate normal distribution, the conditional distribution of $Y(x)$ at any new input $x$ given $y$ is normal due to the properties of the multivariate normal distribution with mean and variance as follows (Rasmussen and Williams, 2006):

$$
E \left[ Y(x) \mid y, \mu, \sigma^2, l \right] = \mu + r^T R^{-1} (y - \mu) 
$$

$$
Var \left[ Y(x) \mid y, \mu, \sigma^2, l \right] = \sigma^2 \left[ 1 - r^T R^{-1} r \right],
$$

where $r = [R(x_1, x), ..., R(x_n, x)]^T$ is the $n$-vector of correlations between the observed responses and $Y(x)$, and $R$ is the correlation matrix where $R_{ij} = R(x_i, x_j; \sigma^2, l)$.

The formulas above are also called the kriging mean and variance formulas. From there we can see clearly that as with evaluating the likelihood, inverting the correlation matrix $R$ is required. This leads to computational intractability as sample size grows, and is the main problem with traditional GP emulators when many fast evaluations of the likelihood and prediction are needed. Thus, we seek fast emulators as alternatives to a GP.

All four candidate methods (TGP, CSC, LAGP, GPC) are implemented using a Bayesian framework, though not all are fully Bayesian since plug-in MLE estimates for model parameters are used. The unknown parameters are estimated analytically for certain choices of prior distributions, or via MCMC. In general, having assumed the unknown process of interest to be a GP, this makes the GP the prior probability distribution over functions. The observed response data are then a single observation of the GP and “combined” with the prior via the usual Bayesian framework to obtain a posterior distribution of all the unknown parameters required by the model. Some of the methods indicate that likelihood-based approaches can be used to estimate the model parameters if desired (e.g. Kaufman et al. (2011)).

### 3.2 Bayesian Treed Gaussian Process Models (TGP)

Gramacy and Lee (2008) propose a model that aims to address different levels of variability in the response. Such “bumpy” surfaces are often modelled with nonstationary GP models, which are difficult to fit and more crucially are even more computationally intractable than a stationary GP model. The Bayesian treed Gaussian process (TGP) model proposed by Gramacy and Lee (2008) can be seen as a further generalisation of the work done by Chipman et al. (2002), who generalised a Bayesian classification and regression tree to create the Bayesian treed linear model by fitting linear models at the tree’s terminal nodes. A TGP model has two main components: Bayesian treed partitioning, and independent GP models at the terminal nodes of the tree.
Firstly as the model name suggests, Bayesian treed partitioning is done on the input space by fitting a treed model. A *treed model* is composed of a binary tree that partitions the input space into disjoint subsets and a parametric model for the response $Y|x$ for each subset of the partition (or equivalently, for each terminal node of the tree) Chipman et al. (2002).

Secondly, stationary GP models with linear mean functions are independently fit at each of the terminal nodes of the tree. That is, each GP model is fit on a subset of the data and has the form

$$Y(x) \sim GP \left( x^T \beta, K(\cdot) \right).$$

This means that the “larger” the tree, i.e. the more terminal nodes there are, the more time is saved by using a TGP model over a single GP.

The parameters for each GP model are estimated while a tree stochastically undergoes the tree-building process. While any valid covariance function can be used for $K(\cdot)$, we use the separable squared exponential covariance function shown in Equation 3.1.8. Estimating a TGP’s model parameters uses a mix of Gibbs sampling, Metropolis-Hastings sampling, reversible jump MCMC (RJ-MCMC), and numerical integration, with the specifics depending on the choice of prior distributions. We use the default priors in the tgp package (Gramacy). The tree-building itself is done via Metropolis-Hastings sampling, following the setup by Chipman et al. (2002).

Note that a distinction between the algorithm used by Gramacy and Lee (2008) and that described by Chipman et al. (2002) is that in the tree-building process, in addition to having grow, prune, swap, and change steps, a rotate step is introduced (see Figure 2 in Gramacy and Lee, 2008). This additional step is shown to provide a more diverse set of candidate nodes for pruning, thus better exploring the model space, which in turn results in better mixing of the Markov chain. The tree with highest posterior probability is known as the *maximum a posteriori* (MAP) tree.

Given the observed data, the predicted value of $y(x)$ and its uncertainty can be found via the kriging mean and variance formulas in 3.1.9, where the specific GP used is that obtained from the MAP tree model. Alternatively, we sample from the posterior distribution of trees based on their posterior probabilities can be used for prediction. This results in a smoother predictive surface and incorporates the uncertainty that comes from tree and parameter estimation.

In summary, Gramacy and Lee (2008) proposed the TGP model suitable for responses with different levels of variabilities throughout the surface by using flexible GP models at the terminal nodes of a treed partition on the input space. When the tree has many terminal nodes, the computation time of a TGP model is lower than that when fitting a single GP model to the entire dataset, making the TGP model a candidate for our choice of emulation model.
3.3 GP Models Using Compactly Supported Correlation Functions (CSC)

The method proposed by Kaufman et al. (2011), which we abbreviate as CSC, is in principle the same as that of a single traditional GP except with two main differences: the choice of covariance function and form of mean function.

The correlation function used is compactly supported. Using a compactly supported correlation function introduces sparsity into the covariance/correlation matrix. This allows for computationally efficient sparse matrix techniques to be used, when finding the determinant or inverse of the correlation matrix when calculating the likelihood or for prediction, as described in Section 3.1. Specifically, we use a product of truncated power functions suggested by Kaufman et al. (2011) to get an anisotropic correlation function.

The second key aspect of the CSC model is that the mean function is not the usual constant mean. Instead, the mean function in a CSC model is a linear combination of basis functions, which of course is more flexible than a simple linear combination of the inputs. As opposed to the common practice of using a constant mean for the GP, using regression functions is motivated by having a linear combination of the basis functions \( f_i \) capture large-scale variation in the response surface. Intuition then suggests that the GP should have shorter range correlation, which can justify using correlation functions with compact support. That is, the response has the following structure:

\[
Y(x) \sim GP \left( \sum_{j=1}^{k} f_j(x) \beta_j, K(\cdot) \right),
\]

where \( k \) is a tuning parameter, and \( f_1, ..., f_k \) are fixed regression functions given \( k \), \( \beta_1, ..., \beta_k \) are unknown regression coefficients, and \( K(\cdot) = \sigma^2 R(\cdot) \) is the GP’s covariance function. Indeed, Kaufman et al. (2011) show that using a richer mean structure leads to improved predictive performance compared to only using compactly supported covariance.

In summary, for this project \( K(\cdot) \) is a product of truncated power functions and the set of basis functions \( f_i \) are the tensor products of Legendre polynomials. As with the TGP model, the choice of covariance and basis functions, and all tuning parameters are set at “default” values from their respective packages or papers unless their performance in Chapter 4 is dreadful predictively.

3.4 Locally Approximate Gaussian Process Models (LAGP)

Gramacy and Apley (2015) propose the locally approximate Gaussian process (LAGP) model, which is a very natural approach for fast computation. A GP model is fit given a neighbourhood of training points around location where prediction is to be made, and prediction can proceed as usual via the kriging formula Equation 3.1.9. The main features
of the LAGP model are as follows: a GP is fit independently for each point where prediction is to be made, which means local neighbourhoods of points must be found to train each GP. Different criteria with various advantages and disadvantages can be used to find the local neighbourhoods. We also discuss why the LAGP model is fast compared to the traditional GP model.

The LAGP model “combines” the initial model-fitting step with prediction in the sense that a neighbourhood of points, or a *subdesign*, is found given a new location/input $x$. A GP with a constant mean and covariance function $K(\cdot)$ is trained using the subdesign. The subdesign size is a fidelity parameter because its value determines the emulator accuracy and computational expense.

It may seem that the additional step of having to find a local subdesign for each new point would not be any faster than the traditional full GP model, but in fact fast computational speed is the main advantage of using an LAGP model. Firstly for each new input, the associated subdesign and thus GP model are calculated independently of one another, making parallelisation trivial. The process of finding the subdesign is also simplified by taking a suboptimal greedy approach. That is, the subdesigns are found sequentially in the sense that points are added to the subdesign one at a time such that the specific point added depends on all those added before it. The tradeoff for this greedy approach is that the optimal subdesign may not be found, but the gains in computational speed for a generally small reduction in predictive performance can be substantial for large datasets. Of course, reducing the neighbourhood size directly lowers computation time.

As for finding the subdesign for an input, having fixed a local design size (the number of points in the subdesign), the greedy algorithm sequentially adds points that minimise some criterion to the subdesign until termination. Though any criterion can be used, we focus on Euclidean distance and average reduction in global variance, which Gramacy and Apley (2015) abbreviated as *NN* and *ALC* respectively. When the criterion for a point to be added to the subdesign is to minimise the Euclidean distance to the input we wish to predict, the LAGP model is equivalent to a nearest neighbour (NN) approach that has been studied in previous work (e.g. Stein et al. (2004)). However, Euclidean distance is generally not a recommended measure unless we truly believe that correlation structure is locally isotropic. Using the ALC criterion means that each point added to the subdesign is that which maximises the average reduction in global variance. The ALC criterion used by Gramacy and Apley (2015) is an approximation of the computationally expensive *active learning Cohn*. Despite ALC being a faster approximation, finding the subdesign using the NN criterion is the faster of the two. Thus, when fast approximate predictions are the goal, using the simple NN approach may be adequate.

Lastly, Gramacy and Apley (2015) use Bayes inference that results in the distribution of the response given the data and parameters to be analytic. Specifically, at each step of the subdesign-building process, the GP parameters are estimated via MLE. Given the
subdesign thus far and the estimates of the GP parameters, the response follows a Student-\(t\) distribution with degrees of freedom equal to the subdesign size. Having an analytic solution to the distribution of the response is useful because there is no need to rely on sampling to calculate the mean and variance throughout the subdesign-building process.

LAGP is expected to work well when a process is at least “locally” stationary because it only fits a GP locally to the new input. Thus as with TGP models, LAGP can accommodate problems where the response surface has different levels of variability. In fact, one advantage of the LAGP model is that we can control the computation time required via the size of the neighbourhood and choice of subdesign criterion. In this way a user may choose how they wish to balance predictive accuracy and speed more directly than with a TGP model.

### 3.5 Generalised Polynomial Chaos Models (GPC)

Chakraborty et al. (2017) replace the GP in a typical emulation model with a basis function expansion. This makes GPC the most “different” of the four methods. That is, the response is modelled as follows:

\[
Y (\mathbf{x}) \sim N \left( \beta_{k0} + \sum_{j=1}^{k} f_{kj} (\mathbf{x}) \beta_{kj} \sigma^2 \right),
\]

where \(k\) is the unknown number of basis functions, \(f_{k1}, ..., f_{kk}\) are the \(k\) basis functions, and \(\beta_{k0}, \beta_{k1}, ..., \beta_{kk}\) are the \(k + 1\) unknown regression coefficients. In other words for \(j \leq k\), \(\beta_{kj}\) is the coefficient of the \(j\)th basis function conditioned on the model having \(k\) basis functions.

The response appears to have the same mean structure as that of CSC. However, the key difference is that \(k\) is fixed in CSC whereas in GPC it is treated as an unknown parameter. Further, CSC still assumes that the response is GP thus requiring a covariance function, as opposed to GPC, which models the unaccounted variability as simple white noise.

As with the previous three methods, Chakraborty et al. (2017) implement GPC using a fully Bayesian setup. Basis function expansions themselves has already been done, but with GPC both the sparsity and expansion order are controlled by the data since \(k\) is estimated based on the data using RJ-MCMC. That is, the user prespecifies an oversized set of basis functions (such as a set of Legendre polynomials) and chooses bounds on the maximum order of main effects, interaction effects etc. The emulation model itself is then a linear combination of some subset of \(k\) basis functions and the regression coefficients are estimated.

Having chosen a set of basis functions, the remaining tuning parameters are as follows:

- \(k_{\text{max}}\): The maximum number of basis functions allowed in the model \((k \leq k_{\text{max}})\).
- \(r\): The maximum order of any basis function \(f_{kj}\) from a single variable \((r \leq t)\).
- \(s\): The maximum number of variables in an interaction \((s \leq d)\).
- \(t\): The maximum total order of any basis function \(f_{kj}\) \((rs \leq t)\).
By using reversible jump MCMC, each step of the MCMC may add, delete or modify basis functions to the emulation model at the current step. This means that the user need not specify how many terms should be included in the model, which is the approach taken by past work. Chakraborty et al. (2017) argue that allowing the data to determine sparsity in addition to expansion order is advantageous because model uncertainty is better represented. Uncertainty quantification is then more complete when using model averaging for prediction. That is, prediction at a new input is done as usual with an average of predictions from a sample of models drawn from the posterior distribution.

Using a linear mean structure as opposed to the covariance matrix used by GPs is computationally advantageous. The former requires only the inversion of a $k \times k$ matrix ($k$ is the number of terms currently in the model), as opposed to an $n \times n$ matrix inversion ($n$ is the number of training points).

That is, in addition to the parameters mentioned in the paper regarding maximum order, interaction order, etc. unlike the BTGP and LAGP models, parameters for the prior distributions used by the MCMC may also need to be tuned for competitive performance. This can result in the need to tune approximately ten parameters, which can be highly time-consuming. Nevertheless, if tuning is not particularly difficult for a particular dataset, GPC can be one of the fastest methods available for model emulation.
Chapter 4

Comparison

In this chapter, the various large dataset emulation methods are compared. All simulations are done in R, using packages \texttt{tgp}, \texttt{SparseEm}, and \texttt{laGP} for the TGP, CSC, and LAGP models respectively, and supplementary code from the original articles for GPC and GPC (R Core Team (2016); Gramacy; Kaufman et al. (2011); Gramacy (2018)).

The four emulation methods TGP, CSC, LAGP, and GPC, are compared by studying their performance on four numerical models. All input spaces are scaled to be on $[0,1]^d$. The responses are all univariate and scaled via with the mean and standard deviation of the training sets.

The first dataset consists of draws from a GP model with a three-dimensional input, to have zero mean, and a squared-exponential covariance function and length-scale parameters 0.1 in all dimensions. A \textit{nugget} is added for numerical reasons. That is, a small number, in this case $1 \times 10^{-8}$ is added to the diagonal of the covariance matrix. A nugget is often used when attempting to invert a near singular covariance matrix. The second response function models the water flow through a borehole. It is has eight inputs but is considered to be simple and is widely used as a benchmark for testing computer experiment methods (e.g. Morris et al. (1993); Gramacy and Apley (2015), Kaufman et al. (2011); etc.). The function used in Dette and Pepelyshev (2010) also has an eight-dimensional input and and exhibits a fair amount of curvature in several input dimensions. This makes the function useful for comparing computer experiment designs. Lastly, the Franke function has a two-dimensional input, and has two Gaussian peaks of different heights, as well as a smaller dip. It is used by Franke (1979) as a test function in interpolation problems.

The response functions are shown in Chapter 4. These models were chosen to present a wide range of response surfaces, while still being reasonably smooth allowing for GP-based models to be sensible candidates. Of the four functions, the GP, Dette, and Franke response functions all have input spaces on a unit hypercube: $[0,1]^d$. The original domain of the Borehole function is as follows: $r_w \in [0.05,0.15], r \in [100,50000], T_u \in [63070,115600], H_u \in [990,4000], T_l \in [63.1,116], H_l \in [700,820], L \in [1120,1680], K_w \in [9855,12045]$. The Borehole function’s input space was scaled to be on $[0,1]^8$ before emulation.
Table 4.0.1: Response functions used for simulation studies. The GP, Dette, and Franke response functions all have input spaces on a unit hypercube: $[0, 1]^d$. The original domain of the Borehole function is as follows: $r_w \in [0.05, 0.15], r \in [100, 50000], T_u \in [63070, 115600], H_u \in [990, 4000], T_l \in [63.1, 116], H_l \in [700, 820], L \in [1120, 1680], K_w \in [9855, 12045]$. The Borehole function’s input space was scaled to be on $[0, 1]^8$ before emulation.

For each response function, training sets of 100, 200, 500, and 1000 were simulated. Inputs were chosen based random latin hypercube designs, which is common for computer experiments (e.g. Sacks et al., 1989; McKay et al., 1979). Test sets of size 1000 were also randomly simulated for each response function using random latin hypercube designs. All sets of responses were scaled to have zero mean and standard deviation of one. Models were fit using the training data with each emulation method, and predictions of the response were computed sequentially. That is, predictions are not made in parallel of one another. Details about model training and tuning are given in Section 4.1.

The emulation approaches are then compared via the following metrics:

1. The average time to predict one new input.
2. The mean-squared prediction error (MSPE), which is a measure of overall predictive performance, relative to the variation in the test set.
3. The coverage probability in the validation set.

The entire simulation process, including generating new random latin hypercube designs, was repeated five times.

4.1 Training and Testing Procedure

Tuning for all emulation models and dataset combinations was done minimally and to the extent that the results must not appear to be obviously wrong. That is, the authors’ default settings were used when they existed. Similarly, settings for prediction were kept at default or were chosen to minimise computation time while preserving the predictive performance.
**TGP**

**Training:** Default package settings were used. The covariance function is from the separable power family with fixed $p_0 = 2$. Though Gramacy and Lee (2008) recommends restarting the MCMC chain multiple times, this did not seem necessary for any of the test functions.

**Prediction:** Though the default is to use the kriging mean and variance estimates from the MAP tree, sampling from the posterior distribution of trees was done in order to incorporate parameter uncertainty into the mean and variance estimates. This is also sensible when there are multiple trees with similarly high posterior probability. For each new input, a posterior sample of size 50 was used. This number was chosen since coarse testing suggests that at least for the test functions in question, larger posterior sample sizes does not significantly change mean or variance estimates.

**CSC**

**Training:** Default package settings were used. The set of basis functions is a set of Legendre polynomials where all the main effects up to order two are included, as well as interactions in two dimensions. The sparsity is known as the proportion of off-diagonal entries in the covariance matrix that are zero, and is set at 0.99.

**Prediction:** A sample of $n_{post} = 150$ based on every tenth $\tau$ in the MCMC chain (with the burn-in removed) is used as a sample from the posterior. The predicted mean and variance are averages of the $n_{post}$ results from applying kriging formulas to each posterior sample.

**LAGP**

**Training:** Default package settings were used for the subdesign/neighborhood size ($N = 50$) and the criterion used for calculating subdesigns (ALC). Following Gramacy and Apley (2015), the isotropic squared exponential covariance function is used. LAGP is run with $N = 50$ and criterion NN to obtain an MLE estimate of the length-scale parameter. This estimate is used as a starting point for length-scale parameter estimation during the “main run” of LAGP using ALC.

**Prediction:** The kriging mean and variance formulas 3.1.9 are used.

**GPC**

**Training:** For most datasets, GPC requires tuning of at least the four main parameters $k_{max}, r, s, t$ to obtain sensible results. In some cases, further tuning of the hyperparameters for the priors used in the MCMC are required.

For the response functions tested, this additional tuning was required when training with the Franke function. Here, $\sigma_a, \sigma_b$ are hyperparameters for the prior of $\sigma^2$. 
<table>
<thead>
<tr>
<th></th>
<th>$k_{\text{max}}$</th>
<th>$r$</th>
<th>$s$</th>
<th>$t$</th>
<th>$\sigma_a$</th>
<th>$\sigma_b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>GP</td>
<td>100</td>
<td>4</td>
<td>3</td>
<td>12</td>
<td>2.1</td>
<td>2.2</td>
</tr>
<tr>
<td>Borehole</td>
<td>40</td>
<td>6</td>
<td>2</td>
<td>12</td>
<td>2.1</td>
<td>2.2</td>
</tr>
<tr>
<td>Dette</td>
<td>50</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>2.1</td>
<td>2.2</td>
</tr>
<tr>
<td>Franke</td>
<td>80</td>
<td>10</td>
<td>2</td>
<td>20</td>
<td>20</td>
<td>22</td>
</tr>
</tbody>
</table>

Table 4.1.1: Tuned parameters for GPC.

**Prediction:** A sample of $n_{\text{post}} = 50$ draws from the posterior distribution are used to estimate the mean and variance of the prediction. The estimates include the addition of the posterior estimates of $\sigma^2$ as opposed to leaving it out as this is analagous to the difference between variance of a prediction versus the variance of the mean.

### 4.2 Results

All computation was done on the Simon Fraser University colony cluster, where each simulation was run as a separate job with a single processor on a single node, and two GB of memory. The figures in this section compare the four candidate emulation methods by plotting a metric against the training set size on the x-axis for each of the datasets from the four response functions.
Figure 4.2.1: Predictive performance based on a test set size of 1000 against the four training set sizes (100, 200, 500, 1000). The points are the results from each simulation while the lines are plotted using the average results across the five simulations. The upper left plot in Figure 4.2.1 displays the results relevant to when the response function is GP, and the decreasing lines show that the predictive performance of all four emulation methods improves as the training set size increases. In this case, the LAGP model dominates the remaining three for all training set sizes. It appears that the gain in predictive performance is most substantial when the training set size increases from 100 to 200 points. The remaining plots show that LAGP performs well for all the response functions, except for the Dette function.
Starting with the upper left plot in Figure 4.2.1, when the response function is GP, the predictive performance of all four emulation methods improves as the training set size increases. In this case, the LAGP model dominates the remaining three for all training set sizes. It also appears that the gain in predictive performance is most substantial when the training set size increases from 100 to 200 points. Looking at the remaining plots in the Figure 4.2.1, LAGP performs well for all the response functions, except for the Dette function. Generally for all four plots in Figure 4.2.1, the predictive performance improves with larger training set size with two exceptions: (1) CSC does not perform well with either the GP or Franke dataset from the start, and does not improve as the training set size increases. This suggests that the default complexity of the mean structure is not high enough to capture the variability well. (2) GPC’s performance with the Borehole dataset appears to worsen when the training set size increases from 100 to 200 and somewhat from 500 to 1000. This is likely due to some parameters being improperly tuned.

There is no clear winner for all the datasets, though the results for LAGP and TGP seem to have robust performance even without any tuning at all.
Figure 4.2.2: Empirical coverage probability based on a test set size of 1000 against the four training set sizes (100, 200, 500, 1000). The nominal value is 0.95. The points are the results from each simulation while the lines are plotted using the average results across the five simulations. In the upper left plot, the estimated credible intervals from CSC, GPC and LAGP model tend to overcover for all training set sizes. In contrast the results from TGP show that undercoverage not only persists, but gets worse as training set size increases. Similar observations can be made of the remaining three plots. The CSC model tends to have coverage probabilities closest to the nominal value of 0.95 for all response functions and training set sizes. The LAGP model overcovers for all response functions except for the Dette.
Figure 4.2.2 shows the empirical coverage probabilities of the validation sets. In the upper left plot, the estimated credible intervals from CSC, GPC and LAGP model tend to overcover for all training set sizes. In contrast the results from TGP show that undercoverage not only persists, but gets worse as training set size increases. Similar observations can be made of the remaining three plots. Notably, the CSC model tends to have coverage probabilities closest to the nominal value of 0.95 for all response functions and training set sizes. The LAGP model overcovers for all response functions except for the Dette. Recall that this is also the one response function where the LAGP model’s predictive performance appears poor. These results suggest that estimated credible intervals from TGP models are likely to be too small, and those from LAGP models may also undercover.
Figure 4.2.3: Average model training time against the four training set sizes (100, 200, 500, 1000). The points are the results from each simulation while the lines are plotted using the average results across the five simulations. All four plots appear very similar. The key point overall is that the TGP model scales poorly with training set size. When the training set has 1000 points, it can take up to 5-6 hours to train a TGP model.

Minimising the model training time is not important for modularised calibration, which is discussed in Section 5.1, because training the emulation model can be done in advance instead of during the calibration process. However for practical reasons, we check how the model training time scales with the training set size in Figure 4.2.3. All four plots in
Figure 4.2.3 appear very similar. The key point overall is that the TGP model scales poorly with training set size. When the training set has 1000 points, it can take up to 5-6 hours to train a TGP model. While such training times are not considered long in practice, they are excessive for this project because multiple emulation models will need to be trained and compared.
Figure 4.2.4: Average prediction time for a single new input against the four training set sizes (100, 200, 500, 1000). The points are the results from each simulation while the lines are plotted using the average results across the five simulations. The average prediction time for a single new input increases as the training set size increases. All four plots are similar in that we immediately see that both the TGP and CSC models' prediction times do not scale well with the size of the training set. When the training set has 1000 points, it takes well above 30 minutes to predict a new input, regardless of the response function in question.

Though the model training time is only important for the execution of this project, minimising the average prediction time for a new input is vital for fast model calibration.
Figure 4.2.4 shows the average prediction time for a single new input increases with the training set size. All four plots are similar in that we immediately see that both the TGP and CSC models’ prediction times do not scale well with the size of the training set. When the training set has 1000 points, it takes well above 30 minutes to predict a new input, regardless of the response function in question. Such prediction times are prohibitively high.

Consider that the \textit{tgp} package does not provide a way to specify the number of terminal nodes to be large. This means that TGP suffers in terms of speed when the response surface is highly stationary because very little partitioning is done. This in turn means that a GP is fit over nearly the entire training set. It is immediately clear that both training (Figure 4.2.3) and prediction time (Figure 4.2.4) scale very poorly when using TGP. Even though a so-called “wiggly?” function was purposely chosen with the idea that partitioning may be favoured, the vast majority of trained models show that the MAP trees have either zero or only a single split. The result is that both the training and prediction time suffer compared to the other methods, even when sampling only 50 times from the posterior distribution.

Clearly, Figure 4.2.1 showed that performance of CSC seems poor compared to the other methods for some of the datasets. However the parameters used for training all the models are not properly tuned. An alternative to the default model settings would be to use a polynomial order of four or five, which Kaufman et al. (2011) suggest may work decently for most applications. Increasing the order and hence set of basis functions of course means an increased computational burden due to the additional regression coefficients that must be estimated, while not impacting prediction time. The results therefore show that for the purposes of this project, CSC’s prediction time does not scale well enough. In fact, tuning even the two parameters of maximum polynomial degree and maximum interaction order is time-consuming given that the training time for CSC does scales up quite rapidly as well. For problems where prediction time is not of primary importance, CSC has potential especially considering its performance with completely untuned parameters.

In summary, thus far we eliminate the TGP model from further consideration due to its prediction time. Despite having the best empirical coverage probabilities overall, CSC is also eliminated due to its prediction time. GPC and LAGP both have similar performance and have training and predictions times so fast that they necessitate separate plots from TGP and CSC. The following plots compare the remaining two candidate emulation methods.
Figure 4.2.5: Comparing GPC and LAGP’s average prediction time for a single new input against the four training set sizes (100, 200, 500, 1000). The points are the results from each simulation while the lines are plotted using the average results across the five simulations. These plots are a repeat of Figure 4.2.4, focussed on the GPC and LAGP models’ average prediction time. For all four plots, the LAGP model’s prediction time increases approximately linearly with the training set size. In contrast, the GPC model’s prediction time does not always increase so uniformly as training set size increases. However when we examine the scale on the y-axis for all four plots, both the GPC and LAGP models can be said to have very low prediction times.
Figure 4.2.5 is a repeat of Figure 4.2.4, focussed on the GPC and LAGP models’ average prediction time. For all four plots, the LAGP model’s prediction time increases approximately linearly with the training set size. In contrast, the GPC model’s prediction time does not always increase so uniformly as training set size increases. However when we examine the scale on the y-axis for all four plots, both the GPC and LAGP models can be said to have very low prediction times.

Consider that though GPC has the most tuning parameters (not including those directly related to MCMC), we can see in all four plots from Figure 4.2.3 that the training speed is among the fastest and scales reasonably well. Following the guidelines in Chakraborty et al. (2017) and through trial and error, tuning a model to the point where it can perform decently is not overly arduous after having done so a couple of times. However as mentioned in the simulation procedures, tuning the hyperparameters of the prior is necessary, which of course is time-consuming. Juggling up to 10 parameters can also be difficult. Secondly, the tuning process cannot be easily automated. This is because several plots, including the histograms of the posterior distribution of the number of basis functions $k$ and $\sigma^2$, are examined to determine whether the parameters used are suitable.

### 4.3 Recommendations

In terms of training and prediction speed, the GPC models are the winners. GPC performs well according to nearly all of the metrics discussed, across all the datasets examined. However, tuning can become quite involved in the sense that it is a process that is not easily automated given the need to examine multiple histograms and metrics. The training speed is the fastest and so this may not be an issue for many problems, but more intensive tuning of the hyperparameters for the prior distributions and MCMC itself may be required, resulting in over ten parameters that require tuning. In fact of all four methods, GPC requires the most “effort” to produce sensible results. That is, while tuning may improve the performance of the remaining three methods, they work “out-of-the-box”.

On the other hand, the LAGP models also have excellent performance in terms of training and prediction speed given the y-axis labels in Figure 4.2.5. LAGP also clearly outperforms GPC in terms of predictive performance for three out of the four datasets and has similar results for coverage probability. That LAGP seems to have problematic coverage for the Dette dataset as training set size increases may be an anomaly particular to this dataset as it is also the one dataset where GPC outperforms LAGP. This may be due to the lack of local stationarity for this response function.

Thus, LAGP is the recommended emulation method in general because of its excellent overall performance, and is an approach that requires very little effort to attain good results since it only has two main parameters: the size of the subdesigns and the criterion used to find the subdesign. In fact, many pairs of values result in good predictive performance and
often juggling the parameters are only for the purposes of finding those that minimise computational time. For the purposes of this project, the vastly simplified tuning is particularly beneficial because for data with multivariate output, more than one emulation models are required. The use of multiple independent models for modelling multivariate output is explained in Section 6.1.
Chapter 5
Model Calibration

Calibration is an important application of computer models (e.g. Kennedy and O’Hagan, 2001; Higdon et al., 2004). It involves combining simulator outputs with physical observations from the field to build a predictive model, which is a type of regression problem. Model calibration also aims to estimate unknown calibration parameters, \( u \), which are inputs that are not measurable or adjustable in the physical system, but govern the behaviour of the computer model, by solving a kind of inverse problem. The calibration parameters can be seen as missing covariates. Inputs that are not calibration parameters are either measurable or adjustable. Kennedy and O’Hagan (2001) and Higdon et al. (2004) can be considered seminal papers for model calibration. Much work has been built on top of this framework, such as a method to integrate expert judgment by Reese et al. (2004), and alternative ways to combine output from multiple experiments with field data by Qian and Wu (2008) and Goh et al. (2013).

Suppose the true calibration parameter values are \( u^* \). Overloading the notation used in Section 3.1 somewhat, let \( Y \) represent the mean of the physical process. Define \( Y_M(x, u) \) to be computer simulator output and \( Y_F(x) \) to be field observations. Let \( n_M \) and \( n_F \) be the number of computer simulation outputs and field observations available respectively.

Let \( \varepsilon(x) \) be measurement error, resulting in the following model for the field data

\[
Y_F(x) = Y(x, u^*) + \varepsilon(x).
\] (5.0.1)

The computer model is assumed to be deterministic. Since the computer model does not perfectly model the physical process, the following statistical model relating the computer output to reality includes an additive bias term:

\[
Y(x, u^*) = Y_M(x, u^*) + b(x, u^*).
\] (5.0.2)

This hierarchical model can be “collapsed” down to the following form:

\[
Y_F(x, u^*) = Y_M(x, u^*) + b(x, u^*) + \varepsilon_F(x).
\] (5.0.3)
Immediately, identifiability problems are expected and discussed in Loeppky et al. (2006) and briefly in Kennedy and O’Hagan (2001). However in this project we will not be concerned with the bias term \( b = 0 \) and subsequently not be plagued by lack of identifiability arising for this reason.

Chapter 3 covered model emulation, specifically using GPs as a statistical surrogate for computer output. For the problem of interest in this project, using a fast model emulator simplifies the calibration process because this means that we are not encumbered by slow/limited runs from the computer simulator. Fitting a separate model for the simulator output is a form of modularisation, which is discussed in detail by Liu et al. (2009).

Using a modularised calibration framework means that the simulator output and bias are modelled independently of one another. Though Liu et al. (2009) justify modularisation in various ways, a practical motivation for modularisation is computational tractability (e.g. Jacob et al., 2017). That is, when \( n_M >> n_F \), meaning that the number of simulator runs vastly outstrips the number of field observations, including the field observations in the emulator is unlikely to impact the emulator in any positive way. In fact Gramacy et al. (2015) argue that joint inference with the bias only makes diagnosing the emulator for future improvement more difficult.

For this project, we focus on establishing that using LAGP as the fast model emulator, modularisation, and mesh adaptive direct search to perform model calibration results in adequate parameter estimation. Most importantly, the computational time is dramatically reduced compared to the commonly used method established by Kennedy and O’Hagan (2001). After an overview of the calibration procedure and setup, a demonstration is presented with two of the response functions previously used to compare emulation methods.

5.1 Methodology and procedure

Having chosen LAGP as the fast emulator of choice based on the results from Chapter 4, calibration is done following the thrifty approach used by Gramacy et al. (2015). In contrast to the usual Kennedy and O’Hagan (2001) formulation for calibration, Gramacy et al. (2015) treats calibration as an optimisation problem.

The “thriftiness” comes from using LAGP model emulation with a modularised calibration framework (Liu et al., 2009). Instead of using a full GP, which is often crippled by its computational speed, using the LAGP model as the emulator is in contrast very fast as demonstrated in this project. Modularisation further improves speed because fitting the emulator \( \hat{Y}_M \) is done independently from the bias and only the simulator outputs are used to build the emulator. For the purposes of demonstration, field data are simulated by adding noise on top of simulator data. Lastly, the optimisation problem is solved using mesh adaptive direct search to find the calibration parameter values that produce the highest likelihood. The result of Gramacy et al. (2015) treating calibration as an optimisation
problem means that all the evaluations of the objective function done during the search also provides distributional information in place of posterior sampling, which is more computationally expensive.

Throughout the remainder of this project, we assume that there is no systematic disagreement between the computer model and reality, resulting in

$$Y_F (x, u^*) = Y_M (x, u^*) + \varepsilon_F (x),$$

(5.1.1)

where the surrogate emulator model $\hat{Y}_M$ is an LAGP model used in place of the full GP. This replacement allows for a large $n_M$, in other words a large number of “computer model” runs.

For both examples in the following section, as well as the datasets used for the motivating problem, field data are not available. Field data are thus generated by adding white noise to computer model runs. Indeed, since all the calibration runs in this project are setup such that $n_M >> n_F$, the small number of field data points does not affect the emulator meaningfully in terms of performance, which is the main motivation of modularised calibration.

Let the field data be $D^F_{n_F} = (X^F_{n_F}, Y^F_{n_F})$, where $X^F_{n_F}$ is a design matrix of $n_F$ inputs, and $Y^F_{n_F}$ is the corresponding responses. The $n_M$ computer model data are represented by $D^M_{n_M} = \left( \left[ X^M_{n_M}, U_{n_M} \right], Y^M_{n_M} \right)$, where the columns of $X^M_{n_M}$ and $U_{n_M}$ together is the computer experiment design matrix.

The computer model data $D^M_{n_M}$ is used to train an emulator $\hat{y}^M (\cdot, u)$. The emulator is used to obtain $\hat{Y}^M_{n_F}|u = \hat{y}^M \left( X^F_{n_F}, u \right)$, which are the predictions at the $X_F$ locations given a setting of the calibration parameter(s) $u$.

Since LAGP is the emulation model, we have the fidelity parameter $N$, which is the local sub-design size. The choice of $N$ controls the trade-off between computation time and faithfulness of the emulator compared to a full GP. Recall that for all $j \in \{1, 2, ..., n_F\}$ each $\hat{y}^M_{j|u}$ is predicted independently and in parallel to one another. Each $j$-th predicted value has an associated local sub-design $X_N \left( x^F_j, u \right) \subset \left[ X^M_{n_M}, U_{n_M} \right]$ and locally estimated length-scale parameter $\hat{\theta}_j = \hat{\theta} \left( D_N \left( x^F_j, u \right) \right)$.

Having obtained $Y^M_{n_F}|u$, define the fitted discrepancies as $\hat{Y}^B_{n_F} = \hat{Y}^F_{n_F} - \hat{Y}^M_{n_F}|u$. When the response is scalar, $\hat{Y}^B_{n_F}|u$ is a vector of length $n_F$. Since $n_F$ is typically small, the discrepancy can be modelled using a full GP regression model using the data $D^B_{n_F} (u) = \left( X^F_{n_F}, Y^B_{n_F}|u \right)$. Note that since we assume that there is no systematic disagreement between the computer model and reality, the discrepancy model in this project is in fact purely an estimator for error term $\varepsilon_F (x)$, which is modelled as i.i.d. Gaussian error. In other words, the likelihood associated with $\left( \hat{\theta}_j, D^B_{n_F} (u) \right)$ is simply the density for an i.i.d. normal distribution with zero mean. This is equivalent to simply averaging over $\sigma^2_{\varepsilon}$, where they assume $\varepsilon$ is i.i.d. $N \left( 0, \sigma^2_{\varepsilon} \right)$. 

35
The calibration objective function is the posterior probability of $u$, which is equal to the likelihood associated with $(\hat{\theta}_j, D_{n_F}^B(u))$ described above multiplied by the prior probability of $u$. We seek the $u$ setting(s) that maximise the posterior probability of observing $Y_{n_F}^F$. Using independent local design searches for $\hat{y}_{n_F}^M|u$ causes the objective to appear “noisy”. It follows that a derivative-free optimisation approach is a better choice than that with derivatives. Gramacy et al. (2015) use an implementation of the mesh adaptive search algorithm called NOMAD. In short, inputs are tested on the objective function on a sequence of meshes, which converge to a local optima under weak regularity conditions. Since the search algorithm is a local solver, the initialisation points are those $u$-values evaluated on a random space-filling design.

Notice that since the LAGP model is the emulator choice and the discrepancies are modelled with a full GP model, as long as the sub-design size $N$ and number of field observations $n_F$ are not too large, calibration is expected to be extremely fast even with large number of computer model runs $n_M$. The calibration procedure uses functions from the laGP package.

**Steps for evaluating the calibration process**

1. Set up the computer model function $M$.
2. Choose the priors for each of the calibration parameters.
3. Simulate $n_F$ field data points by adding white noise to $n_F$ outputs generated from $M$.
4. Generate $n_M$ computer model runs using a Latin hypercube design over the input space, along with ten-fold replicated field design inputs jointly. Note that $n_M$ can be large and quickly generated by using a fast emulator to model the shot data in the motivating problem.
5. Evaluate the objective function, which calculates the posterior probability of $u$, on a space-filling design of calibration values that is slightly smaller than the input space. The design is a sequential D-optimal design for a stationary GP.
6. Use mesh adaptive search on the best input(s) found on the grid from the previous step. Various settings can be changed for a more or less thorough search that can be chosen based on needs in performance and/or computation time. The predicted calibration parameters are the values that maximise the log-likelihood.

This procedure, which is for problems where there is no bias term, is demonstrated in the following section using the Franke and Dette response functions from Chapter 4 before we return to the motivating problem.

### 5.2 Examples

All results for the remainder of this project was computed on a computer with a 2.3 GHz Intel Core i5 processor and 8 GB memory. Recall the Franke and Dette response functions
from Chapter 4. For both response functions, the number of computer model and field data runs are set as \( n_M = 10000 \) and \( n_F = 50 \) respectively.

The Franke function has a 2-dimensional input space and the Dette function has an 8-dimensional input space. To demonstrate the effectiveness of the calibration procedure described in the previous section, we specify one input of the Franke function \( (u_1 = x_2) \), and five inputs of the Dette function \( (u_1, u_2, u_3, u_4, u_5) = (x_4, x_5, x_6, x_7, x_8) \) to be calibration parameters. The “true” calibration parameter values are all set as the 75th percent quantile of the \( \text{Beta}(2, 2) \) distribution as this is the choice of prior for all the parameters as well. That is, the calibration procedure was done with the input spaces scaled to a unit hypercube, with all the plots below returned to the original scale of the response functions.

**Figure 5.2.1:** Density estimate of the calibrated parameter for the Franke function. The true value is the vertical line. The change in colour of the points from dark blue to green to yellow illustrates the sequence of input settings assessed during the optimisation process. The posterior mode looks close to the true parameter value.

Starting with the easier problem of calibrating a single parameter given a 2-dimensional input space, Figure 5.2.1 shows that the posterior mode looks close to the true parameter value. The change in colour of the points in Figure 5.2.1 from dark blue to green to yellow illustrates the sequence of input settings assessed during the optimisation process.

Since calibration is treated as an optimisation problem, the procedure results in only the values of the posterior probability for the values of \( u \) calculated during the search instead of samples drawn from the posterior distribution. When there is a single calibration parameter, we can easily plot the 1-dimensional density estimate. As the Dette function has
five calibration parameters, visualising a 5-dimensional density is unfeasible. Instead we show the sets of calibration values searched during the optimisation process in Figure 5.2.2.

![Dette: Values searched during optimisation](image)

Figure 5.2.2: Progression of the values searched for the five calibration parameters for the Dette function. Apart from the fourth parameter, as the iterations proceed, the calibration settings searched tend toward the true values. The true values are the horizontal black line. Though the fourth parameter appears to be far from the truth, re-running the calibration procedure many times shows that estimates can be either close or far away. This can occur when there are local maxima that cause the optimisation algorithm to land on different settings as the solution.

In the plot on the far left of Figure 5.2.2 for \( u_1 \), we can see that the change in colour of the lines from dark blue to green to yellow illustrates the sequence of \( u_1 \) values tested during optimisation. Figure 5.2.2 shows that as the iterations increase, the calibration settings chosen for all five parameters tend to gather about the true values, represented by the horizontal lines. Note that though the fourth parameter appears to be far from the truth, re-running the calibration procedure many times shows that estimates can be either close or far away. This can occur when there are local maxima that cause the optimisation algorithm to land on different settings as the solution.

Despite one of the estimated parameters being rather far off from the truth in this particular run, when predictive performance is the goal of calibration, as is often the case, the calibration procedure overall can still be seen as successful for the Dette function. Figure 5.2.3 shows a way of assessing the calibration procedure by running the process 100 times, then using the calibrated parameters to predict on a test set of 1000 inputs. For both
response functions, the predictive performance of the calibrated parameters is excellent as not a single run estimated parameters that result in models leaving more than 0.5% of variation unexplained.

![Boxplot](image.png)

Figure 5.2.3: Predictive performance of the calibrated parameters for both the Franke and Dette functions based on 100 runs of calibration. In each run, the calibrated parameters are used to predict a test set of 1000 points. For both response functions, the predictive performance of the calibrated parameters is excellent as not a single run estimated parameters that result in models leaving more than 0.5% of variation unexplained.

The last metric of interest is of course the time required for calibration. Figure 5.2.4 shows the time taken for the 100 runs of the calibration procedure. Clearly computation time is not at all a problem for the Franke function, with all the runs taking well under five minutes each.
Figure 5.2.4: Time taken to calibrate the parameter(s) for both the Franke and Dette functions based on 100 runs of calibration. Computation time is not at all a problem for the Franke function, with all the runs taking well under five minutes each. Taking an average time of below 20 minutes to calibrate five parameters for the Dette function is also reasonable.

That it takes longer to calibrate five parameters than one partly explains the longer time taken to calibrate the Dette function. Computation time can be decreased by modifying some of the mesh adaptive search settings when calibrating the Franke function’s parameters. These settings were changed to allow for a less thorough search of the input space when initial tests showed that intensive search of the Franke function’s input space is unnecessary for good results. Further, recall in Section 4.2 that LAGP performed the worst predictively when using the Dette response function compared to the other three. It is thus reasonable if calibration via LAGP is more time-consuming for a function poorly-modelled by LAGP than one that is represented well.

For both examples, the runtime is reduced by using the nearest neighbour criterion for the LAGP models and setting the length-scale parameters at a fixed value. By using the nearest neighbour criterion, and not requiring the estimation of the length-scale parameter, this will result in LAGP models that run much faster compared to those using the default settings, i.e. ALC criterion, and MLE estimation of the length-scale. The reasoning behind these settings for the LAGP models originates from Section 3.2 of Gramacy (2018), and will be explained further in Section 6.2.
These examples demonstrate that modularised calibration using LAGP and mesh adaptive search can produce good results in terms of estimates and predictive performance of calibration parameters. For the purposes of this project, that adequate results can be attained in a much shorter amount of time compared to traditional calibration methods makes this calibration procedure ideal for the main goal of speeding up the experimental science workflow.
Chapter 6

Return to the materials science problem

In this chapter, we put together the results of Chapter 3 to Chapter 5 and apply them to a material science application. Recall from Section 2.2 that the data from our motivating application is from combining simulations and velocimetry measurements from three plate impact experiments. This results in three datasets, each with a 7-dimensional input space and 4-dimensional response. See Figure 2.2.1 for a plot of the four points of interest in a typical velocimetry profile taken to be the response vector. After describing a method for modelling data with multivariate output, we then investigate the performance of LAGP and find that it works well as the emulator of choice for the three experiments. Lastly, multivariate calibration adapted from the methodology described in Section 5.1 is demonstrated. The results from emulation and calibration are also briefly compared and discussed with those from Walters et al. (2018).

6.1 Modelling data with multivariate output

Up to now, only univariate responses have been considered. A common way to model data with multivariate responses when the observations are over time is to simply treat time as an input (e.g. McFarland et al. (2008)). Bayarri et al. (2007) handle multivariate outcomes by using a wavelet decomposition, then using GP emulators to model the wavelet coefficients. Another approach is to use a model based on the singular value decomposition (SVD) used in Higdon et al. (2008), which we found to perform better than “converting” the problem to have a scalar output by indexing time when modelling our data from dynamic shock experiments. Thus in this section, we only present the method and results of using a SVD-based model.

Consider a simulator with \( d \)-dimensional inputs with a \( t \)--dimensional output \( y(x) \). As before, suppose we have \( n \) training observations, a design matrix \( X \) \((n \times d)\) as defined in 3.1.1, and let \( Y = Y(X) = [Y(x_1), ..., Y(x_n)]^T \) be the \( n \times t \) matrix of observed responses.
The singular value decomposition (SVD) on $Y$ is

$$Y_{n \times t} = U_{n \times k} D_{k \times k} V^T_{k \times t}, \quad (6.1.1)$$

where $k = \min(n, t)$.

Let $p \leq k$ be the number of “important” singular values. Choosing $p$ can be done empirically by the cumulative percentage criterion

$$p = \min \left\{ m : \sum_{i=1}^{m} \frac{d_i}{\sum_{i=1}^{k} d_i} > \gamma \right\}, \quad (6.1.2)$$

where $\gamma$ is some pre-determined threshold of explained variation. For example, Jolliffe (2002) suggest that $\gamma = 0.9$. In practice, the choice of $p$ may be guided by other requirements such as computational issues, etc.

The SVD-based GP model used by Higdon et al. (2008) assumes that

$$y(x) = \sum_{i=1}^{p} b_i w_i(x) + \varepsilon, \quad (6.1.3)$$

where $\varepsilon$ is an error term and $b_1, ..., b_p$ are orthogonal basis vectors. That is, $B = UD = [b_1, ..., b_k]$. The model assumes that each $w_i(x)$ are independent GPs ($i \in \{1, ..., p\}$). The $n$-dimensional vectors $v_i$ are thus viewed as a realisation of the GP $w_i(x)$. In other words, following SVD on $Y$ and choosing $p$, we may assume any kind of GP model for the $w_i(x)$’s as desired, including any of those discussed in Chapter 3.

### 6.2 Emulation

In this application for each of the three shots there are $n = 1000$ observations. Each response matrix $Y$ is $1000 \times 4$, and design matrix $X$ is $1000 \times 7$. An SVD was applied to $Y$ and $k = 3$, accounting for a total explained variation of at least $\gamma = 0.99$. This means that each shot’s emulation model requires fitting three independent GP models, all of which are LAGP models. Thus, the next problem is to determine suitable settings for the LAGP models: the subdesign size and subdesign selection criterion.

At the end of Section 5.2, we note that using the nearest neighbour criterion reduces the computational speed of LAGP greatly compared to using the ALC criterion defined by Gramacy and Apley (2015). However as discussed in Section 3.4, the nearest neighbour criterion is not sensible unless we believe that the length-scale (correlation) parameters for all the input variables are one. If we knew the true values of the length-scale parameters,
then scaling the input space by these parameters should render nearest neighbours a “sensible” criterion for subdesign selection.

In both examples presented in Section 5.2, the input spaces are scaled by maximum likelihood estimates (MLE) of the length-scale parameters. These estimates are from the \texttt{mleGPsep} function from the \texttt{lagp} package, which uses analytic derivatives of the log-likelihood with the built-in \texttt{optim} function. Thus, we define the process of obtaining MLE estimates of length-scale parameters and scaling the input space as \textit{stretching and compressing (SC)} the input space.

We extend the idea of SC an input space from the case where the response is scalar to the current problem where the response is multivariate by estimating a separate set of length-scale parameters for each of the three models that make up an emulator. Each model’s response is the corresponding singular vectors with the same set of input variables.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure6.2.1.png}
\caption{Maximum likelihood estimates of the length-scale parameter for each input and “sub-model”, for all three shots.}
\end{figure}

In Figure 6.2.1, the right-most plot shows the length-scale parameter estimates for shot 106 for each of the three independent models that make up the emulator, each represented by a different shape. Notably, $x_m$ has very different length-scale estimates depending on which basis vector is the response. In contrast, input $c$ for shot 104 on the far left plot has very similar length-scale estimates. Overall we can see that for a given shot, the length-scale estimates can be quite different for each of the three models. This suggests that for each model, different inputs are more “important” than others i.e. those with smaller estimated length-scales.

Next, recall that the \texttt{laGP} package does not support using a separable correlation function. Scaling the input space for each model with the length-scales in order to make
choosing nearest neighbours as the subdesign selection criterion also has an additional benefit. That is, it follows that fixing the global length-scale parameter at 1 for the LAGP model is reasonable. Though we do not expect the length-scale estimates to be perfect, they may be close enough such that there is no need to estimate a global length-scale parameter. Indeed, in later results we find that this is the case when modelling all three experiments. After testing many versions of LAGP models, we narrow down the combinations of settings to two particular sets shown in Table 6.2.1.

<table>
<thead>
<tr>
<th>Settings (1)</th>
<th>Settings (2)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Subdesign size</strong></td>
<td><strong>Input space</strong></td>
</tr>
<tr>
<td>50</td>
<td>SC not applied</td>
</tr>
<tr>
<td>50</td>
<td>SC applied</td>
</tr>
</tbody>
</table>

Table 6.2.1: Two sets of settings for emulation of the three datasets with multivariate output.

Table 6.2.1 shows the difference between two sets of settings used for emulation of the data from each of the three shots. In the first row, the subdesign size is 50 for both settings because testing revealed that increasing the size up to 200 did not significantly improve the predictive performance of the emulation models, while increasing computation time. Decreasing the subdesign size did cause the performance to suffer. Next, the choice of whether SC is applied on the input space determines the options in the last two rows, as discussed earlier in this section.
Table 6.2.2: Results of using LAGP models for all three experiments when the training set is 800 simulations and the test set is the remaining 200 simulations. The subdesign size is 50. All numbers are on the original scale of the data. For all shots, stretching and compressing the input space (SC) for all the models that make up the emulator produces better performance predictively and can lead to improvement in the coverage probability as well. For all shots, the predictive performance, shown in the rows with $MSPE/Var(y_{test})$, is better when using Settings (2) than Settings (1).
Table 6.2.2 shows that for all the shots, an emulator built applying SC on the input space and the nearest neighbour (NN) criterion for all three of the required GP models (Settings (2)) has better performance than an emulator built without applying SC and using the ALC criterion (Settings (1)). The results are all on the original scale of the data and are from fitting the shots with the two types of settings when a training set of 800 simulations and test set of 200 simulations are used.

In the first row of Table 6.2.2, we can see an example where for $v_1$ of shot 104, the predictive performance is improved by more than four times. Increasing the subdesign size up to 200 does not improve the predictive performance enough to justify the increase in computation time. The estimated standard deviations and width of the 95% credible intervals remain similar between the two sets of settings. Though the estimated errors are very small, they are being underestimated rather severely for some responses, particularly $v_1, v_2, v_3$ for shot 105 since the 95% empirical coverage probability is very far from the nominal value. Note that the coverage probability nearly always improves when comparing the results of Settings (2) to Settings (1), with the most dramatic being $v_4$ for shot 104 where the coverage probability jumps from 0.39 to 0.78.

Having found that Settings (2) is the clear winner over Settings (1), we assess the LAGP under Settings (2) more thoroughly by using leave-one-out cross-validation (LOOCV).

<table>
<thead>
<tr>
<th>104S</th>
<th>$v_1$</th>
<th>$v_2$</th>
<th>$v_3$</th>
<th>$v_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSPE/Var(ytest)</td>
<td>0.017</td>
<td>0.018</td>
<td>0.038</td>
<td>0.030</td>
</tr>
<tr>
<td>SD</td>
<td>0.0007</td>
<td>0.0006</td>
<td>0.0001</td>
<td>0.0001</td>
</tr>
<tr>
<td>95% CI width</td>
<td>0.0023</td>
<td>0.0021</td>
<td>0.0005</td>
<td>0.0002</td>
</tr>
<tr>
<td>95% coverage probability</td>
<td>0.92</td>
<td>0.87</td>
<td>0.86</td>
<td>0.67</td>
</tr>
<tr>
<td>105S</td>
<td>$v_1$</td>
<td>$v_2$</td>
<td>$v_3$</td>
<td>$v_4$</td>
</tr>
<tr>
<td>MSPE/Var(ytest)</td>
<td>0.054</td>
<td>0.062</td>
<td>0.138</td>
<td>0.060</td>
</tr>
<tr>
<td>SD</td>
<td>0.00053</td>
<td>0.00044</td>
<td>0.00019</td>
<td>0.00055</td>
</tr>
<tr>
<td>95% CI width</td>
<td>0.00178</td>
<td>0.00145</td>
<td>0.00065</td>
<td>0.00214</td>
</tr>
<tr>
<td>95% coverage probability</td>
<td>0.52</td>
<td>0.43</td>
<td>0.48</td>
<td>0.99</td>
</tr>
<tr>
<td>106S</td>
<td>$v_1$</td>
<td>$v_2$</td>
<td>$v_3$</td>
<td>$v_4$</td>
</tr>
<tr>
<td>MSPE/Var(ytest)</td>
<td>0.047</td>
<td>0.075</td>
<td>0.109</td>
<td>0.023</td>
</tr>
<tr>
<td>SD</td>
<td>0.00108</td>
<td>0.00105</td>
<td>0.00037</td>
<td>0.00022</td>
</tr>
<tr>
<td>95% CI width</td>
<td>0.00395</td>
<td>0.00389</td>
<td>0.00143</td>
<td>0.00081</td>
</tr>
<tr>
<td>95% coverage probability</td>
<td>0.94</td>
<td>0.90</td>
<td>0.81</td>
<td>0.83</td>
</tr>
</tbody>
</table>

Table 6.2.3: Results of LOOCV when using LAGP models and Settings (2) (subdesign size of 50; apply SC to the input space; find the subdesign with the NN criterion; fix the global length-scale parameter as 1) for all three experiments. For all models the input space have been stretched and compressed using MLE estimates of the length-scale parameters.
Comparing the three MSPE/Var(ytest) rows in Table 6.2.3 to the blue rows with the same labels in Table 6.2.2 shows that the holdout sets of 200 simulations are fairly representative of the respective full datasets in the sense that the predictive performance is not very different. Using LOOCV to assess the performance also allows for comparison with the results in section IV A of Walters et al. (2018).

Figure 6.2.2: Comparison of the LOOCV predictions versus the true simulation value for each of the four response velocities and three shots altogether on a single plot. The velocities are on the original scale of the data and multiplied by 10000 for the sake of comparison with Walters et al. (2018). Most of the points lie fairly close to the diagonal, meaning that the emulation model has decent performance.

Figure 6.2.2 plots the estimates from all three emulation model against the true computer model response. Most of the points lie fairly close to the diagonal, meaning that the emulation model has decent performance.
Figure 6.2.3: Comparison of the LOOCV predictions versus the true simulation value for each of the four response velocities and three shots. The velocities are on the original scale of the data and multiplied by 10000 for the sake of comparison with Walters et al. (2018). The plots in the two columns on the right show that the $v_3$ and $v_4$ are much easier to predict accurately because all the points lie closer to the diagonal, compared to the left-most two columns showing the results for $v_1$ and $v_2$. We can see that for both shots 105 and 106, there are many more points farther from the diagonal, indicating that predicting $v_1$ and $v_2$ is difficult for these shots.

Figure 6.2.3 is the same Figure 6.2.2 in that it plots the estimates from all three emulation model against the true computer model response, though separated into separate panels depending on the shot and specific response velocity. In the two columns on the right, these plots show that the $v_3$ and $v_4$ are much easier to predict accurately because all the points lie closer to the diagonal, compared to the left-most two columns showing the results for $v_1$ and $v_2$. We can see that for both shots 105 and 106, there are many more points farther from the diagonal, indicating that predicting $v_1$ and $v_2$ is difficult for these shots.

Figure 6.2.2 can be directly compared with Figure 6 in Walters et al. (2018). Though the plots look similar, our thrifter emulator as expected does not perform quite as well as the traditional full GP emulator. Walters et al. (2018) reports the performance of their
emulator in terms of the root mean-squared error (RMSE) from predictions via LOOCV for all 12 features, referring to the four points of interest along each of the three shot curves. They find that the RMSE for all 12 features over all 1000 simulations is 5.5 m/s, with the best and worst predicted points having RMSE values of 1.8 m/s and 8.6 m/s respectively. In comparison, the LAGP emulator with settings (2) has an overall RMSE of 8.28 m/s, with the best and worst predicted points having RMSE values of $4.6 \times 10^{-4}$ m/s and 99 m/s respectively.

That is, the overall performance is very comparable overall, but our emulator has a much wider range in predictive performance in that it can predict extremely poorly as well as extremely accurately. With this in mind, we can see in Figure 6.2.2 and Figure 6.2.3 that for simulation velocities below approximately 150 m/s and above 400 m/s, seem well predicted than those between 150 and 400 m/s. Note that in both Figure 6.2.2 and Figure 6.2.3, the velocities are on the original scale of the data and multiplied by 10000 in order to compare directly with the units used in Walters et al. (2018).

When closely examining the features/velocities and simulations that were predicted extremely poorly by LAGP, we find that the bad performance is due to the model predicting simulations whose responses do not have a neighbourhood of similar responses. For example, the first ($v_1$) and second feature ($v_2$) for the 747th simulation in shot 106 are predicted very badly. Figure 6.2.4 shows the simulations included in the subdesign for all three LAGP models that make up the emulator for shot 106. The true simulation is visibly far away from any of the simulations in the neighbourhood found by LAGP. Thus, it is unsurprising that the model could not cope when there is extrapolation of this form.
Figure 6.2.4: Typical example of a simulation with features predicted poorly by LAGP. The neighbourhood of simulations shown are those that were chosen to be in the neighbourhoods of all three independent models that contribute to the overall emulator. The responses of the neighbourhood found by LAGP (in red) are all quite far from the true response (in blue). Unsurprisingly, the predicted response (in green) is not close the truth.

Figure 6.2.4 is a representative example of a simulation that is poorly predicted by the emulator. The plot shows that the responses of the neighbourhood found by LAGP (in red) are all quite far from the true response (in blue). Unsurprisingly, the predicted response (in green) is not close the truth.

Note that under Settings (2), changing the subdesign size up to 200 and repeating LOOCV for all three shots results in an overall RMSE of 7.36 m/s, with the best and worst predicted points having RMSE values of $3.9 \times 10^{-4}$ m/s and 65 m/s respectively. While there is an improvement in performance, the increase in computation does not make raising the size of the subdesign worthwhile. An alternative to the SVD-based model approach for multivariate response was tested. That is, we build an independent LAGP model for each of the four response velocities, thereby changing the functional response into four separate scalar response problems. The predictive performance of this approach is poor compared to the SVD-model based approach.

Though using the LAGP model for the SVD-based approach suffers when predicting some of the simulations, it has very comparable performance overall to the full GP model used by Walters et al. (2018). One strong advantage to the LAGP emulator is that as shown
in Section 4.2, the computation time scales very well as the dataset size grows. Thus, the final chosen emulator for each shot is that constructed by three independent LAGP models, each using Settings (2) from Table 6.2.1.

6.3 Calibration using multivariate output

Since the goal is to calibrate as quickly as possible, we extend the fast calibration method for scalar response by Gramacy et al. (2015). Recall that the method is described in Section 5.1 and subsequently demonstrated in Section 5.2. After extending this framework for multivariate response, we show its results using the three datasets in our application problem.

6.3.1 Methodology and procedure

The changes made to the framework by Gramacy et al. (2015) in order to render it suitable for data with multivariate output are both rather natural and reasonable. The modifications to Section 5.1 are as follows:

1. The emulator \( \hat{Y}_M \) is an SVD-based model, as described in Section 6.1, where each independent component model is an LAGP model.

2. As the fitted discrepancies \( \hat{Y}_{n_F} = \hat{Y}_{n_F}^F - \hat{Y}_{n_F}^M \) is a matrix where with \( n_F \) rows and \( \text{dim}(Y_M) \) columns, the error model is again changed to an SVD-based model in place of a single full GP regression model. That is, each independent component model is a full GP model.

3. The likelihood \( \hat{Y}_{n_F}^B \)'s model is found by serialising the \( p \) \( n_F \)-tuples obtained from the \( p \) component GPs of the error model. That is, we put together the estimates from the \( p \) GP models as the matrix \( \hat{W}^B_{n_F} \), where

\[
\hat{W}^B_{n_F} = \begin{bmatrix}
\hat{w}_{11}^B & \cdots & \hat{w}_{1p}^B \\
\vdots & \ddots & \vdots \\
\hat{w}_{n_F1}^B & \cdots & \hat{w}_{n_Fp}^B
\end{bmatrix} = \begin{bmatrix}
\hat{w}_1^B & \cdots & \hat{w}_p^B
\end{bmatrix}
\]
then serialise these estimates as follows:

\[
\begin{pmatrix}
\hat{w}^{B|u}_1 \\
\vdots \\
\hat{w}^{B|u}_p
\end{pmatrix} = \begin{pmatrix}
\hat{w}^{B|u}_1 \\
\vdots \\
\hat{w}^{B|u}_{n_F}
\end{pmatrix} = \sim N(0, K). \tag{6.3.2}
\]

Since \(\hat{w}^{B|u}_i\) and \(\hat{w}^{B|u}_j\) are independent for all \(i, j \in \{1, ..., p\}\), the covariance matrix \(K\) is a block diagonal matrix:

\[
K = \begin{bmatrix}
K\left(\hat{w}^{B|u}_1, \left(\hat{w}^{B|u}_1\right)'; \sigma^2, l\right) & 0_{n_F \times n_F} & 0_{n_F \times n_F} \\
0_{n_F \times n_F} & \ddots & 0_{n_F \times n_F} \\
0_{n_F \times n_F} & 0_{n_F \times n_F} & K\left(\hat{w}^{B|u}_p, \left(\hat{w}^{B|u}_p\right)'; \sigma^2, l\right)
\end{bmatrix}. \tag{6.3.3}
\]

We set \(K(\cdot)\) as the isotropic squared-exponential covariance function (see Equation 3.1.7), specifically we have

\[
K(w, w'; \sigma^2, l) = \sigma^2 \exp\left\{-\frac{1}{2l^2}|w - w'|^2\right\}. \tag{6.3.4}
\]

Both \(l\) and \(\sigma^2\) are fixed and discussed below.

In the context of the motivating problem, the surrogate emulator \(\hat{Y}_M\) has the same settings as those for the emulator for the computer model function \(\hat{M}\): length-scales are fixed at 1, the sub-design size is 50. The second modification has rather more details of note. Note that the number of singular values chosen for the measurement error model \(p\) is set as the same as that for the emulator model, in our case \(p = 3\) works well. Tests show that using an identity correlation matrix with zero mean for each of the constituent GPs for the error model produced very poor results in terms of both estimating the calibration parameters as well as prediction on a test set using the estimated parameters. Instead, using a squared exponential correlation function where \(l = \sigma^2 = 1\), in other words a global fixed length-scale of 1 (and a nugget of \(1 \times 10^{-8}\) for numerical stability) results in decent performance.

Fixing the length-scales at some value also seems reasonable because of the use of modularisation. In the usual Bayesian calibration scheme, estimation of the length-scale parameter(s) is constrained by additional data and unknown parameters. However with modularisation the error model is fit separately, considering only the fitted error values, which means that if the length-scales are not fixed and instead estimated, we would assume
that the estimated length-scales would be such that the GP models would overfit the errors. As the calibration parameters are estimated by optimising the likelihood from the discrepancy model, we would not expect overfitting to lead to good estimates.

Another vital aspect of using this calibration procedure with the three datasets in question is that careful scaling can be the difference between decent and horrific performance. That is, recall that the results in Section 6.2 recommend that the LAGP emulator models use data where the input spaces are “stretched and compressed” using MLE-estimates of the length-scale parameters for each input. Testing showed that setting the calibration parameters on a unit hypercube, and then performing SC internally as a step of the emulator models, instead of having the calibration parameters start on the SC space leads to very different results. The reasons for this are likely technical as opposed to theoretical. Consider that when the SC space is very different from a unit hypercube, finding a set of priors for the calibration parameters that works well is difficult. Finding the settings for the optimisation step where mesh adaptive search is done also requires more effort.

With these modifications and details in mind, we discuss the results of putting together our fast emulation model from Section 6.2 with this thrifty calibration scheme applied to the three datasets from the motivating problem in the following section.

6.3.2 Results

Following the calibration procedure described in the previous section, all the shots used identical settings, including those for mesh adaptive search. The number of field data runs is set at $n_F = 50$, there are $n_M = 10000$ computer model runs.

For all shots, the Johnson-Cook model parameters are set as the calibration parameters. Thus, there are five calibration parameters: $u = (u_1, u_2, u_3, u_4, u_5) = (a, b, c, x_m, x_n)$. The “true” calibration parameter values are all set as the 75th percent quantile of the $\text{Beta}(2, 2)$ distribution as this is the choice of prior for all the parameters as well. That is, the calibration procedure was done with the input spaces scaled to a unit hypercube for reasons described in the previous section, though all the plots below are on the original scale of the response functions.
Figure 6.3.1: Progression of the values searched for the five calibration parameters from shots 104, 105, and 106. The true value for each parameter is the 75th quantile of the $Beta(2, 2)$ distribution scaled to the original input space (black lines). For shot 104 (first row), apart from the fourth parameter $x_m$, as the iterations proceed, the calibration settings searched tend toward the true values. Similar observations apply for shots 105 and 106, except parameters $b$ and $c$ respectively are the ones not well predicted.
Figure 6.3.1 shows the sets of \( u \) values searched throughout the optimisation process for shots 104, 105, and 106 respectively in each row. As the iterations proceed, represented by the colours changing from dark navy blue to yellow, overall the calibration parameter estimates tend to choose settings close to the true values. Each shot has one of the parameter estimates landing relatively further away from the truth. In contrast, some parameters are estimated very well, with \( a \) and \( x_n \) both having estimates extremely close to the true value for all the shots. Despite some parameters appearing to be far from the truth, re-running the calibration procedure 100 times shows that estimates can both over- and under-estimates. This can occur when there are local maxima that cause the optimisation algorithm to land on different settings as the solution. That is, the specific runs chosen for Figure 6.3.1 are those that appear representative of the 100 runs.

Figure 6.3.2: Predictive performance of the calibrated parameters for each shot based on 100 runs of calibration. In each run, the calibrated parameters are used to predict a test set of 1000 points. For all shots, the predictive performance of the calibrated parameters is reasonable for all the response velocities on average, with only shot 106 having estimated parameters that result in models leaving more than 4% of variation for responses \( v_1 \) and \( v_2 \) unexplained.

As with the calibration example using the Dette function in Section 5.2, despite some of the parameters being somewhat far off from the truth for each shot, when predictive performance is the goal of calibration, as is often the case, the calibration procedure overall can still be seen as adequate. Figure 6.3.2 assesses the calibration procedure from a predictive performance standpoint by running the process 100 times, then using the
calibrated parameters to predict on a test set of 1000 inputs generated from the computer model. Looking at the right-most plot in, Figure 6.3.2 we can see that there are a handful of runs that have relatively poor predictive performance, such as $v_1$ and $v_2$ in shot 106. On the other hand, the results for shot 104 are excellent for all four response velocities. Attempts to improve the performance by having the mesh adaptive search to be more thorough, raising the number of computer model runs, and changing the priors does not significantly improve these results. However for all shots, the predictive performance of the calibrated parameters is reasonable for all the responses on average, with only shot 106 having estimated parameters that result in models leaving more than 4% of variation for responses $v_1$ and $v_2$ unexplained.

Figure 6.3.3: Time taken to calibrate the parameter(s) for each shot based on 100 runs of calibration. Looking at the range of times for each shot, the plot shows that on average a single run of calibration takes under 30 seconds for any shot.

The major advantage gained by using a thrifty emulator together with this specific calibration procedure is the computation time. Figure 6.3.3 shows the time taken for the 100 runs of the calibration procedure for each shot. Looking at the range of times for each shot, we find that on average a single run of calibration takes under 30 seconds for any shot. These results are particularly outstanding considering that the computation was done on a laptop with modest specifications.
Lastly we compare the calibration estimates and time taken using the methodology in this project, which we will call LAGP calibration for short, with that used in Walters et al. (2018). Walters et al. (2018) used the calibration framework proposed by Higdon et al. (2008), which uses a full GP model as the model emulator. For this comparison, we calibrate each shot individually and set all seven inputs for each shot as calibration parameters, which is in accordance with the application problem itself. Note that we chose to calibrate five inputs in this chapter so that a test set can be generated based on those inputs set as non-calibration parameters. The same three datasets are used for both approaches, meaning that there are $n_F = 1$ field data runs, and $n_M = 1000$ computer model simulations for each shot. We use the $Beta(2, 2)$ distribution as the prior for all calibration parameters, whereas Walters et al. (2018) use a uniform prior based on the domain of the input space. We run the MATLAB code used by Walters et al. (2018) for calibration on the same laptop used throughout Chapters 5 and 6 (2.3 GHz Intel Core i5 processor and 8 GB memory).
Figure 6.3.4: The posterior distributions for calibrations of shot 104, 105, and 106 individually from Walters et al. (2018), each using 10000 MCMC samples. The black lines are the calibration parameter estimates obtained using an LAGP emulator and the fast calibration method proposed by Gramacy et al. (2015) adapted for multivariate response.
In Figure 6.3.4, each row of plots shows the posterior distributions for the calibration parameters for each of the three shots. Each of the calibration runs used 10000 MCMC samples. The black lines are the calibration parameter estimates obtained using LAGP calibration. Looking at the first plot on the left in the first row, we can see that the estimate from LAGP calibration is not quite in line with the posterior mode for \( a \). However in Figure 7 of Walters et al. (2018), the posterior mode for \( a \) also does not agree with the estimate by Gray et al. (1994). There are other plots in Figure 6.3.4 with similar difference in posterior mode and estimate from LAGP calibration. In contrast, some results from both calibration methods agree, such as \( G_2 \) from the shot 105 second row and \( a \), and \( x_m \) from shot 106 in the last row.

In terms of computation time, calibrating three shots using the method from Walters et al. (2018) took 5.68, 5.68, and 5.64 hours for shot 104, 105, and 106 respectively. In contrast, LAGP calibration took 6.64, 4.26, and 4.67 seconds respectively. Consider also that Walters et al. (2018) use 250000 MCMC samples. We use 10000 samples since the goal is to demonstrate the vast difference in computational time between the two calibration methods.

Overall, we demonstrate that there is a dramatic reduction in computation time when using modularised calibration using LAGP and mesh adaptive search. As shown in this section when specifying five our of the seven inputs as calibration parameters, we also show that the predictive performance is decent.
Chapter 7

Discussion

This project starts by comparing the performance of fast statistical emulators. While all the tested candidates are related to the traditionally slow GP in some way, the LAGP and GPC models are the clear winners, particularly when speed is important. Even though the GPC model is faster than an LAGP model and appears to scale better as training set size grows, we recommend LAGP because it requires less tuning for specific datasets.

We then demonstrate the calibration framework proposed by Gramacy et al. (2015), where the key differences to traditional calibration by Kennedy and O’Hagan (2001) are the use of modularisation (by modelling the computer model data separately via an emulator), and the treatment of calibration as an optimisation problem. When the emulator is itself also fast even with large experiments, the result is fast calibration that can easily handle the large datasets that cripple the full GP model used for calibration (Kennedy and O’Hagan, 2001).

The fast calibration approach described by Gramacy et al. (2015) and implemented in the laGP package was designed for scalar responses. Here, the approach is modified to apply to functional data. The efficacy of this framework is shown via three datasets that combine simulations and velocimetry measurements from plate impact experiments. We also show that emulation using LAGP models for all three experiments is improved in terms of predictive accuracy and coverage probability by “stretching and compressing” the input spaces using MLE estimates of the length-scale parameters. Given the results and dramatic reduction in computation time demonstrated in Section 6.3.2, it is reasonable to assume that the emulation and calibration methodology demonstrated will scale well with the data size.

The next major step after assessing these procedures with more datasets and biased calibration would be to develop a framework for “real-time” sequential experimental design. That is as new data enters the model, scientists are given the input(s) that are predicted to maximise the reduction in estimated calibration parameter variance, or some other criterion of choice. Sacks et al. (1989) and indeed the ALC criterion defined by Gramacy and Apley (2015) are both good starting points. Real-time analysis of data during time-constrained/expensive experiments allows for strategic and efficient decision-making,
and would complete the acceleration of the experimental science workflow shown in Figure 2.2.2. Ultimately, this results in more significant and meaningful scientific discoveries at equal or lower cost. Fast emulation and calibration are key for this goal and has been achieved.
Ahrens, J.


Boteler, J. M. and D. P. Dandekar


Chipman, H. A., E. I. George, and R. E. McCulloch

Dette, H. and A. Pepelyshev

Diggle, P. and P. J. Ribeiro Jr.

Franke, R.

Goh, J., D. Bingham, J. P. Holloway, M. J. Grosskopf, C. C. Kuranz, and E. Rutter

Golchi, S., D. R. Bingham, H. Chipman, and D. A. Campbell
Gramacy, R. B.

tgp: an R package for Bayesian nonstationary, semiparametric nonlinear regression and design by treed Gaussian process models.

Gramacy, R. B.

2018. lqGP: Large-Scale Spatial Modeling via Local Approximate Gaussian Processes in R.

Gramacy, R. B. and D. W. Apley


Gramacy, R. B. and H. K. H. Lee


Gramacy, R. B. and H. Lian


Grimmett, G. and D. Stirzaker


Higdon, D., J. Gattiker, B. Williams, and M. Rightley


Higdon, D., M. Kennedy, J. Cavendish, J. Cafeo, and R. Ryne


Jacob, P. E., L. M. Murray, C. C. Holmes, and C. P. Robert


Jolliffe, I. T.


Jones, D. R., M. Schonlau, and W. J. Welch


Kaufman, C. G., D. Bingham, S. Habib, K. Heitmann, and J. A. Frieman

Kennedy, M. C. and A. O’Hagan

Liu, F., M. J. Bayarri, and J. O. Berger

Loeppky, J., D. Bingham, and W. Welch

McFarland, J., S. Mahadevan, V. Romero, and L. Swiler

McKay, M. D., R. J. Beckman, and W. J. Conover

Morris, M. D., T. J. Mitchell, and D. Ylvisaker

Qian, P. Z. G. and C. F. J. Wu

R Core Team

Rasmussen, C. E. and C. K. I. Williams

Reese, C. S., A. G. Wilson, M. Hamada, H. F. Martz, and K. J. Ryan

Sacks, J., W. J. Welch, T. J. Mitchell, and H. P. Wynn

Savitsky, T., M. Vannucci, and N. Sha

Shang, L. and A. B. Chan
Stein, M. L., Z. Chi, and L. J. Welty
2004. Approximating likelihoods for large spatial data sets. *Journal of the Royal

Walters, D. J., A. Biswas, E. C. Lawrence, D. C. Francom, D. J. Luscher, D. A. Fredenburg,
2018. Bayesian calibration of strength parameters using hydrocode simulations of
symmetric impact shock experiments of Al-5083. *Journal of Applied Physics*,
124(20):205105.

Williams, C. L., K. T. Ramesh, and D. P. Dandekar