Uncertainty Quantification for Stochastic Simulators with Application to Offshore Wind Farms

Jack Callum Kennedy

Thesis submitted for the degree of
Doctor of Philosophy

School of Mathematics, Statistics & Physics
Newcastle University
Newcastle upon Tyne
United Kingdom

December 2022

Declaration

I declare this thesis is my own reasearch and unless explicity stated, the writing, research and code used to implement methods is my own.

The Athena model was developed at the University of Strathclyde, apart from the modifications required to implement a spare parts strategy in section 2.1.1, which are my own.

Parts of Chapters 4 and 5 have been published in the Journal of the Royal Statistical Society, Series C (Applied Statistics) Kennedy *et al.* (2023). This paper was co-authored by my supervisors, Kevin Wilson and Daniel Henderson, however the research and novel ideas are my own.

This thesis has not been submitted for examination at another university.

Acknowledgements

There are lots of people I am grateful for. Here are some select highlights.

First of all, I'd like to acknowledge the EPSRC for making this entire thing financially viable. Spending over 20 years in formal education is simultaneously a badge of honour but also a source of crippling pain. Because of you, EPSRC, I've been able to enjoy the experience of not paying council tax until my mid-to-late twenties. The downside of this arrangement is my grandparents have been asking, with minimal optimism, "Jack, are you *ever* going to get a job?".

A massive thank you to my PhD supervisors, Kevin Wilson and Daniel Henderson. Without the pair of you this thesis would be littered with typos are *even more* of my bad jokes. Kevin, I've enjoyed our many, excessively long conversations about the way statistics should be done. I'm not sure if anyone else really agrees with us. Daniel, thank you for keeping me *and Kevin* on track.

To everyone who has held a desk in PhD 4, it's been good (apart from chaotic bit where going in the office was banned). It would be unjust if I did not recognise Jack as a door sign aficionado and the world's most effective rubber duck.

I'd also like to thank my friends in the Royal Statistical Society, especially the Young Statisticians, for reminding me that statistics is something I actually enjoy, even when this thesis has convinced me otherwise.

My friends outside of statistics should also be mentioned. I am grateful that you have pretended to be interested in my thesis, doing a PhD becomes an annoying personally trait which is hard to shake off. Jack, Tom, I have always assumed that "massive nerd" is a term of endearment.

Finally, Aimée, thank you for everything you have done for me. Putting up with me being in a right strop on many occasions for reasons such as my code crashing for the fifth time in as many hours, or because reviewer 2 has been particularly unforgiving. Without your love and support this would have been much more difficult.

Abstract

Computationally expensive computer models, known as simulators, are fundamental to the modern scientific process. In recent years, there has been an increased interest in *stochastic simulators* as analysts aim to model uncertainty in their predictions. Stochastic simulators are found in all areas of science. We focus on a particular simulator, known as the Athena simulator, which is used to assess the performance of offshore wind farms in their early life.

Athena is typical of many stochastic simulators in that it can take a long time to run and the variance exhibited by the simulator may depend on the simulator input. This poses multiple computational challenges for subjective Bayesians who wish to perform computer-intensive tasks such as sensitivity analysis or decision support. A common solution to the large computational cost of such analyses is to construct a fast statistical surrogate model, known as an *emulator*, to produce an efficient approximation to the simulator with an appropriate quantification of uncertainty. Two popular emulators for stochastic simulators, HomGP, the homoscedastic Gaussian process, and HetGP, the heteroscedastic Gaussian process.

The first contribution of this thesis is the proposition of a method, based upon Kennedy & O'Hagan (2000), which seeks to exploit fast approximations to stochastic simulators. We apply the general methodology to the Athena model in particular to construct an improved emulator. The approach considers an autoregressive structure for different 'levels' of a simulator. We name the new approach stochastic multilevel emulation. The second contribution of this thesis is investigating algorithms to fit HetGP emulators. Many Bayesian emulators require computationally intensive methods such as Markov chain Monte Carlo to quantify uncertainty about emulator parameters. We recommend using a conjugate prior for HetGP which provides further computational savings and allows for tractable uncertainty quantification about emulator parameters. Our Bayesian HetGPs are used to perform a sensitivity analysis as preparation for expert elicitation for parameters in the Athena simulator. The final contribution of this thesis is combining two distinct approaches. We show, via Bayesian Optimisation and History Matching, that decision making and decision support approaches can work in harmony. We apply this novel approach to construct a set of feasible solutions to a resource allocation problem within an offshore wind farm.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 *Motivation*: *computer experiments*

Historically, science was undertaken almost exclusively via physical experiments. In search of new knowledge, the scientific community did many unusual, and sometimes dangerous, things. Marie Curie's experimentation with X-rays almost surely led to her premature death, Ernest Rutherford (or rather, his students, Hans Geiger and Ernest Marsden), sat in dark rooms for hours in order to prove that atoms are indeed divisible, and to investigate the feasibility of cryonics, Smith *et al.* (1954) froze hamsters and reanimated them using microwaves.

The scientific process has since evolved. We have developed ethical standards for the types of experiments we are allowed to perform, as well and health and safety protocols. A key advancement has been the development of mathematical and statistical modelling to predict what may happen in complex systems. Such models take inputs, $\mathbf{x}$, transform them through a function, $f(\cdot)$, and produce an output, $y = f(\mathbf{x})$. The function $f(\cdot)$ may encode our understanding of a physical process via a set of complex equations and $\mathbf{x}$, the parameter settings, may represent a hypothetical scenario. Evaluating $f(\cdot)$ at many values of $\mathbf{x}$ allows a scientist to ask many "what if" questions, at a fraction of the cost of running an equivalent physical experiment. The function $f(\cdot)$ is usually intractable, thus must be evaluated via a computer code. We refer to a computer code as a *simulator*. Typical simulators can be of the order of hundreds, or even thousands, of lines of code. Answering these "what if" questions via a computer is what we might term a *computer experiment*. Within the biological community, this form of experiment is sometimes termed an *in silco* experiment.

In many engineering contexts, such as the generation of energy, we encounter physical systems that are incredibly complex. An engineer may want to optimise parameters of these systems, or simply explore what would happen under many "what if" scenarios. Performing physical experiments on gas distribution networks, full scale nuclear power plants, or offshore wind farms located miles away from the coast is not a straightforward task. Even if these experiments were financially feasible, the experiments would undoubtedly take *a very long time*.

Using a computer simulation to mimic a physical system is no panacea. Despite modern computing capabilities, it is typical of many computer simulators to have long run times. Exactly what 'long' means is context-dependent, but the Athena simulator (Zitrou *et al.*, 2013, 2016, 2021), used throughout this thesis, has typical run times of 5 minutes for a single parameter setting on a modern desktop computer. A 5 minute run time might be acceptable for making a handful of runs, but becomes problematic when we want to make thousands of runs. A more extreme example is HadCM3 — the climate model used by the MET office — run times are in excess of a week for a single parameter setting (Domingo *et al.*, 2020).

Despite the advantages of using a model in place of physical experiments, we must always remind ourselves that the model is exactly that. It is a model. An imperfect representation of the physical system it aims to mimic.

1.1.1 *The role of statistical thinking in computer experiments*

Statistics has made, and continues to make, contributions to computer experiments and the wider field of Uncertainty Quantification (UQ). The main role of statistics in this field is the development and application of surrogate models known as *emulators* (Kennedy & O'Hagan, 2001; Vernon *et al.*, 2014). Emulators take a set of inputs X and evaluations of the simulator at those inputs $\mathbf{y} = f(X)$, to construct a statistical model typically several orders of magnitude faster to run than the original simulator (Salter & Williamson, 2016). The statistical model is an approximation to the simulator, thus an appropriate quantification of uncertainty is attached to the emulator's point predictions.

Statistical thinking does not just allow us to perform efficient prediction, and besides, prediction for its own sake is rarely the objective of a simulator. We typically develop simulators as a means to perform another task, such as optimisation, inference, calibration or decision analysis. Phrasing the use of a complex

simulator in statistical terms has lead to many developments. A popular example is *Bayesian Optimisation*, where ideas from statistical decision theory allow us to optimise expensive simulators with relatively few runs of the simulator (Frazier, 2018). Sampling schemes such as Latin Hypercubes (McKay *et al.*, 1979), allow us to construct designs for simulation experiments which cover the input space efficiently which eases the task of emulation. Elicitation and subjectivism allow us to consider model discrepancy and thus allow us to calibrate simulators to observational data, whilst allowing for the fact that the simulator (and emulator) are imperfect representations of a corresponding physical system (Kennedy & O'Hagan, 2001; Vernon *et al.*, 2022*a*).

1.2 *Contributions*

1.2.1 *Stochastic multilevel emulation*

The first main contribution is the development an emulator that we call a stochastic multilevel (SML) emulator. The SML emulator combines two established approaches; HetGP and multilevel emulation.

HetGP (Goldberg *et al.*, 1998; Binois *et al.*, 2018) is a data-hungry model, which is otherwise well suited to the emulation of stochastic simulators. It uses a pair of GP emulations to construct flexible mean and variance estimates for stochastic simulators. Multilevel emulation leverages the fact that many simulators can be run at varying levels of complexity, and thus we have a more accurate simulator at higher 'levels'. The more accurate/complex the simulator is, the longer it takes to run. The multilevel emulator (Kennedy & O'Hagan, 2000), finds a way to combine runs from different levels of *deterministic* code, to provide improved predictions of the most expensive level of code. Our SML emulator combines these two approaches to enable us to use cheap versions of a stochastic simulator to construct improved predictions for a more expensive version. We also consider several estimation techniques for HetGP and SML, and find a computationally efficient way to incorporate prior information into both HetGP and SML emulators.

1.2.2 *A decision support framework for offshore wind*

Our second main contribution is the development of a decision support framework using stochastic simulators. This is an extension of the decision support frameworks given in Lawson *et al.* (2016) and Owen *et al.* (2020) who study deterministic simulators. The core idea in decision support is to find decisions (model inputs) which are within a given tolerance of some approximately optimal reference value. Our application has several novel ideas; firstly we discuss how decision making and decision support can be brought closer together via Bayesian optimisation and history matching. Our application also has a challenging feature; a subset of our simulator inputs form a discrete simplex. Within the computer experiments literature, inputs are usually formed on a hypercube. We discuss a method for generating uniformly distributed samples from the discrete simplex which are within tolerance of the reference value. We also consider post-processing of the uniform samples to improve the space-filling properties of a design for a multi-wave computer experiment.

1.3 *Thesis outline*

In Chapter 2 we introduce the Athena simulator, a complex, stochastic simulator used to model the behaviour of a large offshore wind farm. The purpose of the Athena simulator is to aid decision making under uncertainty. This uncertainty is decomposed into two parts. Firstly, aleatory uncertainty; the inherent randomness of the world. The second is epistemic uncertainty; we represent this mathematically via a (prior) probability distribution. Athena employs a large and complex prior probability distribution over many unknown quantities for which there is little or no relevant data available; many relate to the lifetime distributions of critical components in Athena.

Probability elicitation is important within Athena as many of the parameters are uncertain and thus are represented in the Athena simulator by an expert's prior distribution. We want to propagate the uncertain parameters through the Athena simulator to understand how input uncertainty induces output uncertainty. We therefore discuss some core concepts from probability elicitation. Since Athena is used to help make operation and maintenance decisions within an offshore wind setting, we also provide an introduction to Bayesian decision analysis and utility

theory; a framework for making decisions.

The use of computationally expensive models makes analyses relying on many evaluations of $f(\mathbf{x})$ extremely computationally expensive. Such analyses include uncertainty analysis or decision analysis. In Chapter 3, we introduce the notion of an *emulator*. Emulators are fast, statistical approximations to complex computer simulators. Emulators repurpose and adapt regression techniques to produce approximations to complex computer simulators.

In Chapter 4, we describe the heteroscedastic Gaussian Process (HetGP) as a way to emulate stochastic computer models which exhibit input-dependent noise. We develop a general approach for SML emulation, which uses different versions of a simulator to improve the final emulator. Finally, we compare some estimation methods for HetGP and SML via an application to the Athena simulator.

We return to probability elicitation in Chapter 5. In particular, we perform a probabalistic sensitivity analysis of the Athena simulator. This is facilitated by the HetGP and SML emulators fitted in Chapter 4.

Chapter 6 marks a change of direction in the thesis. We review some common optimisation approaches with a view towards decision making (which can be expressed as an optimisation problem). We then discuss how emulators can be incorporated into the optimisation process. We consider some practical differences between *making* decisions and *supporting* decision making. We motivate the use of history matching inspired techniques as an appropriate method for facilitating decision support. Finally, we discuss algorithms for generating sets of decisions that are consistent with an approximately optimal decision.

In Chapter 7, we put into practice some of the techniques described in Chapter 6. The optimisation techniques are used to maximise the expected utility function which depends on the output of Athena. We utilise techniques given in Owen *et al.* (2020) and extend them to the stochastic case. We also consider the problem of generating decisions on a subset of a discrete simplex; this is a nonstandard problem within the uncertainty quantification literature.

The thesis is drawn to a close in Chapter 8. We review the findings of this thesis and provide future research avenues.

Chapter 2

Background

This thesis aims to develop tools to aid computationally expensive analyses that a subjective Bayesian may perform. These computationally expensive tasks usually amount to computing $Y = f(\mathbf{X})$ where $f(\cdot)$ is an expensive to evaluate function, which is thought to be highly non-linear, $\mathbf{X}$ is an uncertain vector of inputs to $f(\cdot)$ and Y is the (uncertain) quantity of interest. We consider two concrete instances of such a problem.

The first problem we consider is a sensitivity analysis to aid the elicitation of $\mathbf{X}$. The sensitivity analysis relies on calculating the intractable integrals $\mathrm{E}_X\{f(\mathbf{X})\}$, $\mathrm{Var}_X\{f(\mathbf{X})\}$, as well as conditional expectations and variances, such as $\mathrm{E}_{X_i}\{\mathrm{Var}_{\mathbf{X}_{-i}}(f(X) \mid X_i)\}$, where $\mathbf{X}_{-i}$ denotes all elements of $\mathbf{X}$ apart from the ith element. This in turn relies on evaluation of $f(\mathbf{x})$ for many values of $\mathbf{x}$. The word 'many' is somewhat vague, but we would expect that evaluating $f(\cdot)$ at somewhere in the region of $10^4 - 10^6$ unique input configurations to be necessary to obtain accurate estimates of the required expectations and variances.

The second problem we consider is decision analysis, in which we elicit a function of the form $U(\mathbf{x}) = \mathrm{E}_{\boldsymbol{\theta}}\{u(\mathbf{x}, \boldsymbol{\theta})\}$ where $\boldsymbol{\theta}$ is a collection of unknown quantities, $\mathbf{x}$ is the vector representing the decision to be made, and u is a function which itself depends on $f(\mathbf{x})$. This problem is computationally intractable; evaluation of $U(\mathbf{x})$ is typically only achievable via Monte Carlo methods. This makes maximising $U(\mathbf{x})$ difficult because $U(\mathbf{x})$ itself cannot be computed exactly. The difficulty of an intractable decision analysis is increased when $U(\mathbf{x})$ is expensive to compute, which is a problem we have.

We will perform analyses on the computationally expensive Athena simulator, which is a stochastic model of an offshore wind farm (Zitrou *et al.*, 2013, 2016,

2021). This chapter will consist of primers on the Athena simulator (Section 2.1), probability elicitation (Section 2.2), and Bayesian decision analysis, including the elicitation of utility functions.

2.1 *The Athena simulator*: *a stochastic model of an offshore wind farm*

The Athena simulator (Zitrou *et al.*, 2013, 2016, 2021) is a point process model of a large offshore wind farm. The intended purpose of Athena is to aid decision making under uncertainty, as such, Athena does not model raw power output or take into account precise details of the wind, but aims to model the reliability of a wind farm using techniques from reliability analysis. The reason for not modelling raw output or including synthetic weather data is that they are highly uncontrollable and very uncertain, although their aggregate effects can be characterised much more precisely. We can however, make plausible predictions about the *reliability* of the wind farm.

The Athena simulator generates predictions by simulating events within an offshore wind farm over a pre-specified time period $[0, T_{max}]$ years. There are many types of event that may happen within an offshore wind farm. Typical events include components breaking, components being repaired or replaced, and the deployment of boats to perform repairs. Other, less common, events may be the triggering of farm-wide maintenance, or turbines being switched on in the early stages of the wind farm's operational life.

To simulate events, the simulator starts from time $t = 0$ and calculates the hazard function of each event possible at each time point over the period of interest; this is a function of time and the state of the wind farm. From this the "total" hazard (at each time point), known as the Force of Mortality (FOM), is calculated which then implies the next event time. If we are at T_{p-1} then the time to the next event, T_p is found as $R^*(T_p) = R^*(T_{p-1}) + E$ where $E \sim Exp(1)$ and R^* is the cumulative intensity function of the wind farm. The use of an $Exp(1)$ random variable is justified by a general, theoretical result; see Pham (2006) (Theorem 8.8).

The event time T_p is the deterministic solution to an integral which depends

on the wind farm's current, stochastic, state. The integral is

$$R^*(\tau_n) - R^*(\tau_{n-1}) = \int_{\tau_{n-1}}^{\tau_n} r^*(u)\,\mathrm{d}u \tag{2.1}$$

where r^* is the wind farm's intensity function. If $\tau_{n-1} = T_{p-1}$ is the time of the last event, then $R^*(\tau_{n-1})$ is known. The goal is to find $R^*(\tau_n)$ by sequentially increasing τ_n to $T_{p-1} + \Delta t$, $T_{p-1} + 2\Delta t$, ... until $R^*(\tau_n) - R^*(\tau_{n-1}) > E$. The first value of τ_n satisfying the inequality is taken as T_p. Athena then decides which component caused the event. Events are typically subassemblies of a particular turbine failing or being repaired. The Athena simulator models the turbines as being constructed of 8 main subassemblies and a 9th 'catch all' subassembly, which collectively models the behaviour of several less important subassemblies which together have a non-negligible effect. The subassemblies are the gearbox, generator, frequency converter, transformer, main shaft bearing, the blades, tower, foundations and the catch all.

Events are not caused only by subassemblies, for example, cables and transformers can fail and be repaired. Let $y_{j,k}(T_p)$ be an indicator taking the value 1 when subassembly (j,k) has failed and is 0 otherwise. Let $\nu_{j,k}(T_p)$ be the failure intensity of subassembly (j,k) and let $\mu_{j,k}(T_p)$ be the corresponding restoration intensity, then the probability that subassembly (j,k) caused the event at time T_p is

$$\begin{aligned} p_{j,k}(T_p) &= \frac{y_{j,k}(T_p)\nu_{j,k}(T_p) + (1 - y_{j,k}(T_p))\mu_{j,k}(T_p)}{P + Q} \\ P &= \sum_{j,k} \left\{ y_{j,k}(T_p)\nu_{j,k}(T_p) + (1 - y_{j,k}(T_p))\mu_{j,k}(T_p) \right\} \\ Q &= \sum_{l} \left\{ y_{l}(T_p)\nu_{l}(T_p) + (1 - y_{l}(T_p))\mu_{l}(T_p) \right\} \end{aligned} \tag{2.2}$$

where $y_l(T_p)$ is an indicator for the lth component which is not a subassembly, with failure intensity $\nu_l(T_p)$ and restoration intensity $\mu_l(T_p)$. The probabilities $\{y_{1,1}(T_p), y_{2,1}(T_p), \ldots, y_1(T_p), y_2(T_p), \ldots\}$ form a partition of $[0,1]$, so drawing a $U(0,1)$ random variable allows us to simulate which event occurred. This is repeated until we reach the end of the pre-specified simulation period T_{max}.

The time to failure, $T_{j,k}$, of subassembly j in turbine k is modelled by a non-stationary Weibull distribution: $T_{j,k} \mid t \sim Weibull(\alpha_{j,k}(t), \kappa_{j,k}(t))$. Recall that if T

is the time at which a component fails, then the survival function is $S(t) = \mathrm{P}(T > t)$, the hazard function is given by

$$h(t) = \frac{-S'(t)}{S(t)} \tag{2.3}$$

$$\iff S(t) = \exp\left\{-\int_0^t h(u)\,\mathrm{d}u\right\}. \tag{2.4}$$

We can then construct a Weibull distribution with time-dependent parameters by constructing a hazard function which changes form with time. In particular, a hazard function for a subassembly follows a 'bathtub' curve which controls $\alpha_{j,k}(t)$ and $\kappa_{j,k}(t)$. A bathtub hazard function corresponds to three phases of component life

(i) infant mortality (decreasing hazard)

(ii) useful life (constant hazard)

(iii) degradation (increasing hazard).

Phase (i) corresponds to a period in which a larger than expected number of components fail due to manufacturing faults, or lack of operator experience. This is an important consideration when novel technologies and being employed in harsh environments, or existing technologies are employed in unfamiliar environments. Both of these scenarios are present within an offshore wind setting. In the next phase, the component works as expected; failures are caused only by external factors, such as extreme weather events damaging turbines or random operator error. Finally, components enter the degradation phase, in which components are starting to feel the effects of ageing, the components are becoming increasingly worn out with time and thus are increasingly likely to fail with time.

We can write a bathtub hazard function in the following form:

$$h(t) = \begin{cases} \alpha_0\kappa_0(\alpha_0 t)^{\kappa_0-1}, & t \in [0, t_1) \\ \alpha_1, & t \in [t_1, t_2) \\ \alpha_2\kappa_2(\alpha_2 t)^{\kappa_2-1}, & t \in [t_2, \infty) \end{cases} \tag{2.5}$$

where $\kappa_0 < 1, \kappa_2 > 1$ and the constant hazard case ($h(t) = \alpha_1$) could be represented by $h(t) = \alpha_1\kappa_1(\alpha_1 t)^{\kappa_1-1}$ with $\kappa_1 = 1$, that is, exponentially distributed failure times.

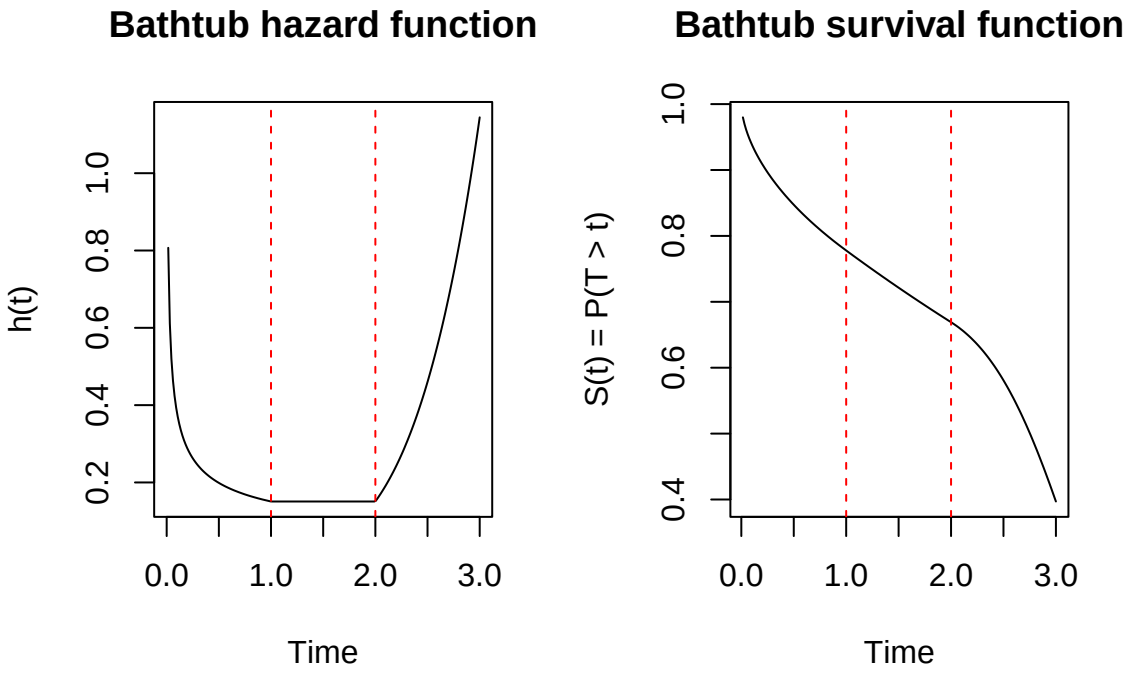

Figure 2.1: An example of a bathtub hazard function (left) with the corresponding survival function (right). The dashed red lines indicate t_1 and t_2, the points in time when the nature of the hazard function changes.

The hazard function is continuous so that $\alpha_0\kappa_0(\alpha_0 t_1)^{\kappa_0-1} = \alpha_1 = \alpha_2\kappa_2(\alpha_2 t_2)^{\kappa_2-1}$ and In Figure 2.1 we present an example bathtub hazard function (left) with the corresponding survival function (right). Athena incorporates additional details into the hazard function. Performing maintenance tasks extends the expected life of a subassembly, whereas operator misuse decreases lifetimes, and the probability of operator misuse decreases with time, reflecting increased experience. A concept known as 'virtual life' allows us to further manipulate $h(t)$. If we replace t by $v(t)$, the virtual life, the hazard function, $h(v(t))$, can then reflect the state of the subassembly it models more accurately. We can think of $v(t)$ as the effective age of a component. Performing preventative maintenance should improve the lifespan of a component, thus it is effectively a 'newer' component than prior to maintenance (Finkelstein, 2007). A driver of subassembly lifetime is the onset of ageing, that is, the start of phase (iii) of the hazard function.

In practice, the values of many model parameters in Athena are unknown, thus uncertainty distributions are to be elicited from experts and propagated through

Athena to understand how input uncertainty induces uncertainty in key metrics. A key model output is a time series which tracks the "availability" of a wind farm over time. Availability is a measure of reliability (performance) of offshore wind farms; the availability at time t is the energy output of the wind farm as a proportion of the maximum possible energy output at time t. In Figure 2.2 we show a collection of 300 availability trajectories, from a variant of Athena which is explained in Section 2.1.1. We see two modes in this particular collection. The main mode exhibits trajectories that never appear to drop below 0.8. The lesser mode consists of trajectories which decrease shortly after the wind farm begins operation and do not start recovering until 2.5 years of operational life has passed. After 3.5 years, the trajectories all exhibit stable behaviour, apart from one trajectory which drops below 0.9; this is due to a shortage of spare components.

Offshore wind farms have a mean availability of around 93% for near shore turbines, but this is reduced for turbines further away from the coast since reaching the turbines for repair is much more difficult (Carroll *et al.*, 2016). Availability is related to a wind farm's uptime and hence its profitability. In the first 5 years of operation, *excessive failure* is frequently observed. That is, the wind farm typically under-performs due to higher than expected numbers of component failures; tackling this issue is vital to the feasibility of offshore wind.

2.1.1 *A variant for spare components*

The above description of the Athena simulator assumes that, when a subassembly suffers a failure that warrants replacement, rather than repair, that a replacement component is immediately available. In practice, this is an unrealistic assumption. Improving upon this assumption is necessary for our analysis in Chapter 7. A minor contribution of this thesis is some additional modelling within Athena to incorporate spare components and implementing this as simulator code.

A practical workaround to the problem of subassemblies being difficult to obtain, is to pre-order spare subassemblies and store them in a warehouse that is onshore, but close to the offshore wind farm. Within the Athena simulator, a subassembly can experience one of three types of failure. These are minor, moderate and severe failures. We have assumed that minor and moderate failures are of a nature such that they can be repaired with readily available tools or components. Severe failures are such that replacement of the subassembly is the

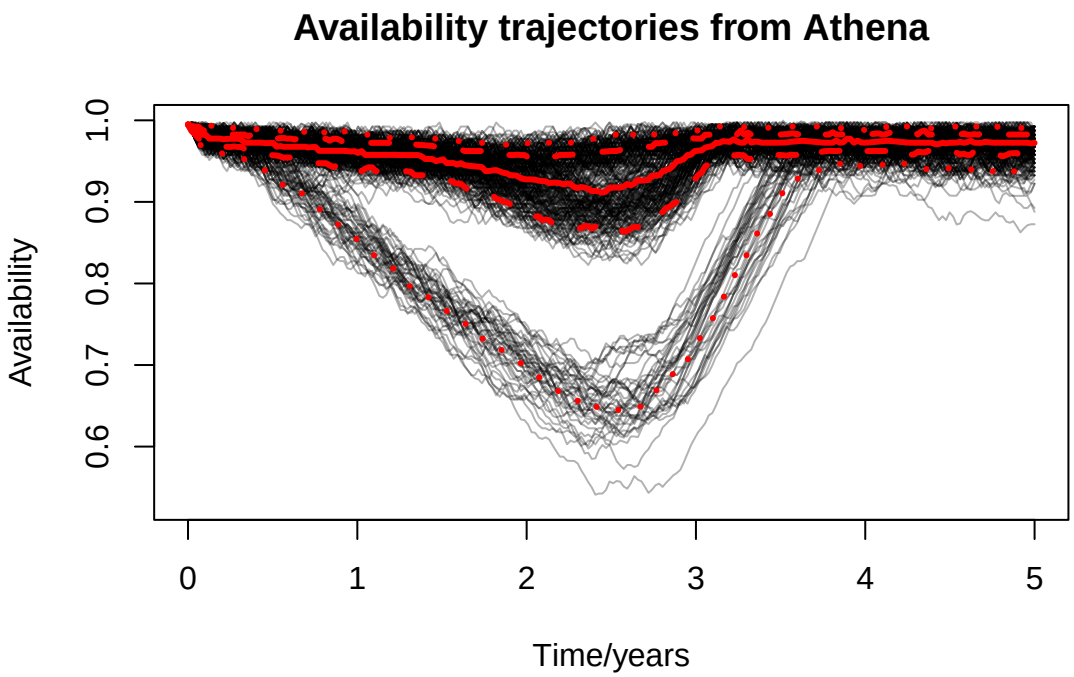

Figure 2.2: A collection of 300 typical availability trajectories (black lines) over the first 5 years of a wind farm's operational life for a fixed set of parameter values. The solid red line represents the point-wise median trajectory, the dashed red lines represent 20^{th} and 80^{th} point-wise percentiles and the dotted red lines represent 5^{th} and 95^{th} point-wise percentiles.

only feasible option. We assume spare components are stored in a warehouse, which holds a fixed number of spare parts. At the start of the simulation, the user must choose how many of each spare part the warehouse can hold. These numbers are fixed for the entire simulation period. Denote the maximum number of spare subassemblies of type i by s_i. Now let $s_i(t)$ be the number of spare parts we have available of type i at time t. In essence, we have a finite number of spare parts readily available, which may be replenished at some future time.

Now suppose that a subassembly of type i has suffered a serious failure. If there is a readily available spare part, then the part is shipped out to the turbine and it is repaired according to the processes within the 'standard' version of the Athena simulator (so we still need the correct type of boat to be available, for example). If no spare part is available, because all the spares of type i have been used, then the repair cannot take place until a new batch of spare parts

arrives. This motivates a mechanism for ordering in more spare parts. We define a threshold, s_i' such that, as soon as $s_i(t) < s_i'$, an operator within the wind farm places an order for more spare components. The s_i' are specified at the start of the simulation and remain fixed for a given run of the simulator. The waiting time for the order is uncertain. If the order is placed at time t then the order arrives at time $t + \Delta t$ where $\Delta t \sim Gamma(w_1, w_2)$, where w_1 and w_2 are uncertain parameters which could be elicited or specified by an expert.

There is also an extra mechanism regarding spare parts. It may be the case that the supplier cannot fulfil the order, or the order is only partially completed. We model the number of ordered spares that can actually be delivered, s, by a Binomial distribution; $s \sim Binomial(s_i - s_i(t), p)$. We would imagine that commonly, the order would be completely fulfilled. In such a case we can choose p to be quite large, for example, $p = 0.99$. Whether an order can be fulfilled or not follows a Bernoulli distribution. We introduce a random variable $\omega \sim Bernoulli(p')$ where p' is the probability that an order is fulfilled. p' would typically take a large value, such as 0.99. The total number of spares delivered from the order is then $s\omega$, and this is delivered at time $t + \Delta t$.

This concludes our introduction to the Athena simulator. We have introduced the necessary details for the applications later in this thesis, although more details can be found at Zitrou *et al.* (2013, 2016, 2021).

2.2 *Elicitation of probability distributions*

When quantities are unknown, and there is a lack of relevant data to estimate them, a feasible solution is to elicit them from an expert, or group of experts. Elicitation is the act of turning the knowledge or beliefs an expert has about unknown quantities into a joint probability distribution.

The elicitation of uncertain inputs, although useful when there is a lack of relevant data (or necessary if we wish to perform a Bayesian analysis utilising genuine prior beliefs), is a time consuming task which needs to be done carefully.

2.2.1 *The basics of elicitation*

In the elicitation process there are two roles. First we have the *expert*. This is the individual whose knowledge or beliefs we aim to elicit and represent via

a probability distribution. The term 'expert' refers to their domain knowledge and expertise and has no reflection of their knowledge of statistics or probability. Commonly we will refer to multiple experts rather than singular; it is encouraged to combine the beliefs of multiple experts into a single distribution. The second role is the *facilitator*; this a person who has good knowledge of probability and statistics, they guide the expert through the elicitation process and fit distributions to the elicited quantities.

Elicitation of a single uncertain quantity, θ, is described as a four stage approach by Garthwaite *et al.* (2005).

The first stage is setting up the elicitation; this involves identifying an expert (or even a group of experts, as in Williams *et al.* (2021)), providing the experts with training and identifying precisely what θ is. By precisely, we mean that if θ is the rate at which the a gearbox, of brand X and model Y, would degrade at an offshore wind farm at location Z, then the experts should be told that θ is this value, rather than just "the rate at which a gearbox degrades".

The second task in the process is the 'main' elicitation step: the expert states various summaries about θ. Summaries are typically of the form $\mathrm{P}(a_i < \theta < b_i) = p_i, i = 1, 2, \ldots, n$. The facilitator can then fit a statistical model to the elicited (a_i, b_i, p_i). In some cases we might be able to do this exactly and analytically. For example, if we elicit two quantities and assume $\theta \sim \mathcal{N}(\mu, \sigma^2)$, we can find μ and σ via a pair of linear simultaneous equations in μ and σ. More generally, if there are more p_i than parameters of the chosen distribution, or the form of the distribution does not lend itself to a tractable solution, then a numerical approach, such as minimising $T(\varphi) = \sum_i (p_i - \mathrm{P}(a_i < \theta < b_i \mid \varphi))^2$ may be appropriate, where φ are the parameters of the distribution to be fitted. This stage, and the previous stage, of fitting are somewhat interrelated; the form of the distribution the facilitator may wish to fit to the expert's beliefs may influence the questions asked. For example, when eliciting a distribution for a proportion, a Beta distribution, $\theta \sim Beta(\alpha, \beta)$, offers an appropriate model. Knowing that we will fit a Beta distribution to the elicited information means we would never ask the expert for p_i such that $p_i = \mathrm{P}(5 < \theta < 50)$.

The 'final' stage is then checking the adequacy of the fitted distribution. In this stage, the facilitator may show the expert what their elicited statements imply about θ. If these statements are inconsistent with the expert's beliefs, we must go back and forth between the second, third and fourth stages until the expert and

facilitator agree on a distribution, $\pi(\theta)$, which is an appropriate description of the expert's beliefs.

We have excluded the details about what kinds of questions a facilitator should ask the expert in order to obtain summaries of θ. For guidance on the exact phrasing of questions, see O'Hagan *et al.* (2006), in particular, Chapter 5. Note that the types of questions to be asked are often related to the nature of the uncertain quantities, thus the general recipe for elicitation is necessarily vague. Some further practical guidance is given in Kadane & Wolfson (1998), who advocate for eliciting *observable* quantities rather than the parameters of any statistical model. An observable quantity is, as its name suggests, a quantity that can be observed by the expert. This means we should ask experts about the quantities exhibited by a physical system (for example, a time to failure of a component in the wind farm), rather than parameters of a statistical model (for example, the parameters of a Weibull distribution).

2.2.2 *The Sheffield Elicitation Framework*

A popular elicitation procedure, the Sheffield Elicitation Framework (SHELF), provides a thoughtful routine for eliciting an unknown quantity from one or more experts. The procedure can be applied to a single expert, but we adopt the convention of multiple experts. SHELF is also used for multiple uncertain quantities, which we describe later.

SHELF consists of a collection of documents, which includes forms and presentation slides (Oakley & O'Hagan, 2019), this is accompanied by software to fit distributions to elicited beliefs, as well as validate them (Oakley, 2021). Both the software and documents are designed to help the facilitator and the experts elicit an uncertain quantity of interest. SHELF covers all aspects of the elicitation process, from the recruitment of experts, to the 'main' elicitation step, as well as checking the adequacy of $\pi(\theta)$ as a representation of the expert's knowledge of θ. An important concept in SHELF is RIO: the rational impartial observer. RIO is a hypothetical fly on the wall who, if they had no strong beliefs about θ prior to observing the discussion amongst experts, would hold beliefs characterised by $\pi(\theta)$ after observing their discussion. The experts are instructed to formulate a distribution, $\pi(\theta)$, which summarises the knowledge of all the experts. This combined judgement does not have to be an equal weighting of each individual's

beliefs; the experts are free to give more consideration to more knowledgeable experts as they see fit. The core elicitation step is broken into two main parts; individual elicitation and a group elicitation as follows:[1]

1. Conduct an individual elicitation for each expert as follows:

   (i) Specify a lower value, L, such that the event $\theta < L$ is 'extremely unlikely'. By 'extremely unlikely', it is meant that if you were to learn that $\theta < L$, your initial reaction would be that the statement $\theta < L$ is false. Repeat the process for an upper limit, U, where the event $U > \theta$ is extremely unlikely.

   (ii) Elicit the median, m such that $p(\theta < m) = 0.5$. m is the value that placing a bet on $\theta < m$ has an equally desirable outcome to placing a bet on $\theta \geq m$.

   (iii) Elicit a lower quartile, Q_1 such that $\mathrm{P}(L \leq \theta < Q_1) = \mathrm{P}(Q_1 \leq \theta < m) = 0.25$. As above, placing a bet on the event $L \leq \theta < Q_1$ should be equally desirable to placing a bet on the event $Q_1 \leq \theta < m$.

   (iv) Elicit an upper quartile, Q_3 such that $\mathrm{P}(m \leq \theta < Q_3) = \mathrm{P}(Q_3 \leq \theta < U) = 0.25$. This step is similar (iii).

   (v) The facilitator now fits a distribution to the elicited knowledge, treating the limits, quartiles and median as data.

2. Conduct a group elicitation as follows:

   (i) As a group, perform all the steps in part 1.

   (ii) Verification: is the fitted distribution what a RIO would conclude as appropriate?

This process is what we will describe as a 'full SHELF' elicitation. Step 1.(v) can be performed in a semi-automatic way via the `SHELF::elicit()` function, which opens a Shiny app. The facilitator fills out boxes within the app corresponding to elicited information. The app can either fit a single chosen parametric form, or try many parametric forms and return the form which offers the closest fit to the elicited beliefs. A screenshot of the app is given in Figure 2.3. In the example,

[1] We have condensed the steps considerably. We aim to communicate the core steps in SHELF rather than replicate the documentation.

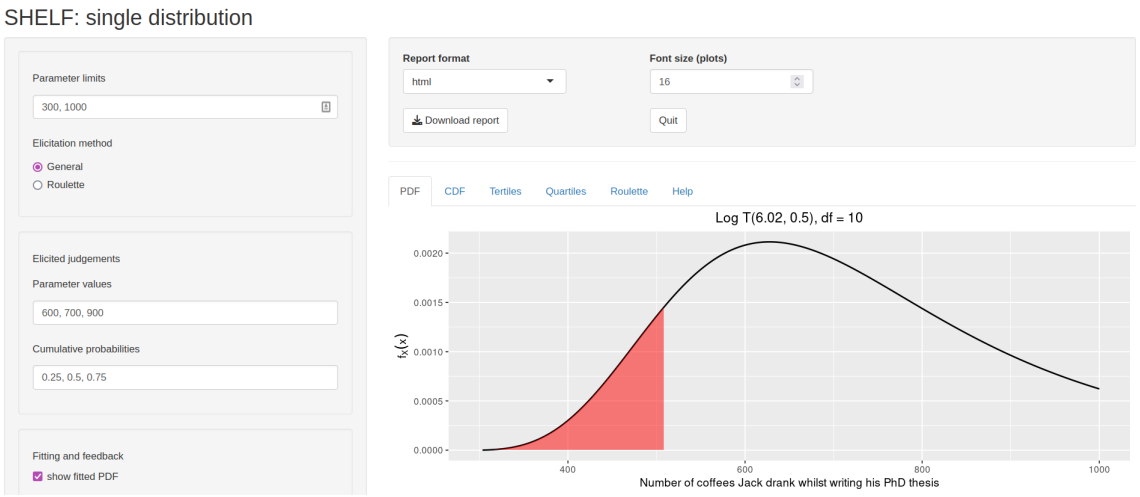

Figure 2.3: Screenshot of distribution fitting functionality provided by the `SHELF` package (Oakley, 2021). In this particular example, we have elicited the number of cups of coffee the author of this thesis believes he has drunk in order to complete this thesis. The red shaded area represents a feedback question.

the elicited information leads to $\log \theta \sim t_{10}(6.02, 0.5)$, where θ is the number of coffees consumed in the production of this thesis and $t_\nu(\mu, \lambda^2) = \mu + \lambda t_\nu$ is a location-scale variant of the t distribution.

2.2.3 *Elicitation of many uncertain quantities*

Suppose we have $k > 1$ uncertain quantities, $\boldsymbol{\theta} = (\theta_1, \theta_2, \ldots, \theta_k)^T$. We wish to characterise uncertainty about $\boldsymbol{\theta}$ via $\pi(\boldsymbol{\theta})$, a joint density. If the expert is willing to assert that $\theta_i \perp\!\!\!\perp \theta_j \, \forall \, i \neq j$, then we may write $\pi(\boldsymbol{\theta}) = \prod_{i=1}^{k} \pi_i(\theta_i)$. In this case, all we need to do is elicit $\pi_i(\theta_i)$ using the univariate method outlined above.

If the components of $\boldsymbol{\theta}$ cannot be assumed to be independent there are two approaches within SHELF. The first is to rephrase the problem in terms of new unknowns, $\boldsymbol{\theta}'$, such that $\theta'_i \perp\!\!\!\perp \theta'_j \, \forall \, i \neq j$. For example, suppose we are planning the programme for a statistics conference, we may want to elicit

(i) θ_1 = the number of delegates at a particular statistics conference

(ii) θ_2 = the number of delegates at the conference who are Bayesian.

Clearly, $\theta_1 \geq \theta_2$, thus a dependence is present. Now let $\theta'_1 = \theta_1$ and $\theta'_2 = \theta_2/\theta_1$, that is, the proportion who are Bayesian. The statement $\theta'_1 \perp\!\!\!\perp \theta'_2$ seems more plausible than $\theta_1 \perp\!\!\!\perp \theta_2$. We recover $\boldsymbol{\theta}$ via $(\theta_1, \theta_2) = (\theta'_1, \theta'_1\theta'_2)$.

The second approach is to try to elicit the dependence between elements of $\boldsymbol{\theta}$. This is a much more challenging task than assuming independence, thus should only be done when necessary. Within SHELF, dependence may be elicited via assuming an appropriate distributional family that lends itself well to dependence (for example, multivariate Normal or Dirichlet). The second option is to incorporate dependence via a copula. A copula is a useful device for constructing a dependence structure between two random variables with known marginal distributions (Bedford & Wilson, 2014; Elfadaly & Garthwaite, 2017). In either case, we must elicit marginal distributions and the dependence structure.

Eliciting even the marginal distributions can take a long time. The European Food Safety Authority (EFSA) state that elicitation of a single unknown takes about a day (European Food Safety Authority, 2014). EFSA believe that elicitation of 4–6 parameters takes about 2 days. The experts are able to perform elicitation more quickly because they become accustomed to the process. Elicitation workshops are time limited for practical reasons. The first is that experts will become fatigued, thus after several days their judgements degrade in quality. The second is that experts will only be available for a limited amount of time; they will have other commitments. We may only have access to the experts for a day or so. For this reason, we must think carefully about which parameters deserve a full SHELF treatment and which parameters require less thought. This amounts to different types of elicitation. The most rigorous is the full SHELF treatment, reserved for the most important parameters.

An approach that is less involved than SHELF, but retains many of its features is probabalistic Delphi (Rowe & Wright, 1999), which can take place remotely (for example, via email). A questionnaire about unknowns, with questions similar to that found in a full SHELF workshop, is sent to experts, which they return to the facilitator. The questionnaire responses are then circulated anonymously amongst all experts so they can review the other opinions about $\boldsymbol{\theta}$ and revise their own beliefs as appropriate. The revised beliefs are sent to the facilitator who then uses a mathematical aggregation rule — such as a linear combination of each expert's $\pi(\boldsymbol{\theta})$ — to combine each expert opinion into a single distribution.

A less sophisticated, but much less time consuming, approach would be to employ a 'minimal assessment'. Minimal assessment may proceed by eliciting L and U from each expert, then for the facilitator to assume a very simple form for $\pi_i(\theta_i)$. For example, we might take $\theta_i \sim \mathcal{U}(L, U)$. Another simple form would

be $\theta_i \sim \mathcal{N}(\mu, \sigma^2)$ with $\mu = \frac{1}{2}(L + U)$ and $\sigma = \frac{1}{2}(U - \mu)$ (remembering that 95% of a Normal distribution's mass lies within $\pm 2\sigma$ of μ). Each expert's L and U could be revised via a Delphi-like approach, or the facilitator could take $L^* = \min L$, $U^* = \max U$ and fit a distribution based on U^* and L^*. A minimal assessment is often employed when we want to use expert judgement for a prior in a Bayesian analysis, but we know that a reasonably large sample of data is available. The sample will be sufficiently large to dominate any prior specification, thus we choose not to worry about precise details of $\pi(\boldsymbol{\theta})$.

How we choose which parameters warrant a full SHELF treatment and which parameters require a less formal treatment can be dealt with via a sensitivity analysis. In almost all cases, the distribution $\pi(\boldsymbol{\theta})$ is not of direct interest. We are usually interested in the distribution $\pi(\boldsymbol{\theta})$ induces on $f(\boldsymbol{\theta})$, a function of the uncertain parameters. Concretely, we may want to propagate uncertainty about model parameters, $\boldsymbol{\theta}$ through a complex model (such as the Athena simulator) to make predictions about quantities relevant to decision making, such as the availability time series, or summaries of it. The SHELF documentation suggests that a one-way sensitivity analysis may be appropriate. In this case we study how much $f(\boldsymbol{\theta})$ changes when θ_i is perturbed to $\theta_i \pm \Delta\theta_i$. How much $f(\boldsymbol{\theta})$ changes when only θ_i changes is a simple, but limited, way to investigate the relative importance of each θ_i. In Chapter 5 we give an in-depth discussion of sensitivity analysis, the relative importance of parameters, and best practices for sensitivity analysis with a view towards eliciting uncertain parameters of a model.

2.3 *Decision analysis*

Unless explicitly stated otherwise, this section is based upon Keeney & Raiffa (1976), Smith (2010) and chapters of Dias *et al.* (2018); in particular González-Ortega *et al.* (2018).

In many of the decision making situations we encounter in our lives, our idea of the 'best' decision may differ to another person's idea about what is the 'best' decision. Consider the problem of preparing a scone to be eaten. One individual may claim that putting jam on a scone *before* the cream is objectively the best method to prepare it. Many other people would express dogmatically that the cream *must* be first. Others would argue that the best way is not defined by the ordering of cream or jam, but much rather, the quantity of cream and jam.

These differences are present because every decision maker (DM) has their own preferences. Much like how we all have different prior beliefs for an unknown quantity, we also have different preferences for outcomes and thus different optimal decisions. A DM's preferences are expressed via a utility function which can be elicited from the DM. This function can have different shapes for different DMs. In fact, different DMs may have utility functions which have different inputs. We will now formalise the notion of a utility function.

Let $\mathcal{X}$ represent the set of all possible decisions the DM could make, and let $\mathbf{x} \in \mathcal{X}$ be a single decision. Further suppose that any relevant unknown quantities, $\boldsymbol{\theta}$, within the decision problem have been assigned a probability distribution $\boldsymbol{\theta} \sim \pi(\boldsymbol{\theta})$. In this thesis $\pi(\boldsymbol{\theta})$ will usually be an elicited prior distribution, but it should be replaced by the posterior distribution $\pi(\boldsymbol{\theta} \mid \mathbf{y})$ when relevant data $\mathbf{y}$ are available. Finally, we need a function $u(c(\mathbf{x}, \boldsymbol{\varphi}))$ which is a mathematical representation of the DM's beliefs for how desirable the consequence $c(\cdot, \cdot)$ is when decision $\mathbf{x}$ has been made and when $\boldsymbol{\theta}$ takes the value $\boldsymbol{\varphi}$. The function $c(\cdot, \cdot)$ may be deterministic when it is clear how decisions impact future events, or it can be a stochastic function when future events are uncertain.

The optimal decision is given by

$$\mathbf{x}^* = \arg\max_{\mathbf{x} \in \mathcal{X}} U(\mathbf{x}). \tag{2.6}$$

where

$$U(\mathbf{x}) = \mathrm{E}_{\boldsymbol{\theta}}\{u(c(\mathbf{x}, \boldsymbol{\theta}))\} \tag{2.7}$$

is the expected utility of the decision $\mathbf{x}$. The expectation in Equation (2.7) depends on several things:

(i) $\pi(\boldsymbol{\theta})$, the probability distribution over all relevant unknown quantities

(ii) $c(\mathbf{x}, \boldsymbol{\theta})$, the *consequence* of decision $\mathbf{x}$ when the unknown quantities take the value $\boldsymbol{\theta}$.

(ii) $u(c(\cdot, \cdot))$, the utility function describing how preferable a consequence is. Larger values of $u(c(\cdot, \cdot))$ indicate more preferable consequences.

For many problems, an 'off the shelf' utility function may adequately describe our preferences. For example, in Bayesian design of experiments, the goal is often to choose a design $\mathbf{x} \in \mathcal{X}$ which allows us to learn the most about some unknown

quantities. One appropriate, default utility function is the Kullback-Leibler divergence between the prior and posterior distributions for the unknowns. A review of appropriate utility functions for Bayesian design of experiments is given in Ryan *et al.* (2016).

For many practical problems, there are many competing objectives which may not be well defined. For example, we would typically want to maximise energy output of the wind farm, and simultaneously minimise operation and maintenance costs. These objectives compete with each other. Increasing maintenance costs may increase energy generation, for example. A less well defined objective is keeping local residents happy with the development of the offshore wind farm, in such a case we must come up with a *proxy* for resident discontent which we can observe. The DM may wish to re-phrase resident discontent as the number of local residents who sign a petition against the development of an offshore wind farm, for example.

Since the development of an offshore wind farm is a highly specialised problem, it is unlikely that an off the shelf utility function will describe the objectives we wish to achieve. Further, different decision makers may have different preferences. For example, the CEO of an energy company may be mainly motivated by profits, whereas, a systems safety engineer's first priority is the safety of those maintaining the wind farm. Therefore, once a DM has been chosen, we need to encode their unique preferences into a utility function. This is done by *eliciting* the utility function, $u(\cdot)$ from the DM. We must also elicit $\pi(\boldsymbol{\theta})$. We do this via the methods given in Section 2.2.

We now provide an overview of some utility elicitation techniques. This allows us to turn an ill-defined decision problem into the well-posed questions of mathematical optimisation.

2.3.1 *Fundamental notions of utility*

There are three axioms of utility theory under risk, where a preference relation $\preceq$ is assumed on a probability space $\mathcal{P}$. To discuss the fundamental notions of utility we need to define a *lottery*. A lottery, $p = [x, y; r]$ is an uncertain event where $p = x$ with probability r and $p = y$ with probability $1 - r$. For lotteries p and q the statement $p \preceq q$ means that q is *at least* as desirable as p. The statement $p \prec q$ means that q is more desirable than p.

The three axioms are as follows:

1. **Weak order:** $\preceq$ on $\mathcal{P}$ is complete; $\forall\ p, q \in \mathcal{P}$ either $p \preceq q$ or $q \preceq p$ and is transitive; $\forall p, q, r \in \mathcal{P}$ $p \preceq q$ together with $q \preceq r \implies p \preceq r$.

2. **Archimedian:** $\forall p, q, r \in \mathcal{P}$ if $p \prec q \prec r$ then $\exists \alpha, \beta \in (0, 1)$ such that $\alpha p + (1 - \alpha) r \prec q \prec \beta p + (1 - \beta) r$.

3. **Independence:** $\forall p, q, r \in \mathcal{P}$ and $\alpha \in (0, 1]$, $\alpha p + (1-\alpha) r \preceq \alpha q + (1-\alpha) r \iff p \preceq q$.

Under these conditions there is a *utility function*, u, satisfying $\forall p, q \in \mathcal{P}$

(i) $p \preceq q \iff u(p) \leq u(q)$

(ii) $u(\alpha p + (1 - \alpha) q) = \alpha u(p) + (1 - \alpha) u(q)$.

From this, we can deduce $p \preceq q \iff \mathrm{E}_p(u) \leq \mathrm{E}_q(u)$, where $\mathrm{E}_p(u)$ is the expected utility of a lottery p.

Note that a utility function is only unique up to a positive affine transformation; $au(\cdot) + b$, for any $a > 0$ and $b \in \mathbb{R}$, represents the same set of preferences as $u(\cdot)$.

2.4 *A basic single attribute utility elicitation*

One of the first steps in a decision analysis is to decide what we measure to decide how good a decision is. The attribute is the quantity that we measure to describe how preferable a decision is. Common attributes would be financial gain in a business setting or whether an individual recovers from surgery in a medical setting. The consequence is the realised value of an attribute after making a decision.

When eliciting a utility function for an uncertain consequence, $c(\cdot, \cdot)$, it is much easier to consider trade-offs under certainty. We therefore elicit the utility function as a function of consequences c rather than decisions $\mathbf{x}$ and unknown states $\boldsymbol{\theta}$. When finding the optimal decision, we revert to consequences being uncertain; that is, replace c by $c(\mathbf{x}, \boldsymbol{\theta})$. We then calculate $\mathbf{x}^*$ by optimising $U(\mathbf{x})$ (Equation (2.7)) with respect to the decision, $\mathbf{x}$.

A basic elicitation procedure is as follows. We assume that $u(c)$ is monotonically increasing in c.

1. Determine $[c^0, c^*]$; the range for the attribute of interest. Assume $c^0 \prec c^*$.

2. Set $u(c^*) = 1 - u(c^0) = 1$.

3. Elicit utilities $\{u_1, \ldots, u_n\}$ for a set of intermediate consequences $\{c_1, \ldots, c_n\}$.

4. Fit a utility function to the quantities $\{(c^0, 0), (c_1, u_1), \ldots, (c_n, u_n), (c^*, 1)\}$. This can be done via a least squares procedure.

5. Check for consistency; ask the DM a few verification questions.

The above recipe assumes a given functional form for $u(\cdot)$; choosing this functional form can be tricky. Further, assigning utility values to intermediate values is not trivial. It is also important to be aware of, and counter, the DM's inaccuracies and biases. If $u(c)$ is not monotonically increasing in c, the problem can frequently be rephrased so that $u(c)$ is monotonically increasing in c. For example, replacing c by $-c$ when $u(c)$ is monotonically decreasing, or replacing c by $(c - c')^2$ if $u(c)$ is increasing on $[c^0, c']$ then decreasing on $(c', c^*]$.

The main step in the above procedure is step 3: eliciting utilities for intermediate consequences. The number n of utilities to be elicited will be a trade off between the accuracy of the resulting utility function and the time available.

There are two main ways to elicit u_i in step 3. The first is *probability equivalence*. Here the DM must specify the probability p where $[c^*, c^0; p] \sim c$ for a pre-determined consequence c. The notation $[c', c''; p] \sim c$ describes a lottery where c' occurs with probability p, c'' occurs with probability $1 - p$, and the uncertain result of this lottery is as desirable as the certain consequence c; that is $U([c', c''; p]) = U(c)$. The other way is to specify one of $\{c, c', c''\}$ when p and the other two elements of $\{c, c', c''\}$ are determined. This is a *certainty equivalence*.

Within each of the two main elicitation approaches, there are choices about how to specify probabilities and consequences. This concerns the choices of the multiple c_i in the (c_i, u_i) pair, or choosing related values of p when constructing many certainty equivalences. Each of these has limitations, which are typically forms of biases, as well as advantages; see Section 10.2.2 and Section 10.2.4 of González-Ortega *et al.* (2018), as well as references within for a discussion of the various biases that can occur when eliciting utilities and how to counteract them.

2.4.1 *Multi-attribute utility elicitation*

When eliciting a multi-attribute utility function (where the consequence space has 2 or more dimensions), it is cognitively complex for the decision maker to simultaneously weigh up the relative merits of many consequences. For this reason, much like probability elicitation, the problem can be greatly simplified when certain notions of independence are assumed. The notion of independence we need is known as *preferential independence*. Two (collections of) attributes C_i and C_j, with $C_i \cap C_j = \emptyset$ are said to possess preferential independence if

$$(c_i, c_j) \succeq (c_i', c_j) \implies (c_i, c_j') \succeq (c_i', c_j')$$

for some c_j and all c_j'. We have introduced the symbol $\succeq$ which is similar to $\preceq$: $p \succeq q \iff q \preceq p$. This type of independence allows the DM to specify judgements of the form *"provided everything else is constant, I prefer X to Y"*. This allows us to elicit individual utility functions for each attribute C_i and then combine them to provide an 'overall' utility function for all m consequences, $C = C_1 \times C_2 \times \ldots \times C_m$. If preferential independence holds for all possible sets i and j then the DM views the consequences as *mutually preferentially independent*. This is sometimes referred to as having *mutually utility independent* consequences. Now, assuming that consequences are mutually utility independent, there are two main ways to proceed. The first is to elicit a utility function which is a weighted sum of the marginal utility functions:

$$u(c) = \sum_{i=1}^{m} w_i u_i(c_i). \tag{2.8}$$

This is appropriate when the DM's preferences between lotteries depends only on the marginal distributions over the C_i and not their joint distribution. If preferences depend on the joint distribution, then we should use a multiplicative

form:

$$\begin{aligned} u(c) = &\sum_{i=1}^{m} w_i u_i(c_i) \\ &+ k \sum_{i=1, j>i}^{m} w_i w_j u_i(c_i) u_j(c_j) \\ &+ \dots \\ &+ k^{m-1} w_1 w_2 \dots w_m u_1(c_1) u_2(c_2) \dots u_m(c_m) \end{aligned} \tag{2.9}$$

where w_i are weights to be elicited and $1 + k = \prod(1 + kw_i)$. The case $k = 0$ corresponds to the additive utility function in Equation (2.8).

In the case of Equation (2.8) we can elicit the weights, w_i, by considering the consequence $c^0_{(i)} = (c^0_1, c^0_2, \dots, c^0_{i-1}, c^*_i, c^0_{i+1}, \dots, c^0_m)$. $c^0_{(i)}$ is the consequence where all attributes are at their worst value, apart from the ith attribute which is at its best value. It is simple to verify that $u(c^0_{(i)}) = w_i$. This gives rise to an elicitation protocol where we cycle over the $c^0_{(i)}$ to obtain each w_i. Note that $\sum_{i=1}^{m} w_i = 1$ since the best consequence, c^*, satisfies $u(c^*) = 1$, and the worst consequence, c^0, satisfies $u(c^0) = 0$. This means only $m - 1$ statements are required to elicit the w_i; additional statements can be used for verification. If the multiplicative form is used, one extra question needs to be asked. For example, we could invert one of the $c^0_{(i)}$ and have all but the ith consequence at its best value, and have the ith consequence at its worst value.

2.4.2 *Choosing functional forms for the u_i*

One of the most important issues is choosing the *functional form* of the utility function. Two important features are: monotonicity (more profit is almost always preferred) and concavity (an assumption about the attitude to risk of the DM).

Suppose $C \in \mathbb{R}$ is a monetary consequence. For a lottery p there are two expectations to consider. The *expected utility* $\mathrm{E}_p[u(c)]$ and the expected (monetary) value $\mathrm{E}_p(c)$. The certainty equivalent, c_p is the amount of money the DM would place on a single play of the lottery; $u(c_p) = \mathrm{E}_p(u(c))$. Equivalently, $c_p = u^{-1}\left\{\mathrm{E}_p(u(c))\right\}$.

The risk premium is defined to be the difference between the expected value of the lottery, and its certainty equivalent. We can interpret this as the difference in the value of infinite plays of the lottery versus a single play:

$$\pi_p = \mathrm{E}_p(c) - c_p. \tag{2.10}$$

The sign of π_p is a result of the shape of u. u is concave $\iff \pi_p > 0$; in such a scenario the DM is described as 'risk averse'. An alternative phrasing of a risk averse attitude is preferring the average result from many plays of the lottery, to a single play of the lottery. A risk averse DM prefers a small deterministic payoff to a random payoff with larger expected value, but some chance of a very small payoff. This is common in financial situations. A risk prone DM ($\pi_p < 0$) will prefer the lottery to its expected consequence; a desperate individual may wish to spend their last £1 on a scratch card, since the prospect of winning thousands is much more appealing than the fact that they will probably lose their stake.

Of course, a decision maker does not have to be strictly risk prone or strictly risk averse. They could be risk neutral (in this case $u(x) = ax + b$ and $\pi_p = 0$) or they could have a varying attitude towards risk. For instance, suppose the DM is a student who is planning their revision strategy for a pass/fail exam. Suppose the pass mark is 60%. When considering their revision strategy, the DM is willing to do almost anything that will get them above the pass mark. However, the DM is not desperate to achieve very high marks (although a very high mark would bring an increased level of personal achievement, which does increase the DM's utility of such events). For events below the pass mark, the DM will be risk prone, for events above the pass mark the DM will be risk averse; this leads to a sigmoidal shape. This is an example of the *local* risk behaviour changing.

Choosing the best functional form from many candidate functional forms can be achieved by considering, for example, the sum of squares in the least-squares procedure used to find the parameters for any given functional form. Verification questions can also be used to see which functional forms are, or are not, consistent with the preferences of the DM. For example, we could ask the DM for c' such that $u(c') = u'$, where u' is not close to any of the previously elicited u_i. If u' is close to the fitted $u(c')$ then we would be comfortable with the fitted $u(c)$. If it is too far away, we should consider alternative functional forms, or even re-elicit the (c_i, u_i) pairs. We should also verify with the DM, once we have constructed and optimised $U(\mathbf{x})$, that $\mathbf{x}^*$ is a decision they would be willing to take. If $\mathbf{x}^*$ is not a decision that the DM is willing to take, this suggests that somewhere in the process there has been an error, or the DM has changed their preferences. In such

cases, we must go back to the start and re-elicit all attributes and functional forms.

2.5 *Discussion*

We have outlined some relevant background material to set the scene for the rest of this thesis.

We have introduced the Athena simulator, a stochastic point-process model used to make statements about the reliability of an offshore wind farm, with a view towards making decisions about the operations and maintenance of the wind farm. The Athena simulator is typical of many simulators within the wider area of reliability analysis as it is stochastic, expensive and relies on many uncertain parameters. This makes the Athena simulator a suitable example for extensive case studies within later chapters.

We then discussed one approach that a subjective Bayesian may use to elicit a (joint) probability distribution. This approach is known as SHELF. We discussed that elicitation is useful when there is a lack of relevant data, although it is not a trivial procedure. We must take great care to elicit an honest and faithful representation of the beliefs experts possess. SHELF promotes the use of sensitivity analysis to gauge the relative importance of each element of $\boldsymbol{\theta}$, in order to obtain the most useful belief specification about $\boldsymbol{\theta}$ in the time available.

Finally, we introduced some basic concepts of decision making, such as utility, as well as motivating subjective utility functions as ways to capture the preferences of a DM and the trade-offs that they must make in complex problems. We discussed some approaches to eliciting multi-attribute utility functions by comparing a given lottery to certain consequences. We discussed ways to combine given marginal utility functions into a multi-attribute utility function. We also discussed aspects of choosing the functional form of the marginal utility functions.

Chapter 3

Emulators

Computer models, or simulators, allow us to predict complex phenomena at little financial or ethical cost. In a wind farm setting, it would cost an extortionate amount of money and literally years of time to test how a real, full scale wind farm performs under a single set of parameter values. Now suppose an engineer wants to know what would happen if we tweaked the set up in some way, corresponding to a change of simulator parameters. This is could be a multi-million pound question. Computer models allow us to investigate many "what if" scenarios at a fraction of the time and cost of the real thing. There are also ethical concerns that computer models may address; poor wind farm design is problematic for wildlife and endangering human life (Bailey *et al.*, 2014; Pedersen & Ahsan, 2020). Athena does not address all of these concerns, but it certainly will get us a long way without costing us much money.

Although simulators such as Athena alleviate the financial and ethical burden associated to large scale physical experiments, they are often computationally costly. Most of the runs of Athena in this thesis take approximately 5 minutes. This timing is somewhat variable since different experiments were performed on different machines, and certain parameter settings take longer to run than others. Most of the work in this thesis concerns exploring the parameter space to inform decision making. We require tools to reduce this computational cost. We will now step away from Athena, and its use cases, to consider how we can use statistical methods to build fast approximations to complex computer models as a general task. That is, construct *emulators* for computer simulators.

3.1 *Gaussian process priors*

Formally, a Gaussian process (GP), is a stochastic processes such that any finite collection of variables from this process follows a multivariate Normal distribution. In this thesis, GPs are viewed as prior distributions for *functions*. If we want to specify a GP prior, for a function $f(\cdot)$, then we only need to specify a mean function,

$$\mu(\mathbf{x}) = \mathrm{E}\left\{f(\mathbf{x})\right\} \tag{3.1}$$

and a covariance function

$$C(\mathbf{x}, \mathbf{x}') = \mathrm{Cov}\left\{f(\mathbf{x}), f(\mathbf{x}')\right\} \tag{3.2}$$

where $\mathbf{x}$ and $\mathbf{x}'$ are inputs to the computer model. Usually $\mathbf{x}$, $\mathbf{x}' \in \mathbb{R}^k$ for some $k \geq 1$.

The purpose of the mean function is clear. $\mu(\mathbf{x})$ is a functional form and should provide a sensible global estimate for $f(\mathbf{x})$. What allows GPs to be more flexible than approaches such as linear models is the covariance function; covariance functions allow us to concisely specify complex structure such as highly nonlinear behaviour. For now we will just set $\mu(\mathbf{x}) = 0$ but we will return to more general forms for $\mu(\mathbf{x})$ soon.

3.1.1 *Covariance functions*

By definition, the covariance function describes the covariance between two function outputs. It is the covariance function that allows us to specify a prior over many possible functions, rather than just specific functional forms. Here we review some common covariance functions with an emphasis on the ones used in this thesis. For a longer discussion we direct the reader to Rasmussen & Williams (2006).

In general, covariance functions can be written as

$$C(\mathbf{x}, \mathbf{x}') = \sigma^2 r(\mathbf{x}, \mathbf{x}') \tag{3.3}$$

where σ^2 is a scale parameter and $r(\cdot, \cdot) \in [-1, 1]$ is a correlation function. The 'meat' of the covariance function is $r(\cdot, \cdot)$; this is the device that allows us to specify the (prior) behaviour of uncertain functions. The role of σ^2 is to transform the

correlation function from a unitless quantity to the same units (and scale) as $f(\mathbf{x})^2$. An important property of a correlation function is that it is a *positive semidefinite function*. This means, for any $(w_1, w_2, \ldots, w_n) \in \mathbb{R}^n$, and vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n \in \mathbb{R}^K$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} w_i w_j r(\mathbf{x}_i, \mathbf{x}_j) \geq 0. \tag{3.4}$$

This property is closely related to the notion of a positive semidefinite matrix, which are used to form covariance and correlation matrices. In particular, if $r(\cdot,\cdot)$ is a positive semidefinite function, then a matrix R with $R_{i,j} = r(\mathbf{x}_i, \mathbf{x}_j)$ is a positive semidefinite matrix.

First we consider a squared exponential covariance function. For a function $f(\cdot)$ with scalar input $x \in \mathbb{R}$, the squared exponential covariance function is given by

$$C(x, x') = \sigma^2 \exp\left\{-\frac{(x - x')^2}{\theta^2}\right\}. \tag{3.5}$$

The scale parameter σ^2 corresponds to a prior marginal variance. If we take $\sigma = 1$ and model $f(\cdot)$ as a GP then we believe that around 95% of function values lie in the range $(-2, 2)$. The covariance function is a scalar multiple of a correlation function. In the case of a squared exponential, the correlation function is $r(\mathbf{x}, \mathbf{x}') = \exp\left\{-\frac{(x-x')^2}{\theta^2}\right\}$. The parameter θ is a *lengthscale* parameter which describes how informative $f(x)$ is for $f(x')$ the (Euclidean) distance between inputs. Stationary covariance functions are those which depend only on $d(x, x') = |x - x'|$; we can see that this holds for the squared exponential covariance function thus a GP with squared exponential covariance is a stationary process (Gneiting, 2002).

As Figure 3.1 shows, for small values of θ the correlation between inputs decays very quickly — at an exponential rate. For larger θ, the correlation still decays exponentially, but at a reduced rate. Smaller θ values generate less "wiggly" functions. Several draws from a GP with $\theta = 0.1$, 0.5, 1 and 2 can be seen in Figure 3.2. Once $|x - x'| > 2\theta$ the function outputs are essentially uncorrelated. We have fixed $\sigma = 1$ in Figure 3.2, changing σ just changes the scale of the generated functions but not their behaviour. Note that, although small lengthscales lead to very wiggly functions, it can be shown that functions drawn from a GP with squared exponential covariance are infinitely mean square differentiable. This seems reasonable for the functions we will encounter later in this thesis. Note that,

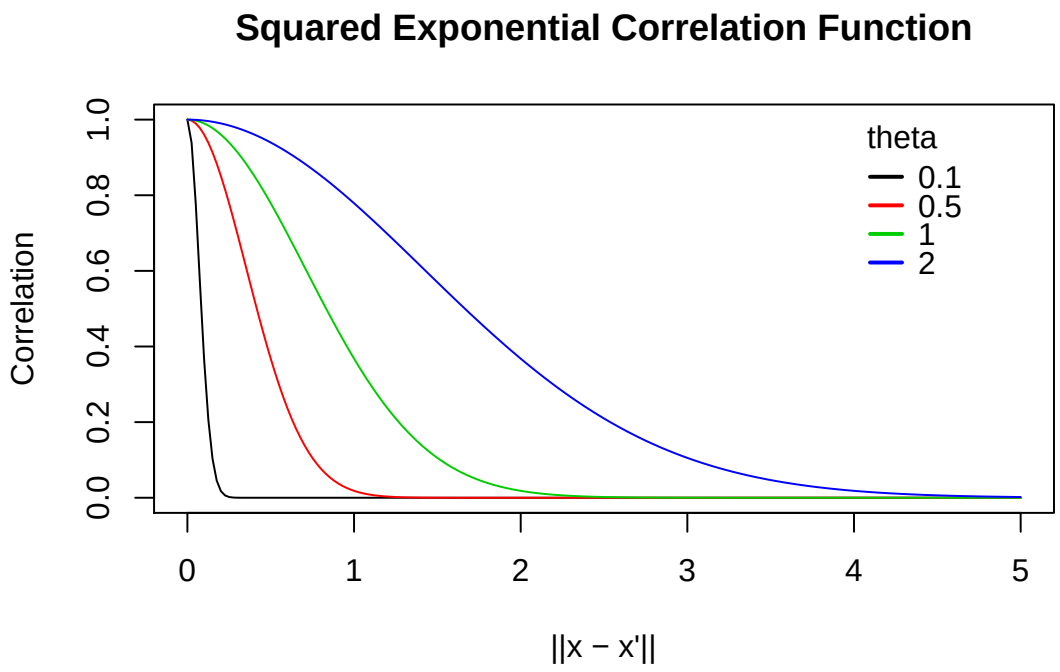

Figure 3.1: Plot showing how θ affects the rate of decay of the correlation between model outputs as a function of the distance between inputs. Correlation functions with smaller values of θ decays to zero faster than those with a larger value of θ.

although each function is drawn randomly, each realisation from a GP represents a deterministic function. It is as if we were to randomly draw parameters of a fixed functional form. The precise form is unknown, but not inherently stochastic. These "spaghetti plots" are great for visualising *joint* behaviour of function draws, but not very good at showing us the marginal behaviour. To do this, we can plot the prior mean and some chosen prior quantiles. In the case of a stationary covariance function, $C(x, x) = \sigma^2$ and because we assumed a zero mean GP the (prior) marginal behaviour is independent of x; $f(x) \sim \mathcal{N}(0, \sigma^2)$.

The squared exponential is a limiting case of the Matérn covariance function, which can be used to generate functions with a finite number of derivatives:

$$C(x, x') = \sigma^2 \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\sqrt{2\nu} \frac{|x - x'|}{\rho} \right)^{\nu} K_{\nu} \left(\sqrt{2\nu} \frac{|x - x'|}{\rho} \right) \tag{3.6}$$

where σ, ρ, $\nu > 0$ and K_{ν} is the modified Bessel function of the second kind. Functions drawn from this prior are differentiable $\lfloor \nu \rfloor$ times in the mean-square

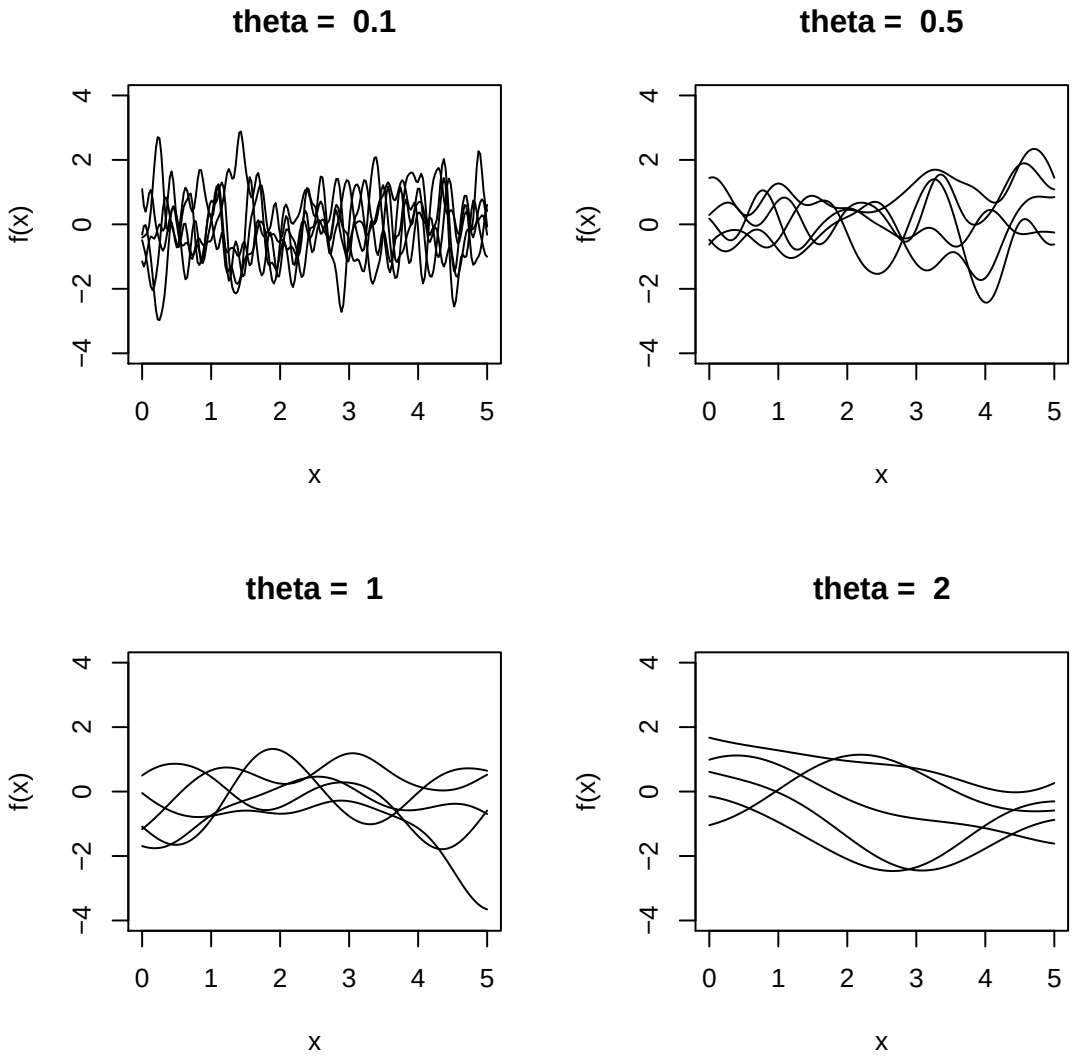

Figure 3.2: Draws from a GP prior with squared exponential covariance function ($\sigma^2 = 1$) for varying values of θ. Larger lengthscales correspond to less wiggly functions; their behaviour is less erratic.

sense. Letting $\nu \to \infty$ recovers the squared exponential covariance function. Like the squared exponential covariance function, the Matérn covariance function is a stationary covariance function.

Another common covariance function is the linear covariance function:

$$\begin{aligned} C(x, x') &= \begin{pmatrix} 1 & x \end{pmatrix} \begin{pmatrix} b_0 & b_1 \\ b_1 & b_2 \end{pmatrix} \begin{pmatrix} 1 \\ x' \end{pmatrix} \\ &= b_0 + b_1(x + x') + b_2 x x'. \end{aligned} \tag{3.7}$$

The linear covariance function depends explicitly on x and x', and not the distance between them. This makes it a non-stationary covariance function. We later show that the linear covariance function is strongly related to many useful mean functions.

The final covariance function we consider is the white noise covariance function. This is just

$$C(x, x') = \lambda^2 \mathbb{I}(x = x') \tag{3.8}$$

where $\mathbb{I}(X)$ is an indicator function of the form

$$\mathbb{I}(X) = \begin{cases} 1, & X = \texttt{TRUE} \\ 0, & \text{otherwise.} \end{cases} \tag{3.9}$$

This covariance function is differentiable precisely *nowhere*. It can be used to generate independent and identically distributed (iid) $\mathcal{N}(0, \lambda^2)$ random variables (RVs). It is not useful on its own for generating functions but does act as a useful device for modelling random variation exhibited by a stochastic simulator such as the Athena simulator.

In the context of emulation, λ^2, the variance of a random error term, is often called the 'nugget' variance (Gramacy & Lee, 2012). Throughout this thesis, we will use both σ^2 and λ^2 as variances of Normal distributions. Unless explicitly stated otherwise, σ^2 will be the marginal (prior) variance of $f(\mathbf{x})$ whereas λ^2 will be the nugget variance.

3.1.2 *Combining covariance functions*

For many practical problems, we are interested in varying multiple function inputs simultaneously. We can make multiple input covariance functions by combining standard ones. To construct a squared exponential covariance function for a K

input GP we multiply K univariate squared exponential covariance functions:

$$C(\mathbf{x}, \mathbf{x}') = \sigma^2 \exp\left\{-\sum_{i=1}^{K} \frac{(x_i - x_i')^2}{\theta_i^2}\right\}. \tag{3.10}$$

From hereon in, $\mathbf{x}$ is a $K \geq 2$ dimensional input vector. Whenever we use x this corresponds to a scalar input. This covariance function will take a large value when all the pairwise distances are small; it takes just one of the differences to be very large for the simulator outputs to be modelled as almost uncorrelated. If $\theta_i = \theta$ for each i this is an *isotropic* covariance function. Otherwise, the covariance function is *anisotropic*.

Adding covariance functions creates functions with different layers of complexity. One thing we might want to do is construct a function with a linear trend but allow for some variability around this trend. This is constructed by adding linear and (say) squared exponential covariance functions together:

$$C(\mathbf{x}, \mathbf{x}') = \sigma^2 \exp\left\{-\sum_{i=1}^{K} \frac{(x_i - x_i')^2}{\theta_i^2}\right\} + h(\mathbf{x})Bh(\mathbf{x})^T \tag{3.11}$$

where $h(\mathbf{x}) = (1, \mathbf{x}^T)$. It can be seen in Figure 3.3 that the resulting functions are roughly linear, but also feature some non-linear variation. In a regression context, this is useful when the data being modelled have an approximately linear trend but also some variation which is more difficult to describe.

3.1.3 *Mean functions*

The simplest mean function is $\mu(\mathbf{x}) = 0$ which corresponds to the GP being "all covariance". GPs with a prior mean of 0 are able to capture complex function behaviour. Generalising to $\mu(\mathbf{x}) = \beta$ centres the predictions around a more useful value; the maximum likelihood estimate for β is $\hat{\beta} = \bar{y}$ (Sacks *et al.*, 1989). But again, this is a pure covariance approach after centring the data. It can be more useful to use mean functions which depend on $\mathbf{x}$ and then use a zero mean GP to model variation about this systematic mean function. That is, model the simulator output as

$$f(\cdot) \sim \mathcal{GP}\{\mu(\cdot), C(\cdot, \cdot)\}. \tag{3.12}$$

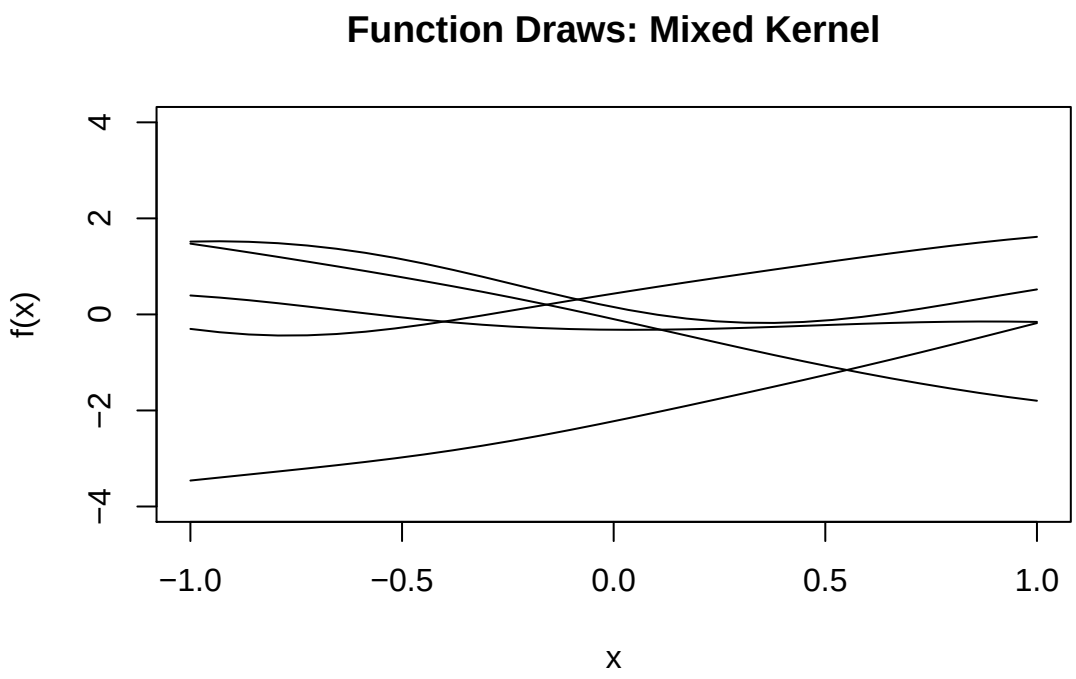

Figure 3.3: Some realisations of a GP where the covariance structure is constructed by the addition of a squared exponential covariance function with a linear covariance function, as in Equation (3.11).

This requires a specification for $\mu(\cdot)$. An precise form may be difficult to specify (which is in part why we are using a GP), thus a popular approach is to construct the mean function hierarchically;

$$\mu(\mathbf{x}) = h(\mathbf{x})^T \boldsymbol{\beta} \tag{3.13}$$

where $h(\mathbf{x})$ is a p-vector with each element taking a simple, deterministic form and $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_{p-1})^T$ are unknown regression coefficients to be inferred. A common form is $h(\mathbf{x}) = (1, \mathbf{x}^T)$. This expresses the belief that the function is approximately linear in its inputs. Some authors are proponents of using a very simple mean functions (constant, or zero), allowing the covariance structure to do the heavy lifting (Henderson *et al.*, 2009; Zhang *et al.*, 2021). Some prefer a complex mean function which accounts for a large amount of variability in $f(\cdot)$ and using the GP to mop up the residuals. Providing the chosen mean function is sensible, this offers robustness against an inappropriate choice for $C(\cdot, \cdot)$ and can

make the emulator more interpretable (Xu *et al.*, 2016; Vernon *et al.*, 2010). This thesis generally takes a Goldilocks approach; the mean function should aim to capture the main global trends in $f(\mathbf{x})$ whilst retaining simplicity. This can require much less effort than using a complex mean function but, despite the simplicity, can be very effective (Kennedy & O'Hagan, 2001; Fricker *et al.*, 2013; Fisher *et al.*, 2022). A different approach is to choose a complex mean form but use a prior that encourages sparsity amongst $\boldsymbol{\beta}$ to automatically choose a mean function (Seshadri *et al.*, 2020).

3.2 *A general Gaussian process posterior*

3.2.1 *Parameter inference*

Inferring the coefficients of the mean function and the parameters of the covariance function can be done in a frequentist or Bayesian way. If useful prior information is available, Oakley (2002) describes how we might elicit $\mu(\cdot)$ and $C(\cdot,\cdot)$. Suppose we have some data from the simulator. In this case, data are a set of simulator inputs and corresponding outputs. That is $\mathcal{D} = \{(y_i, \mathbf{x}_i), i = 1, 2, \ldots, n\}$ where $y_i = f(\mathbf{x}_i) + \varepsilon(\mathbf{x}_i)$ and $\varepsilon(\mathbf{x}_i) \sim N(0, \lambda^2(\mathbf{x}_i))$ models inputs dependent noise in a stochastic simulator. Commonly, the inputs $\mathbf{x}_i$ are collected into a matrix X where the ith row of X is $\mathbf{x}_i$. Note that setting $\lambda^2(\mathbf{x}) = \lambda^2$ recovers a constant level of noise and $\lambda^2(\mathbf{x}) = 0$ recovers a deterministic function. For now we will assume that $\lambda^2(\mathbf{x}) = \lambda^2 \geq 0$. Even if theoretically we should take $\lambda = 0$, because the simulator is deterministic, it often is computationally convenient to set λ to be some small value (say $\lambda = 10^{-6}$) to avoid computational issues surrounding matrix inversion. This is just a 'trick' to brush an ill-conditioned matrix under the carpet. Some authors support estimating λ even when the simulator is deterministic as a way to account for assumptions such as stationary or the choice of covariance being incorrect (Gramacy & Lee, 2012). This approach is not infallible; Andrianakis & Challenor (2012) show this can lead to problems such as (i) bimodal likelihoods and (ii) ill-fitting emulators. Another method to avoid the problems induced by attempting to invert an ill-conditioned matrix is to use an approximate covariance matrix in place of the true covariance matrix. This might be done by using just a subset of the training data which may lead to a covariance matrix with better conditioning (Wilson & Nickisch, 2015). Yet another approximate method is a Hilbert space

approximation, where a covariance function is expressed as an infinite sum, and the approximation stems from truncating the sum; this leads to approximating the GP by a linear model with basis functions (regression functions) derived from the covariance structure (Solin & Särkkä, 2020; Riutort-Mayol *et al.*, 2023).

Regardless of whether the simulator is stochastic or deterministic, the likelihood function is then

$$\mathbf{y} \mid \boldsymbol{\beta}, \Theta \sim \mathcal{N}\left\{H\boldsymbol{\beta}, \Sigma\right\} \tag{3.14}$$

where the mean is constructed via multiplication of H, the design matrix with jth row $h(\mathbf{x}_j)^T$ where $\mathbf{x}_j$ is the jth vector of simulator inputs, and the coefficient vector $\boldsymbol{\beta}$. The covariance matrix is $\Sigma = C(X, X) + \lambda^2 I$, where I is the identity matrix (of appropriate dimensions), Θ are the parameters of the covariance function and $C(X, X)$ is a matrix with $C(X, X)_{i,j} = C(\mathbf{x}_i, \mathbf{x}_j)$. Assuming a squared exponential covariance function, and a nugget term, leads to $\Theta = (\sigma, \theta_1, \theta_2, \ldots, \theta_K, \lambda)$. In a frequentist framework, we can maximise the likelihood function to obtain estimates $\hat{\boldsymbol{\beta}}$ and $\hat{\Theta}$ (Sacks *et al.*, 1989). In a Bayesian framework we could elicit a prior $\pi(\boldsymbol{\beta}, \Theta)$ then use numerical methods to obtain either exact or approximate samples from $\pi(\boldsymbol{\beta}, \Theta \mid \mathcal{D})$ (Svalova *et al.*, 2021). Alternatively, we can maximise the posterior density to obtain *maximum a posteriori* (MAP) estimates (Baker *et al.*, 2020*b*). It is desirable, for computational reasons, to analytically integrate out parameters where possible. If we take $\boldsymbol{\beta} \sim \mathcal{N}(\boldsymbol{b}, B)$ this is possible. Following Gelman *et al.* (2013) (see appendix A of the reference), and noting that $\text{Var}\{H\beta\} = HBH^T$, it can be shown that we can write the distribution of $\mathbf{y} \mid \Theta$ as

$$\mathbf{y} \mid \Theta \sim \mathcal{N}\left\{H\boldsymbol{b}, \Sigma + HB^T H\right\}. \tag{3.15}$$

Now inference about Θ follows by treating Equation (3.15) as the sampling distribution for $\mathbf{y}$. This reduced parameter space can lead to faster inferences. From this we can obtain MLE/MAP estimates or perform a full Bayes analysis. Note that if we take $\pi(\boldsymbol{\beta}, \sigma \mid \boldsymbol{\theta}, \lambda) \propto \sigma^{-2}$ (i.e. implying infinite prior variance on $\boldsymbol{\beta}$) then both $\boldsymbol{\beta}$ and σ can be analytically integrated out. The resulting emulator is a Student's t process (Oakley & O'Hagan, 2002). The Student's t process arises only from certain families of prior specification, thus a GP is often preferred.

3.2.2 *Posterior predictive distribution*

A GP prior on $f(\cdot)$, $\pi(f(\cdot))$, is infinite dimensional, as is the posterior, $\pi(f(\cdot) \mid \mathcal{D})$. Both of these facts are due to there being an infinite number of choices for $\mathbf{x}$. Luckily, this issue can be sidestepped. If we have a collection of $m \in \mathbb{N}$ new inputs, X^*, and then let

$$f(X^*) = \begin{pmatrix} f(\mathbf{x}_1^*) \\ f(\mathbf{x}_2^*) \\ \vdots \\ f(\mathbf{x}_m^*) \end{pmatrix} \tag{3.16}$$

be the vector of (unknown) outputs at the collection of new inputs. Then we can easily find $\pi(f(X^*) \mid \mathcal{D})$. In practice, this is enough to allow us to perform any task that we would use $f(\cdot)$ for.

Now assume a mean function of the form $\mu(\mathbf{x}) = h(\mathbf{x})^T\boldsymbol{\beta}$. Consider a collection of simulator outputs $\mathbf{y}^{(1)} = (y_1^{(1)}, y_2^{(1)}, \ldots, y_n^{(1)})^T$ at inputs

$$X^{(1)} = \begin{pmatrix} \mathbf{x}_1^{(1)} \\ \mathbf{x}_2^{(1)} \\ \vdots \\ \mathbf{x}_n^{(1)} \end{pmatrix} \tag{3.17}$$

and let the corresponding design matrix be H_1. Our objective will be inference about $\mathbf{y}^{(2)} = (y_1^{(2)}, y_2^{(2)}, \ldots, y_m^{(2)})^T$, the outputs at $X^{(2)}$ with corresponding design matrix H_2. After integrating out $\boldsymbol{\beta} \sim \mathcal{N}\{\boldsymbol{b}, B\}$, we are left with the following relationship between the two sets of simulator outputs:

$$\begin{pmatrix} \mathbf{y}^{(1)} \\ \mathbf{y}^{(2)} \end{pmatrix} \mid \Theta \sim \mathcal{N}\left\{ \begin{pmatrix} H_1\boldsymbol{b} \\ H_2\boldsymbol{b} \end{pmatrix}, \begin{pmatrix} K(X^{(1)}, X^{(1)}) & K(X^{(1)}, X^{(2)}) \\ K(X^{(2)}, X^{(1)}) & K(X^{(2)}, X^{(2)}) \end{pmatrix} \right\}. \tag{3.18}$$

where $K(X^{(i)}, X^{(j)}) = C(X^{(i)}, X^{(j)}) + H_i B H_j^T + \text{diag}\{\lambda^2 \mathbb{I}(i = j)\}$. Unless explicitly stated, $C(\cdot, \cdot)$ is used for squared exponential covariance functions and $K(\cdot, \cdot)$ represents the addition of a squared exponential, linear and white noise covariance functions (possibly with $\lambda^2 = 0$).

Then the posterior predictive distribution for $\mathbf{y}^{(2)}$ is also multivariate Normal

(Rasmussen & Williams, 2006). The mean and variance are as follows:

$$\mathrm{E}\left\{\mathbf{y}^{(2)} \mid \Theta, \mathcal{D}\right\} = H_2\boldsymbol{b} + K(X^{(2)}, X^{(1)})K(X^{(1)}, X^{(1)})^{-1}\left(\mathbf{y}^{(1)} - H_1\boldsymbol{b}\right) \tag{3.19}$$

$$\mathrm{Var}\left\{\mathbf{y}^{(2)} \mid \Theta, \mathcal{D}\right\} = K(X^{(2)}, X^{(2)}) - K(X^{(2)}, X^{(1)})K(X^{(1)}, X^{(1)})^{-1}K(X^{(1)}, X^{(2)}). \tag{3.20}$$

Observing the form of the equations we can see that the posterior mean is just the prior mean with an adjustment based on how far the prior mean, $H_1\boldsymbol{b}$, is from the observed outputs, $\mathbf{y}_1$, and how correlated we expect $\mathbf{y}^{(1)}$ and $\mathbf{y}^{(2)}$ to be. The posterior variance is always *less than* the prior variance and, surprisingly, is independent of the observed $\mathbf{y}^{(1)}$. The posterior variance depends only on the locations of simulator inputs we are interested in and their distances from the design points. However, if Θ is inferred, rather than specified, then there is a dependence on $\mathbf{y}^{(1)}$ via the likelihood function. Throughout this thesis, we will often refer to the "conditional Normal equations" which means equations with the same form as Equation (3.19) and Equation (3.20). For brevity, we will denote the posterior mean at input $\mathbf{x}$ by $m^*(\mathbf{x})$ and posterior variance at $\mathbf{x}$ by $v^*(\mathbf{x})$.

3.2.3 *Specific examples*

We will now observe how aspects of a GP prior impact the posterior. We will focus on squared exponential, linear, and white-noise covariances. We will fit an emulator to a simple, tractable example to allow us to easily compare the fit to the truth. We will work with

$$f(x) = 2\sin(2\pi x) + \cos(3\pi x) + 3x. \tag{3.21}$$

In a more realistic scenario, the simulator is described by complex mathematical equations and implemented via many lines of computer code, thus the exact functional form is not known. We will assume a linear trend with prior mean $\mathrm{E}(\boldsymbol{\beta}) = \mathbf{0}$ and $\mathrm{Var}(\boldsymbol{\beta}) = 2I_2$ where I_k is the $k \times k$ identity matrix. Using $n = 10$ design points chosen by uniform sampling and a squared exponential covariance function with $\sigma = 3$ and $\theta = 0.5$ we can calculate the posterior distribution for any x via the conditional Normal equations.

An interesting feature of our prior (Figure 3.4) is that the width of the prior predictive interval depends on x. Some interesting features of GP posteriors are:

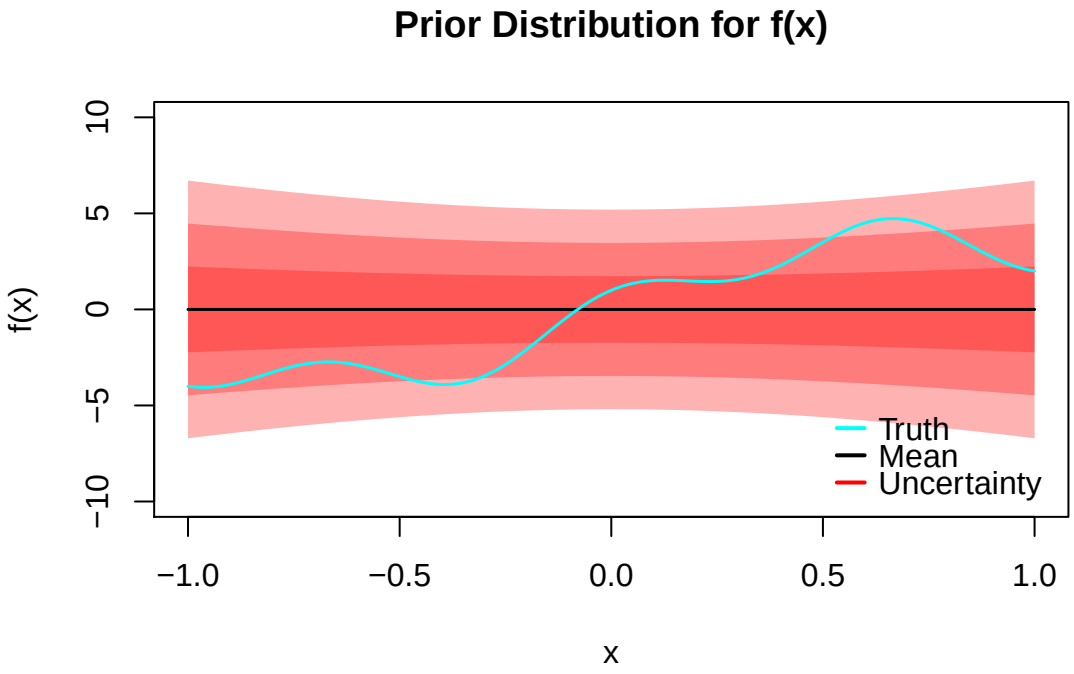

Figure 3.4: Our prior specification for the simulator specified by Equation (3.21). The simulator is shown as the blue line, emulator prior mean as the black line and uncertainty is represented by the red bands; $\mu(x) \pm (1, 2, 3)\sqrt{K(x, x)}$.

(i) $v^*(x) = 0$ and $m^*(x) = f(x) \iff x$ is a design point and $\lambda = 0$.

(ii) $v^*(x)$ is small when x is close to design points.

(iii) When x is far from the design points, $m^*(x) \approx h(x)^T \mathrm{E}\{\boldsymbol{\beta} \mid \mathcal{D}\}$.

(iv) When x is far from the design points, $v^*(x) \approx K(x, x)$.

A "proof" by picture of these properties is given in the the plotted posterior distribution of the synthetic example (Figure 3.5). The uncertainty is very small close to the training points and the mean function is close to the function values (a computational nugget $\lambda^2 = 10^{-6}$ was used here), this illustrates (i) and (ii). When we are far from the training data, the width of our posterior uncertainty bands is close to that of our prior uncertainty bands. Figure 3.5 (left) does not clearly illustrate properties (iii) and (iv). Zooming out gives the right hand plot and shows shows that the expected response is linear, just like the prior mean function, when x is far from the design points.

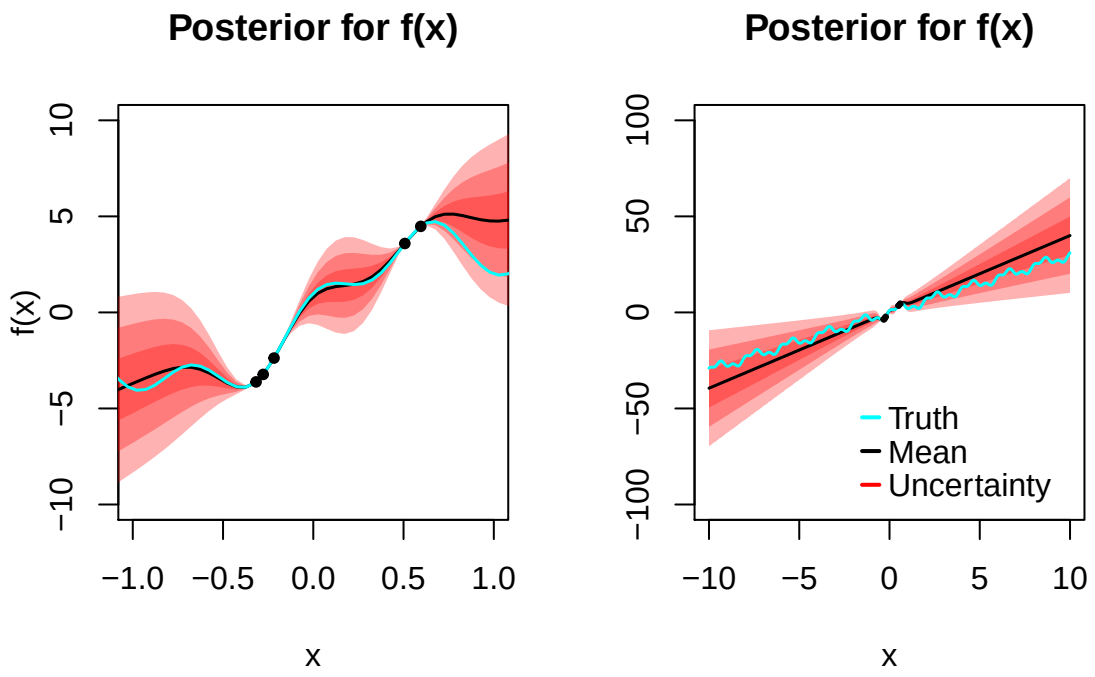

Figure 3.5: The posterior distribution for the simulator specified by Equation (3.21). The simulator is shown as the blue line, the emulator posterior mean as the black line and posterior uncertainty is represented by the red bands; $m^*(x) \pm (1, 2, 3)\sqrt{v^*(x)}$. The left hand plot shows how the emulator captures local variation in $f(\cdot)$ whereas the right hand plot illustrates the estimated global variation.

3.2.4 *Stochastic computer simulators*

It is straightforward to extend emulators to stochastic computer models. That is, those exhibiting random variation. A feature of these models is that runs of the simulator return different values even if $\mathbf{x}$ remains unchanged. Our motivating example, Athena, is stochastic. This approach to simulation allows us to investigate two major sources of uncertainty:

(i) the stochastic nature of the world (aleatory uncertainty)

(ii) randomising inputs to account for lack of knowledge (epistemic uncertainty).

Aleatory uncertainty is a desirable feature of models where the response is ultimately difficult to predict. For example, in the Athena simulator, aspects of the simulator are random to account for difficult to predict features such as weather

and human error. Accounting for epistemic uncertainty is achieved by assigning a probability distribution to a some (or all) input parameters and propagating this uncertainty through $f(\cdot)$, usually with the help of an emulator (Oakley & O'Hagan, 2004).

Under stochasticity we need to be careful about what we want to infer, and hence we will be careful with our notation. Let $y(\mathbf{x})$ be the simulator output at $\mathbf{x}$ and let $f(\mathbf{x})$ be the mean response at $\mathbf{x}$. Under the GP framework, we perform inference on f and y simultaneously. We can specify their relationship hierarchically,

$$y(\mathbf{x}) \mid f(\mathbf{x}), \Theta, \boldsymbol{\beta} \sim \mathcal{N}(f(\mathbf{x}), \lambda^2) \tag{3.22}$$

$$f(\cdot) \mid \Theta, \boldsymbol{\beta} \sim \mathcal{GP}\{h(\cdot)^T\boldsymbol{\beta}, C(\cdot,\cdot)\}. \tag{3.23}$$

Now, since iid $\mathcal{N}(0, \lambda^2)$ noise is equivalent to the white noise covariance structure, we can integrate out $f(\cdot)$ and $\boldsymbol{\beta}$. This leads to

$$y(\cdot) \mid \Theta \sim \mathcal{GP}\left\{h(\cdot)^T\boldsymbol{b}, K(\cdot,\cdot) + \lambda^2 I\right\} \tag{3.24}$$

where $h(\cdot)$ and $K(\cdot,\cdot)$ have their usual meanings. Note the explicit addition of the measurement error/nugget term to $K(\cdot,\cdot)$, we have intentionally separated out the stochasticity to clarify the different between stochasticity and epistemic uncertainty. Prediction of the mean function is exactly the same as in Equation (3.19). Posterior variances of $y(X^*)$ and $f(X^*)$ differ only by λ^2:

$$\text{Var}\left(f(X^*) \mid \mathcal{D}, \Theta\right) = K(X^*, X^*) - K(X^*, X)V^{-1}K(X, X^*) \tag{3.25}$$

$$\text{Var}\left(y(X^*) \mid \mathcal{D}, \Theta\right) = K(X^*, X^*) + \lambda^2 I - K(X^*, X)V^{-1}K(X, X^*). \tag{3.26}$$

where $V = K(X, X) + \lambda^2 I$. Equation (3.25) allows us to make statements about $f(\mathbf{x})$ and Equation (3.26) allows us to make statements about $y(\mathbf{x})$. The difference between these two equations is analogous to the difference between a confidence interval and a prediction interval in a frequentist regression analysis. The properties (i) and (ii) of GP posteriors do not hold here because λ may be arbitrarily large to account for arbitrary amounts of stochasticity. Good prior information, usually specified via the mean function, can be very important in the stochastic setting since, especially if λ is large, $y(X)$ might not be very informative for $f(X)$. We demonstrate this by adapting Equation (3.21) to be stochastic. We introduce

stochasticity by simulating from $y(\mathbf{x}) \sim \mathcal{N}\{f(\mathbf{x}), 0.5^2\}$. This can be seen in Figure 3.6. There is still a fairly large amount of uncertainty close to the design points. An infinite sample size is required for this uncertainty to reduce to zero. The probability bands for $y(x)$ take into account uncertainty about $f(x)$ as well as stochasticity.

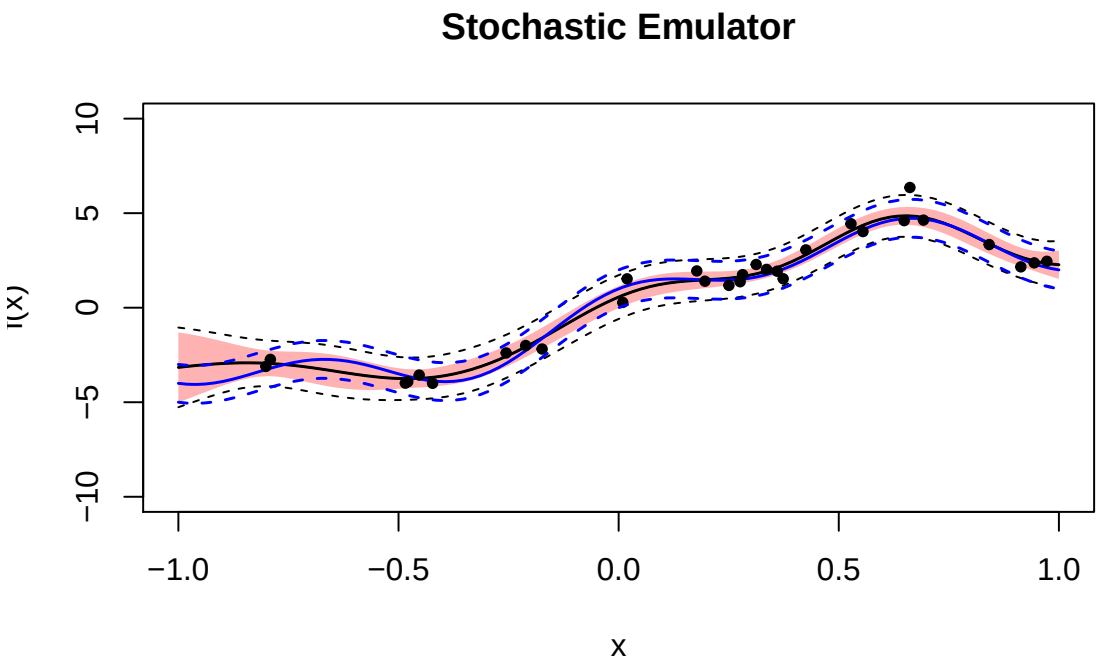

Figure 3.6: An emulator for a stochastic simulator. Solid blue line is simulator mean with blue dashed lines being 95% probability bands. Solid black line is emulator mean, dashed black lines are a 95% probability interval for $y(x)$ and the red band is a 95% probability interval for $f(x)$.

3.3 *Design of computer experiments*

The previous figures in this chapter used a simple uniform sample for selecting locations at which to run the simulator. In practice, this is usually a bad idea since randomly generated designs can often lead to design points being very close together in space. This is wasteful since our smoothness assumptions about $f(\cdot)$ imply that $|f(\mathbf{x}) - f(\mathbf{x}')|$ is small whenever $||\mathbf{x} - \mathbf{x}'||$ is small. It therefore makes sense to spread out the design points where possible. Such designs are termed

"space filling" designs. Another design approach is to sequentially construct the emulator. This involves using the emulator, combined with a rule, $\alpha(\mathbf{x})$, to choose where to next run the simulator. The simulator is then run at the point satisfying $\mathbf{x}^\star = \arg\max_{\mathbf{x}\in\mathcal{X}} \alpha(\mathbf{x})$, the emulator is updated and we repeat the process until we satisfy some stopping rule. In general, there is no globally superior choice as to whether we go with one-shot or sequential designs, and a sequential design may be initialised by a small one-shot design (Zhang *et al.*, 2021). One-shot designs are typically easier and cheaper to implement than sequential designs. We construct the design and run the simulator in isolation of each other. Sequential designs need the simulator and emulator to 'talk' to each other, which might be difficult if the person constructing the emulator is not familiar with the language the simulator is implemented in. Finally, one-shot designs are agnostic to choice of covariance and mean function. This can be useful in exploratory situations where little is known about the underlying covariance function for $f(\cdot)$. For instance, Overstall & Woods (2017) use a single one-shot design to construct and compare many emulators based on just one training set and one validation set. On the other hand, sequential designs can leverage our current understanding of $f(\cdot)$ (or $y(\cdot)$) to achieve specific goals more efficiently that a one-shot approach. A popular class of sequential designs, Bayesian Optimisation acquisition functions (Frazier, 2018), are those which aim to optimise $f(\cdot)$ whilst minimising the number of times we run the simulator. In essence, a one-shot design is a Jack of all trades, master of none, a sequential design is a master of only one.

3.3.1 *One-shot designs*

Perhaps the most popular one-shot designs in the computer experiments literature are Latin Hypercube (LH) designs. A LH, of size n and dimension K, is constructed by partitioning the sample space into n^K equally sized K-dimensional hypercubes and a single uniform sample is taken within each sub-hypercube. This leads to a 'more uniform' design than truly uniform sampling. This construction allows us to guarantee that for a LH of size n, in each margin, there is exactly 1 sample in each interval. This shows a clear advantage over pure uniform designs. In Figure 3.7 we see a particularly unfortunate uniform design in which no samples appear above the line $x_1 = x_2$ and most of the points satisfy $x_2 < 0.2$. The LH offers some protection against this, but is not watertight. For example,

sampling amongst the diagonal boxes would lead to a valid LH design, but not a space filling design. We can enhance LHs with additional structure to obtain a

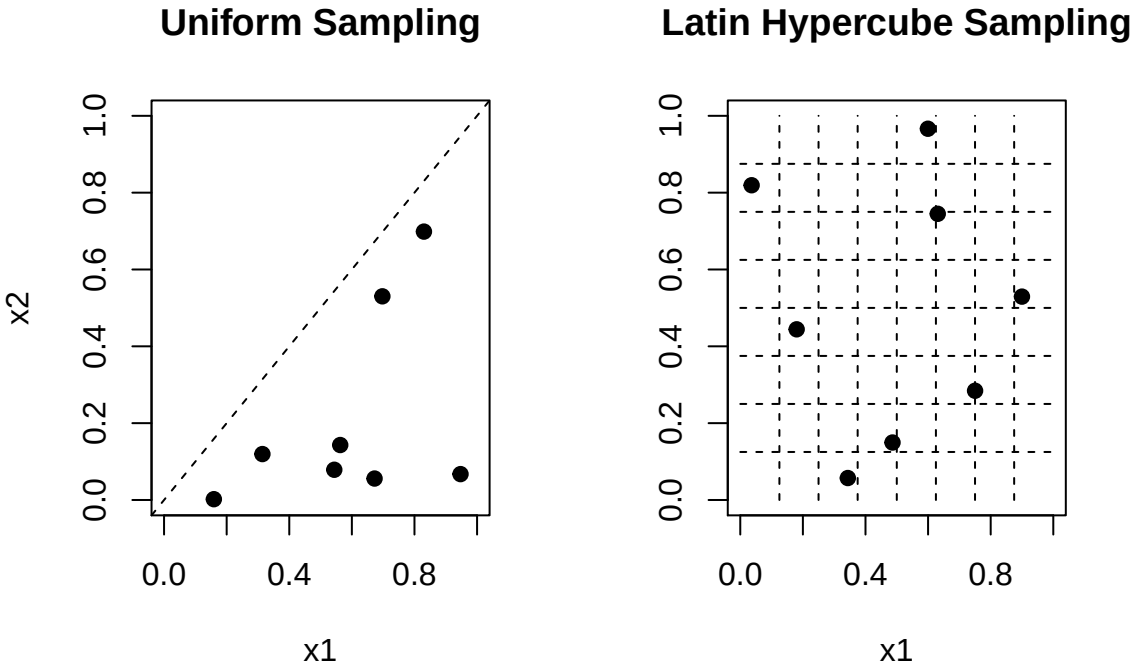

Figure 3.7: A 2d random sample (left) and 2d LH sample (right), both of size $n = 8$. In the left hand plot the dashed line corresponds to unit diagonal. In the right hand plot, the dashed lines correspond to an 8×8 grid of equally sized sub-cubes.

more space filling design. One such enhancement is the maximin LH. A maximin design is one which maximises the minimum distance between design points so that no pair of points are too close. This rule can be added to a LH to prevent the points sitting on the diagonal or for two points to be sitting in nearby corners of adjacent sub-cubes (McKay *et al.*, 1979; Morris & Mitchell, 1995).

3.3.2 *Sequential designs*

As discussed, sequential designs allow us to leverage our current state of knowledge to satisfy some optimality criterion. Many such designs are Bayesian since they can be expressed as expectations of random variables. For example, consider Active Learning McKay (ALM), which is based on posterior variance (and recall, the variance is the expected squared distance from the mean) (Seo *et al.*, 2000).

This method is referred to as "uncertainty sampling" in the machine learning literature (Nguyen *et al.*, 2022). If we have already run the simulator at n locations, then the next point to run the simulator at, $\mathbf{x}_{n+1}$, is given by

$$\mathbf{x}_{n+1} = \arg\max_{\mathbf{x}\in\mathcal{X}} \alpha_{ALM}(\mathbf{x}) \tag{3.27}$$

$$\alpha_{ALM}(\mathbf{x}) = v^*(\mathbf{x}). \tag{3.28}$$

That is, we run the simulator at the location with largest uncertainty (about the underlying mean, so the variance is given by Equation (3.25)) in the design space $\mathcal{X}$. The idea here is to reduce uncertainty to construct a globally accurate emulator. An approach with similar goals is to minimise the integrated mean square prediction error (IMSPE), which is given as

$$\alpha_{IMSPE}(\mathbf{x}) = -\int_{\tilde{\mathbf{x}}\in\mathcal{X}} \{y(\tilde{\mathbf{x}}) - m^*(\tilde{\mathbf{x}})\}^2 \, \mathrm{d}\tilde{\mathbf{x}}. \tag{3.29}$$

IMSPE aims to minimise the average uncertainty across $\mathcal{X}$ rather than point-wise uncertainty. This seems like a better idea than ALM, but requires integration over $\mathcal{X}$ which is difficult (Gramacy, 2020). A simple workaround is to approximate the integral by a sum:

$$\alpha_{IMSPE}(\mathbf{x}) \approx -\sum_{\tilde{\mathbf{x}}\in X} \{y(\tilde{\mathbf{x}}) - m^*(\tilde{\mathbf{x}})\}^2, \tag{3.30}$$

where X is a collection of design points at which the simulator has been run. Typically, sequential designs satisfying criteria such as Equation (3.28) and Equation (3.29) are not space filling. Sequential designs which retain similar criteria whilst also possessing space filling properties are explored by Pronzato & Zhigljavsky (2020).

3.4 *Model fidelity*

In many scenarios, the simulator of interest can be run at varying levels of complexity. The complexity induces a trade-off between the accuracy of simulations and the corresponding computational cost. 'Cost' usually means wall clock computation time, but this could be refer to financial cost if high performance computing facilities are hired to obtain training data. There are two main methods for constructing emulators in this scenario. One is a cumulative roughness model

in which the cheapest code 'level' is thought to be relatively smooth and more expensive levels are thought to have a rougher, more complex response surface (Kennedy & O'Hagan, 2000). The more common type is an autoregressive model (Forrester *et al.*, 2007), which has strong links to co-Kriging in the geostatistics literature (Stein & Corsten, 1991). Both of these methods go by multiple names including 'multilevel emulation', 'multifidelity emulation' or some use co-Kriging regardless of whether the method is applied to the emulation of a simulator or a geostatistical analysis of some physical phenomena.

The core idea in both multilevel emulation and co-Kriging is that using a large, auxiliary data set, can help fill in the gaps of a sparse data set of direct interest. In geostatistics, we might have a small number of expensive, but highly accurate weather sensors placed over an area of interest. We may also have a much larger number of cheap, but inaccurate sensors placed over the same area. The cheap sensors provide a prediction of what the expensive sensors would report. We then construct a GP regression model based on the readings from cheap sensors but then perform a correction based on the expensive readings to provide an improved prediction of the weather at locations without a weather sensor. See Yates & Warrick (1987); Ashraf *et al.* (1997); Lark & Bishop (2007); Adhikary *et al.* (2017) for various examples.

This concept can be adapted to the emulation literature. In many cases, accuracy is increased by making a time step in a numerical differential equation solver smaller, or by constructing finer grids over space in spatial type models. Another scenario would be where there are two separate models; a cheap model with simple physics and an expensive model with much more complex physics. Two methods of combining runs from these these different, but related, simulations are proposed by Kennedy & O'Hagan (2000). Both models then become more accurate in the sense that they are better approximations of the mathematical equations being solved, they are not necessarily better approximations to reality.

3.4.1 *Autoregressive models*

A common approach to modelling simulators of different accuracy is via an autoregressive structure. Suppose we have a simulator $f_t(\mathbf{x})$ where $t \in \{1, 2, \ldots, T\}$ are the simulator levels. If $s > t$ then $f_s(\cdot)$ is assumed to be more accurate but also more expensive to run than $f_t(\mathbf{x})$. We assume that we have n_t runs of each

simulator; when $s > t$ we would expect that $n_s < n_t$. This is not a mathematical constraint but rather a practical aspect of the problem. We expect to have fewer runs of the more expensive simulator, but there is no part of this framework that *requires* it. We then induce the following model on the simulators:

$$f_1(\cdot) \mid \boldsymbol{\beta}_1, \Theta_1 \sim \mathcal{GP}\{h(\cdot)^T\boldsymbol{\beta}_1, c_1(\cdot,\cdot)\} \tag{3.31}$$

$$f_{t+1}(\cdot) = \rho_t f_t(\cdot) + \delta_{t+1}(\cdot) \tag{3.32}$$

$$\delta_t(\cdot) \mid \boldsymbol{\beta}_t, \Theta_t \sim \mathcal{GP}\left\{h(\cdot)^T\boldsymbol{\beta}_t, c_t(\cdot,\cdot)\right\} \tag{3.33}$$

where Θ_t are the hyperparameters of $c_t(\cdot,\cdot)$ and ρ_t an autoregressive parameter. The term $\delta_{t+1}(\cdot)$ essentially represents the discrepancy between $f_t(\cdot)$ and $f_{t+1}(\cdot)$. This model relies on $f_t(\cdot)$ explaining most of the variation in $f_{t+1}(\cdot)$. It is then assumed that the $\delta_t(\cdot)$ processes are much simpler than the corresponding $f_t(\cdot)$. If we take

$$c_t(\mathbf{x}, \mathbf{x}') = \sigma_t^2 \exp\left\{-\sum_{i=1}^{K} \frac{(x_i - x_i')^2}{\theta_{t,i}^2}\right\}. \tag{3.34}$$

and assume that $f_t(\cdot) \perp\!\!\!\perp \delta_{t+1}(\cdot)$, then

$$\mathrm{Cov}\left(f_t(\mathbf{x}), f_{t+1}(\mathbf{x}') \mid \boldsymbol{\beta}, \Theta\right) = \rho_t \sigma_t^2 \exp\left\{-\sum_{i=1}^{K} \frac{(x_i - x_i')^2}{\theta_{t,i}^2}\right\} \tag{3.35}$$

where $\boldsymbol{\beta} = \{\boldsymbol{\beta}_1, \boldsymbol{\beta}_2, \ldots, \boldsymbol{\beta}_T\}$ and $\Theta = \{\Theta_1, \Theta_2, \ldots, \Theta_T\}$. The design of these multilevel experiments is interesting. Kennedy & O'Hagan (2000) shows that the likelihood is greatly simplified if $D_{t+1} \subset D_t$, that is, we run f_{t+1} only at the locations which f_t has been run. Rather than fitting one joint model, this allows us to fit a GP to $f_1(\cdot)$ and then to each $\delta_t(\cdot)$. In such a case, the design matrix is of the form

$$H = \begin{pmatrix} h(X_1) & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0}) \\ \rho_{(1,1)}h(X_1) & h(X_2) & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \rho_{(1,T-1)}h(X_1) & \rho_{(2,T-1)}h(X_2) & \cdots & \rho_{(T-2,T-1)}h(X_{T-1}) & h(X_T) \end{pmatrix}. \tag{3.36}$$

where $\rho_{(t,s)} = \prod_{i=t}^{s} \rho_i$ for $s > t$. Further, conditional on all GP parameters, the covariance function for $f_t(\cdot)$ $(t \geq 2)$ is

$$C_t(\mathbf{x}, \mathbf{x}') = c_t(\mathbf{x}, \mathbf{x}') + \sum_{j=1}^{t-1} \rho_{(1,j)}^2 c_j(\mathbf{x}, \mathbf{x}'). \tag{3.37}$$

A tractable example

Consider the computation of the following integral, viewed as a function of x:

$$f(x) = \int_0^1 \cos(4\pi x^2) \sin(2\pi x t) - 4xt + 1 \, \mathrm{d}t. \tag{3.38}$$

This is an easy to compute integral, but if the integral was intractable, this could be replaced by a Riemann approximation:

$$f_R(x; T) = \frac{1}{T} \sum_{i=0}^{T-1} g(t_i; x), \tag{3.39}$$

where $g(t; x) = \cos(4\pi x^2) \sin(2\pi t x) - 4xt + 1$ is the integrand and $t_i = \frac{i}{T-1}$ are the equally spaced points at which $g(t; x)$ is evaluated. This integrand is cheap but in a complex computer model could be expensive to evaluate. Figure 3.8 shows how increasing T improves the approximation. We see when $x > 0.4$ and $T = 2$ the cheap approximation is a poor approximation to the analytical solution; the local maxima and local minima swap places. For $T \geq 4$ the approximation improves, but with diminishing returns. Suppose we have $n_e = 3$ evaluations of $f_R(x; 40)$ but also have $n_c = 20$ evaluations of $f_R(x; 4)$. The evaluations of $f_R(x; 4)$ may have been from a trial, exploratory run of the simulator or chosen to help us emulate $f_R(x; 40)$. We see, qualitatively, that the multilevel emulator offers an improved fit over the standard emulator (Figure 3.9). Firstly, the mean function more closely reflects the truth, this is especially noticeable once we start extrapolating beyond the range of the observed expensive data. There is also a moderate reduction in uncertainty about the expected function value, reflecting that the auxiliary information from a different level of code *is* informative for a more expensive version. In this toy, 1-dimensional example, it is easy to see that the multilevel emulator is better.

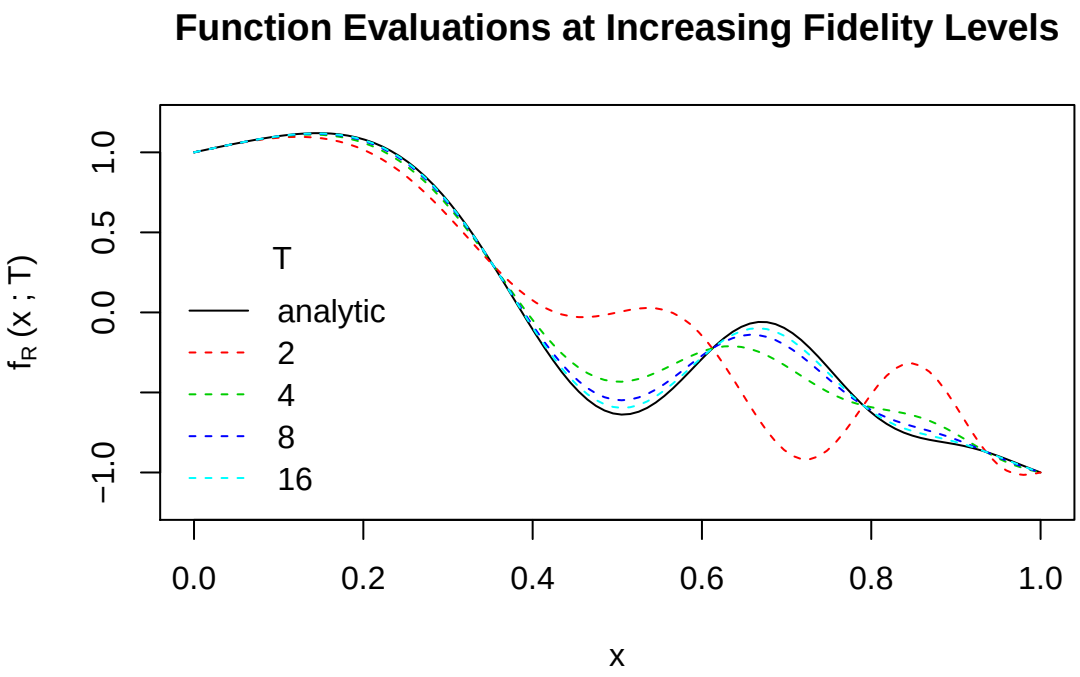

Figure 3.8: Solutions to $f_R(x;T)$ at different fidelity levels. Approximate solutions are given by dashed lines, the analytic solution is given by the solid line. T corresponds to the number of evaluations of $g(t;x)$ in the Riemann integration.

If our most accurate simulator level is still too coarse, one may be interested in producing a multilevel emulator that mimics the effect $T \to \infty$. That is, emulates the exact mathematical model rather than the code. Recent ideas from probabalistic numerics — a field of statistics and machine learning which phrases numerical solutions as inference problems — have been shown to be effective at emulating the case $T \to \infty$; we do not discuss this any further but direct the reader to Teymur *et al.* (2021) for further reading.

3.5 *Emulator comparison*

When fitting an emulator in a higher dimensional space, it is much trickier to assess fit. We will now demonstrate some emulator performance metrics on the standard GP and the multilevel emulator from Figure 3.9. Here we consider the use of two performance metrics to assess the emulator. The first is root mean squared error (RMSE). Given a set of set of 'unseen' data $\mathcal{D}_{\text{test}} = \{(\mathbf{x}_{\text{test},i}, y_{\text{test},i}), i =$

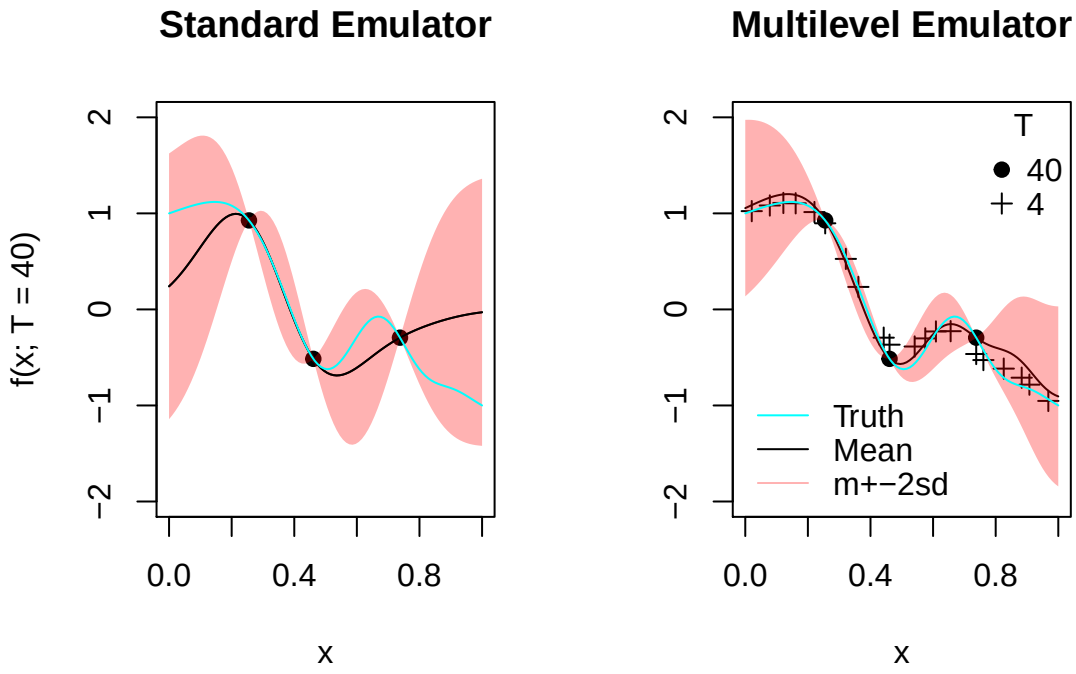

Figure 3.9: A standard GP emulator (left) and corresponding multilevel emulator (right). Here, the 'truth' refers to $f_R(x; 40)$, the object of inference, rather than the analytic solution.

$1, 2, \ldots, n_{\text{test}}\}$, this can be found via

$$RMSE = \sqrt{\frac{1}{n_{\text{test}}} \sum_{i=1}^{n_{\text{test}}} \{y_{\text{test},i} - m^*(\mathbf{x}_{\text{test},i})\}^2}. \tag{3.40}$$

RMSE only considers how close $m^*(\cdot)$ is to $f(\cdot)$. This does not take into account the uncertainty about $f(\cdot)$ or $y(\cdot)$, which motivates the use of a proper scoring rule. Proper scoring rules reward forecasts with high certainty when they are accurate but do not heavily penalise inaccurate forecasts when they have a large uncertainty attached to them (Gneiting & Raftery, 2007). For GP emulators, the following scoring rule (bigger is better) is commonly used (Binois *et al.*, 2018; Jackson & Woods, 2019)

$$S(\mathcal{D}_{\text{test}}) = -\frac{1}{n_{\text{test}}} \sum_{i=1}^{n_{\text{test}}} \left\{ \frac{[y_{\text{test},i} - m^*(\mathbf{x}_{\text{test},i})]^2}{v^*(\mathbf{x}_{\text{test},i})} + \log v^*(\mathbf{x}_{\text{test},i}) \right\}. \tag{3.41}$$

Equation (3.41) is essentially the point-wise log-likelihood based on a Normal distribution. Although the predictions are thought to be correlated, for a small and well spaced test set, the predictions should be approximately independent.

Returning to the multilevel emulators, we compute the RMSE and score for each emulator based on the two proposed metrics based on 100 equally spaced test points. This is a very large number of test points relative to the design. In practice, the size of the test set would likely be no larger than the training design. There does not appear to be any literature on appropriate sizes for test designs, but types of design have been explored (Challenor, 2013).

Table 3.1 gives the two pairs of performance metrics for the competing emulators. The RMSE for the multilevel emulator is much smaller than the standard emulator, this is expected since Figure 3.9 showed that the emulators mean for the multilevel emulator was almost the same as the target function. The score also shows a healthy improvement under the multilevel approach, this tells us that, in a combined sense, the posterior distribution provided by the multilevel emulator is a better probabilistic prediction than the posterior distribution provided by the standard emulator.

Emulator	RMSE	Score
Standard	0.4292	1.917
Multilevel	0.1139	3.588

Table 3.1: Performance metrics for two the standard and multilevel emulators in Figure 3.9.

3.6 *Emulator diagnostics*

As with any statistical model, we must investigate whether the emulator is an appropriate approximation to the simulator. The main emulator diagnostics, for deterministic simulators, were proposed by Bastos & O'Hagan (2009) (from hereon in, BO'H). A similar set of diagnostics have been proposed by Baker *et al.* (2019) for stochastic simulators. Here we review the diagnostics proposed by BO'H that are used later in this thesis. However, BO'H integrate out σ in their emulators, resulting in the posterior being a Student's t process. We will always conditions on σ, thus the posterior is Gaussian. Therefore, the diagnostics we present have been adapted to assume a GP posterior.

3.6.1 *The diagnostics*

The diagnostics all rely on Cholesky Errors (C.E.s). These are analogous to standardised residuals in a standard regression analysis but take into account the covariance structure of the emulator. This is important since, we expect $|f(\mathbf{x}) - f(\mathbf{x}')|$ to be small whenever $||\mathbf{x} - \mathbf{x}'||$ is small. In other words, we expect the residuals to have structure even when the emulator is an appropriate representation of the simulator. The C.E.s are defined to be

$$\boldsymbol{e}_{\text{chol}} = \Sigma_{\text{test}}^{-1/2}(\mathbf{y}_{\text{test}} - m^*(X_{\text{test}})) \tag{3.42}$$

where $m^*(X_{\text{test}})$ is the emulator's posterior mean vector and Σ_{test} is the posterior variance matrix for the test points. $\Sigma_{\text{test}}^{-1/2}$ is the inverse of the lower Cholesky decomposition of Σ_{test}. If the emulator is an adequate approximation to the simulator then $\boldsymbol{e}_{\text{chol}}$ should appear as a random sample from $\mathcal{N}_{n_{\text{test}}}(0, I)$ when plotted against $x_{\text{test},i}$. We can perform further diagnostics based on these residuals. Since they are (supposedly) a random sample from $\mathcal{N}(0,1)$, a QQ-plot can be constructed to assess Normality, as can a coverage plot which compares

$$C_\alpha(\boldsymbol{e}) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}(e_i \in I_\alpha) \tag{3.43}$$

to various values of α. Here, I_α is a (symmetric) interval in which $100(1-\alpha)\%$ of the C.E.s are expected to lie. In the Normal case, $I_\alpha = \left(\Phi(\frac{\alpha}{2}), \Phi(1-\frac{\alpha}{2})\right)$.

3.6.2 *Example*

We will work with the 2-dimensional toy function that BO'H use:

$$f(\mathbf{x}) = (1 - e^{-0.5x_2^{-1}})\left(\frac{2300x_1^3 + 1900x_1^2 + 2092x_1 + 60}{100x_1^3 + 500x_1^2 + 4x_1 + 20}\right), \mathbf{x} \in (0,1)^2. \tag{3.44}$$

We then constructed a 20 point LH on $(0,1)^2$, fit the emulator using the same parameters as BO'H; $\sigma^2 = 3.3316$, $\boldsymbol{\theta} = (0.2421, 0.4240)$ and we used a nugget effect of $\lambda^2 = 10^{-16}$ (BO'H did not explicitly use a nugget term, but we have included one for computational reasons). A second, independent LH of size 20 was constructed to generate a test set of data. We also assumed a linear mean function but with

with $\boldsymbol{\beta} \sim N(\mathbf{0}, 100I_3)$ rather than the improper prior $\pi(\boldsymbol{\beta}) \propto 1$. The C.E.s then lead to a suite of diagnostic plots (Figure 3.10). It is clear from the plots that there is serious emulator-simulator discrepancy. When x_1 and x_2 are both less than about 0.3 there are some very large C.E.s (around 6 in magnitude). This is reinforced by the the QQ-plot; the points do not lie close to the line. For a small sample size the coverage plots can be highly non-smooth, but we can see there is a 'flat line' of observed coverages around 0.8 indicating a gap in the coverages. This also suggests serious emulator-simulator discrepancy. To improve the emulator we will follow BO'H and generate some additional data points and update the GP hyperparameters. If the simulator were very expensive this might not be possible. If further simulator runs are not possible, we can try other methods to improve the emulator. Common approaches include changing the mean function (Overstall & Woods, 2016) and/or covariance function to to better reflect the simulator behaviour (Volodina & Williamson, 2020; Vernon *et al.*, 2022*b*). We generate an additional 10 points from an independent LH design on $(0, 0.5)^2$ in an attempt to build a satisfactory emulator. We update the hyperparameters to the values BO'H used; $\sigma^2 = 4.9389$, $\boldsymbol{\theta} = (0.1764, 0.4116)$. The emulator is then refit leading to the diagnostics in Figure 3.11. The diagnostics are much improved; there are no very large C.E.s (we have one $|e_i| > 1.96$, but this is expected in a sample of size 20). There is no striking systematic pattern in the residuals, perhaps there is some indication of decreasing variance as the x_i increase but this is hard to judge with a small sample size. The QQ-plot shows good fit to the line and the coverage plot is within the realms of what we would expect from a sample of size 20.

3.7 *Other surrogate models*

Emulators do not have to be GPs. Other approaches have different advantages to GPs. For example, if the simulator inputs are permutations, the size of $||\mathbf{x} - \mathbf{x}'||$ implies little about $|f(\mathbf{x}) - f(\mathbf{x}')|$. In permutation space, a Benter model inspired emulator offers a promising solution (Wilson *et al.*, 2018*c*). Most non-GP surrogates do leverage some kind of smoothness about $f(\cdot)$ and thus re-purpose conventional regression techniques. Examples include generalised additive models, neural networks and splines (Marrel *et al.*, 2012; Tripathy & Bilionis, 2018; Barton & Meckesheimer, 2006). We now discuss two non-GP emulators in turn.

3.7.1 *Linear models*

Linear regression (Bayesian or frequentist) offers a much faster inference and prediction framework than GPs with, for example, squared exponential or Matérn covariance structures, (Rougier *et al.*, 2009). This is because GP inference usually relies on inverting a matrix, an $O(n^3)$ operation. Prediction, given a GP precision matrix is then $O(n^2)$. For fully Bayesian treatments with S Markov chain Monte Carlo (MCMC) samples, predictions are typically $O(Sn^3)$, where S is fairly large ($S \geq 10^4$). Inference is more costly still, in part to due mixing and thinning of MCMC schemes. For example, Baggaley *et al.* (2012) needed $550K$ MCMC iterations *after* a series of trial MCMC schemes used to tune their sampling scheme. Under certain prior assumptions, the posterior distribution of a linear regression is fully tractable, thus complex computations such as matrix inversion need to be performed only once. One interpretation of the linear regression emulator is a type of emulator with a covariance that is "all nugget". The advantage of GPs over linear regression is that GPs automatically detect otherwise difficult to model features in the data, so the human cost of linear modelling — fitting many different functional forms, including interaction and polynomial terms — may be greater than the cost of inverting a matrix. The choice between linear regression and GP is context dependent; Salter & Williamson (2016) explore this in detail and find merits in both approaches. Although a simple approach, this can work quite well in both the stochastic and deterministic settings. For example, even when the simulator is deterministic, a nugget term is often incorporated. This might be a very small nugget for computational reasons or a larger nugget as a way to account for deviations from assumptions (Gramacy & Lee, 2012). Another reason is that the fitted emulator might only depend on a subset of the variables fed into the simulator. This means that the emulator has a random component to account for the information 'lost' by omitting unimportant variables from the emulator, thus a nugget is required almost always regardless of whether we have a stochastic or deterministic emulator.

3.7.2 *Bayes linear emulators*

Bayes linear emulators (Jones *et al.*, 2016; Wilson *et al.*, 2018*a*; Bower *et al.*, 2010; Vernon *et al.*, 2019; Jackson & Vernon, 2023) come from a framework which is, in principle, Bayesian. The prior specification is greatly simplified as the prior

specification is only of means and (co)variances. In the Bayes linear framework we want to update our beliefs about a vector of beliefs B having observed a data vector D. Following Goldstein (2007) the "posterior" (known as *adjusted*) mean and (co)variances are given by

$$\mathrm{E}_D(B) = \mathrm{E}(B) + \mathrm{Cov}(B, D)\mathrm{Var}(D)^{-1}(D - \mathrm{E}(D)) \tag{3.45}$$

$$\mathrm{Var}_D(B) = \mathrm{Var}(B) - \mathrm{Cov}(B, D)\mathrm{Var}(D)^{-1}\mathrm{Cov}(D, B). \tag{3.46}$$

These formulae bear a striking resemblance to the conditional Normal equations. Despite the resemblance, they have an important but subtle distinction from the conditional Normal equations. The conditional Normal equations are the posterior moments *of a Normal distribution* and thus imply precise probability statements about $f(\mathbf{x})$. The Bayes linear equations are distribution free statements (in terms of both the likelihood and prior) thus are not making precise probability statements. Proponents of a Bayes linear analysis claim this simplified specification allows for a more robust analysis because we are not tied to probability statements. The downside is that it might be harder to interpret a Bayes linear emulator. With a GP emulator, if $f(\mathbf{x}) \sim \mathcal{N}(0, 1)$ then we can make statements such as $\mathrm{P}(f(\mathbf{x}) > 0) = 0.5$, whereas a Bayes linear emulator will not offer this level of detail. It is possible to make imprecise probability statements by feeding the adjusted moments into certain results from probability theory such as Pukelsheim's 3σ rule (Pukelsheim, 1994) or Markov's inequality. Neither approach is necessarily better, the advantages and disadvantages of each paradigm should be weighed up and it is indeed valid to mix the two approaches to leverage the benefits of each. One example of a mixed approach is Owen *et al.* (2020); a (truncated) GP emulator and a Bayes linear emulator are used together to emulate a simulator exhibiting different behaviour patterns.

Ultimately, a Bayes linear analysis produces a central estimate and quantification of uncertainty in an efficient way, which is all that is often needed from an emulator. A Bayes linear analysis can be thought of as either an approximate Bayesian analysis or the appropriate analysis when we are only willing to specify means and (co)variances. Since Equation (3.45) corresponds to Equation (3.19) and Equation (3.46) corresponds to Equation (3.20), the Bayes linear approach works exceptionally well when B is approximately Normal, but offers a valid analysis when B is non-Normal. In the Bayes linear framework, we would usually

take $D = y(X)$ and $B = y(X^*)$ (or $f(X^*)$) and $\text{Cov}(B, D)$ and $\text{Var}(D)$ can be found via assuming an appropriate covariance function, such as a squared exponential. This allows for a Bayesian emulator whilst avoiding MCMC schemes or other approximations. Although not strictly confined to the Bayes linear framework, proponents of the Bayes linear framework typically use emulators of the form

$$f(\mathbf{x}) = \sum_{j=1}^{q} \beta_j h_j(\mathbf{x}_{[A]}) + w(\mathbf{x}_{[A]}) + \varepsilon(\mathbf{x}) \tag{3.47}$$

where $\mathbf{x}_{[A]}$ are the "active" inputs (the subset of inputs thought to be most influential on simulator output), $\sum_{j=1}^{q} \beta_j h_j(\mathbf{x}_{[A]})$ is a mean function, $w(\cdot)$ is a mean-zero weakly stationary stochastic process and $\varepsilon(\cdot)$ is a white noise process. In this formulation, $w(\cdot)$ is analogous to a (mean zero) GP in the fully Bayesian setting, thus we write $\text{Cov}\{w(\mathbf{x}_{[A]}), w(\mathbf{x}'_{[A]})\} = C(\mathbf{x}_{[A]}, \mathbf{x}'_{[A]})$. The mean function is often very detailed which means that — if $f(\mathbf{x})$ is well approximated by $h(\mathbf{x})^T\boldsymbol{\beta}$ — there is a reduced reliance on choosing the correct form for $C(\cdot, \cdot)$. A more detailed mean function may allow the emulator to be more interpretable, as the regression function is more understandable than the non-parametric component of the emulator (Bower *et al.*, 2010; Vernon *et al.*, 2019).

3.8 *Summary*

The purpose of this chapter was to motivate and establish the core concepts used later in the thesis. We began with the central concept of this thesis; an emulator. That is, a GP approximation to a simulator. We introduced the notion of a GP as a prior distribution over functions. We described how to encode certain prior beliefs about $f(\cdot)$ through this prior; we considered some common covariance structures for emulators and how these each imply different prior specifications for an unknown function. We also examined the effects of combining covariance structures. Multiplication of covariance structures typically leads to multiple input emulators; addition of covariance structures allows us to build emulators with different components. We also discussed mean functions and evaluated the pros and cons of mean functions of different complexities.

We then moved to posterior inferences and design strategies for $f(\cdot)$. The multivariate Normal equations were presented as our path to an analytic posterior,

conditional on GP hyperparameters; we discussed some approaches to estimation of both the hyperparameters and regression coefficients. We had a fleeting discussion of stochasticity; this will be greatly expanded upon in the next chapter. We contrasted one-shot and sequential designs and with a bias towards constructing space filling designs. Chapter 6 will discuss some sequential design approaches in greater detail with a particular emphasis on optimisation as a sequential problem.

The final GP emulator we considered in this chapter was an autoregressive emulator which used one cheap code level to improve both the emulation of a more expensive level of code, with a comparison to co-Kriging in geostatistics. We presented a concrete example of a two-level emulator but presented mathematical details of how to extend to an arbitrary number of levels. We then showed, via appropriate metrics, that the multilevel emulator out performed a standard emulator. We closed the consideration of GP emulators by presenting a suite of appropriate diagnostics for GP emulators.

We concluded this chapter on emulators by considering some approaches to emulation *not* based on GPs. The first was linear regression based emulation, and the other was Bayes linear emulation. Although both are different to GP emulation, they can both have strong links. Linear regression can be thought of as a GP emulator based on linear and white noise covariance functions. We saw that the multivariate Normal equations are mathematically identical to the Bayes linear update equations.

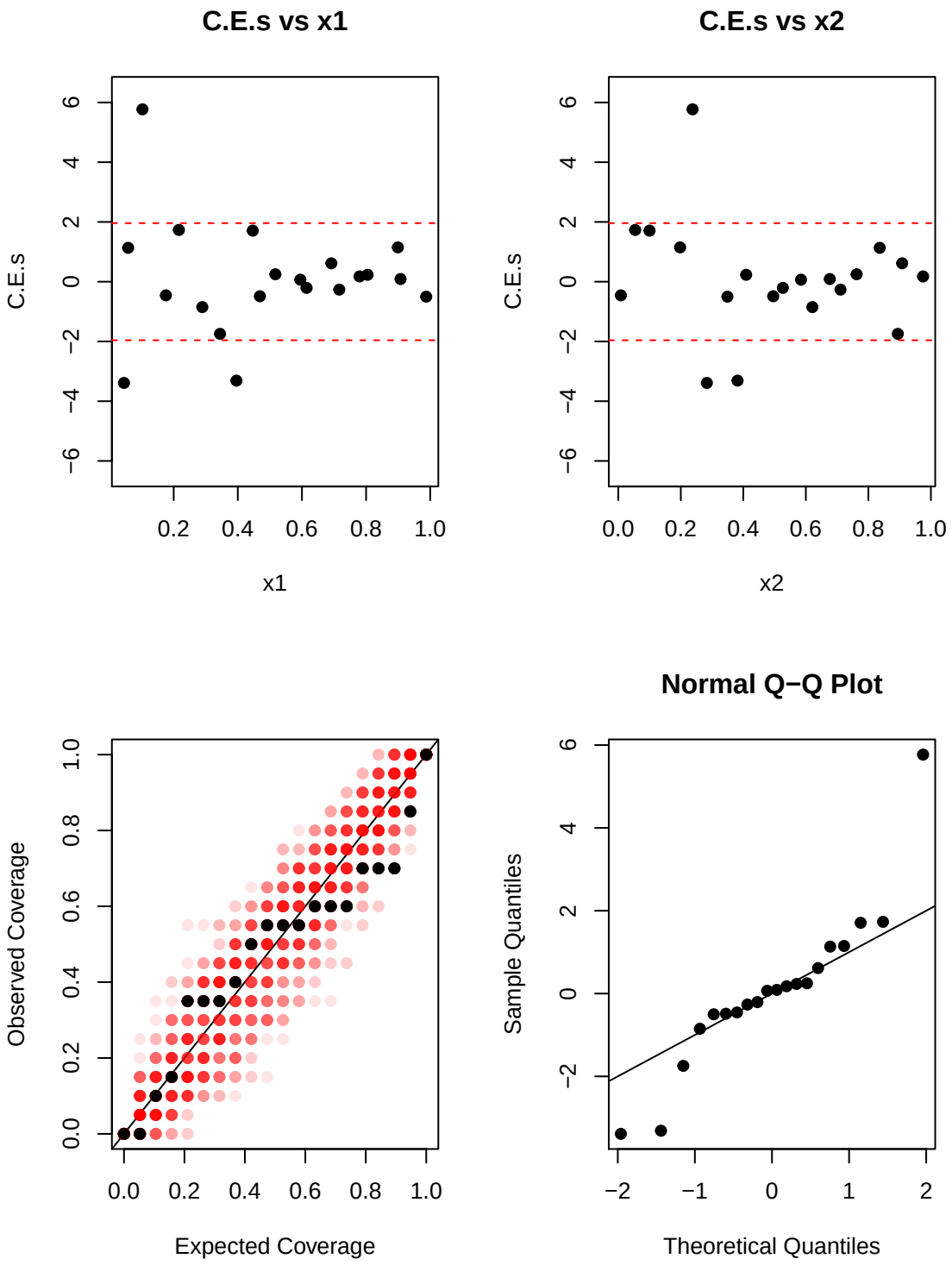

Figure 3.10: Diagnostic Plots for the toy emulator. Top two plots are C.E.s against the inputs (left is input 1, right is input 2). Bottom right shows a QQ plot with the unit diagonal superimposed. Bottom left is the coverage plot, black points correspond to observed C.E.s, translucent red dots correspond to the coverage observed from 100 iid $\mathcal{N}(0, 1)$ samples of size 20.

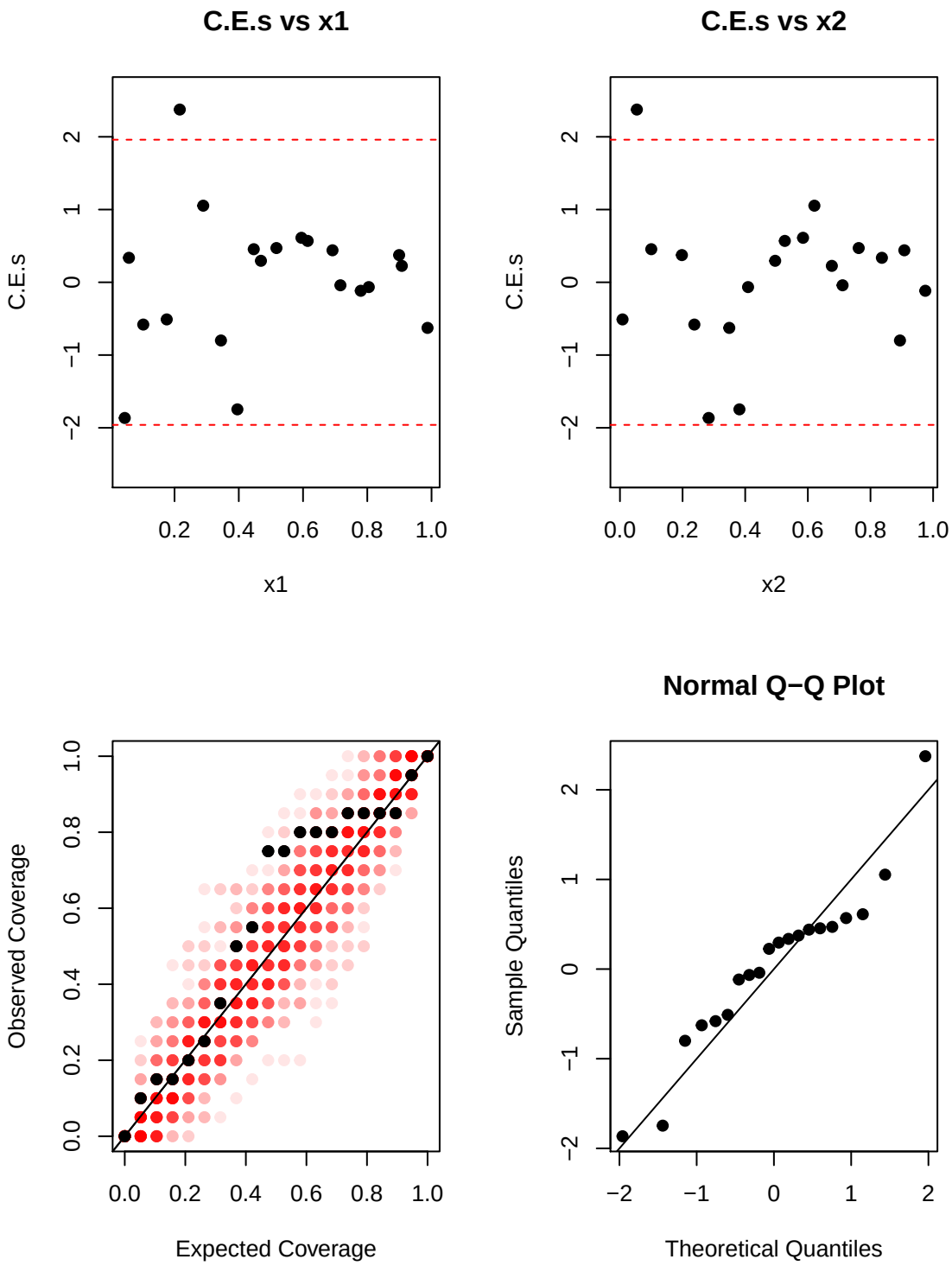

Figure 3.11: Diagnostic Plots for the toy emulator. Top two plots are C.E.s against the inputs (left is input 1, right is input 2). Bottom right shows a QQ plot with the unit diagonal superimposed. Bottom left is the coverage plot, black points correspond to observed C.E.s, translucent red dots correspond to the coverage observed from 100 iid $\mathcal{N}(0, 1)$ samples of size 20.

Chapter 4

Heteroscedastic & Multilevel GP Emulators For Stochastic Simulators

The work in this chapter has been written up as a paper and recently accepted for publication in *Journal of the Royal Statistical Society: Series C (Applied Statistics)* (Kennedy *et al.*, 2023). The main contribution of this paper is the introduction of the stochastic multilevel (SML) emulator. A secondary contribution of this paper is comparing algorithms to fit SML emulators, as well as comparing algorithms to fit HetGP emulators, which SML emulators are a natural extension of.

4.1 *Introduction*

The previous chapter focused on using GP priors to model complex computer simulators. We reviewed many of the classical emulation strategies which were developed between the 1980's and early 2000's. This literature focused on *deterministic* simulators, since, at the time, they were the most common type of simulation model (Sacks *et al.*, 1989; Craig *et al.*, 1997; Kennedy & O'Hagan, 2001; Santner *et al.*, 2003). These deterministic models were implemented even if the corresponding real world process exhibited stochasticity (Kennedy & O'Hagan, 2001). Of course, this absent stochasticity is 'wrong', as are all models. Nonetheless, these deterministic models still have practical value.

To incorporate uncertainty into predictions, modellers have produced increasing numbers of stochastic simulators. As stochastic simulation has become more prominent across all sciences, so has the interest in the emulation of stochastic

computer simulators (Henderson *et al.*, 2009; Astfalck *et al.*, 2019; Rocchetta *et al.*, 2018; Boys *et al.*, 2018). There are a variety of approaches to (GP based) emulation of stochastic computer models; see Baker *et al.* (2022) for a recent overview.

The simplest approach to modelling stochastic simulators, which was outlined in the previous chapter, is to add $\lambda^2 I$ to the covariance structure. This homoscedastic approach is rather limiting when Var $\{y(\mathbf{x})\}$ is thought to depend on $\mathbf{x}$. A canonical example of heteroscedasticity is the 'motorcycle data', which comes directly from a stochastic computer experiment (Schmidt *et al.*, 1981). This computer experiment outputs accelerometer readings over time following a crash. Although Silverman (1985) fits splines to this data, we will use GPs and start with a homoscedastic GP (HomGP).

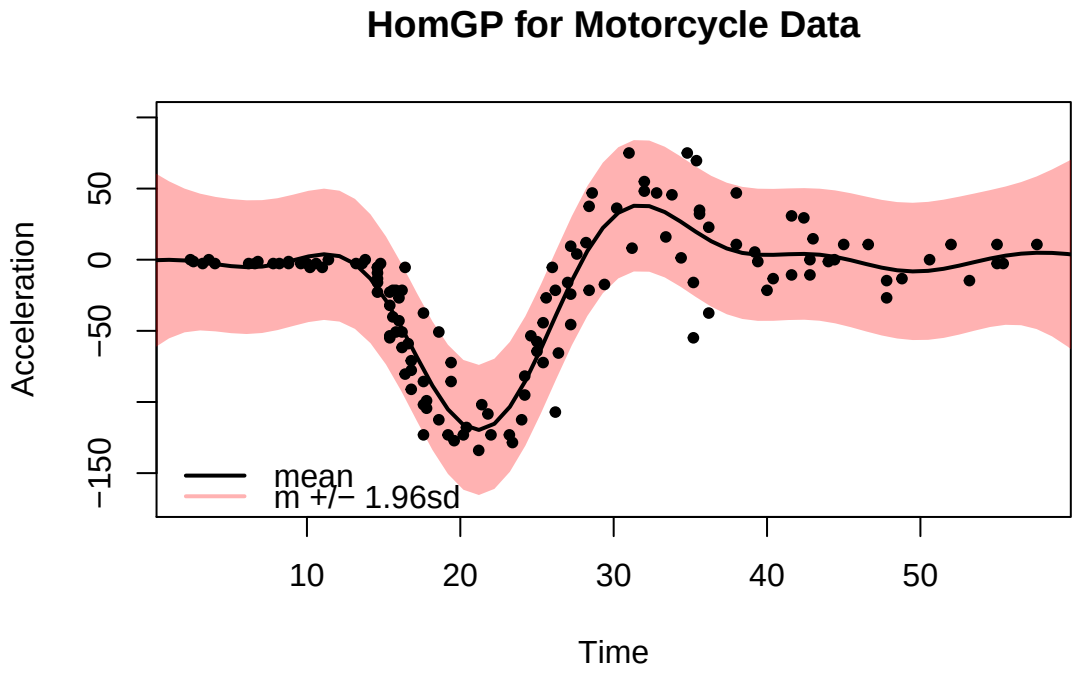

Figure 4.1: The motorcycle data set with HomGP superimposed. The black line is the emulator mean and the red band corresponds to a 95% posterior predictive interval for $y(x)$.

The fitted mean function in Figure 4.1 for the motorcycle data is reasonable. The pitfall is that the emulator's predictive variance is a poor match for the variability in the data, thus interval estimates for $y(\mathbf{x})$ are poorly calibrated. There will also be a knock-on effect on uncertainty quantification for E $\{y(\mathbf{x})\}$. The

misspecified variance is most prominent for Time < 15, where it is clear that $\text{Var}\{y(\mathbf{x})\}$ is very small but the emulator estimates $\text{Var}\{y(\mathbf{x}) \mid \mathcal{D}) \approx 28$. The predictive variance also looks too large for Time > 40. The predictive variance is reasonably accurate for Time $\in [15, 40]$. Ultimately, to produce surrogates for heteroscedastic simulators, we need to produce emulators with a heteroscedastic element baked into them. Our goal here is to find an approach which can:

(i) produce a mean estimate of $y(\mathbf{x})$;

(ii) quantify uncertainty about $\text{E}\{y(\mathbf{x})\}$;

(iii) provide flexible estimates of $\lambda^2(\mathbf{x}) = \text{Var}\{y(\mathbf{x})\}$.

4.2 *Separate mean and variance emulation*

To estimate the mean response of a heteroscedastic stochastic simulator, we might apply Stochastic Kriging (SK) to runs of the simulator (Ankenman *et al.*, 2010). SK is similar to the constant noise set up. The difference being that instead of having a (prior) covariance matrix of the form $\text{Var}\{y(X)\} = C(X, X) + \lambda^2 I$, we have

$$\text{Var}\{y(X)\} = C(X, X) + \text{diag}\left(\frac{s^2(\mathbf{x}_1)}{r_1}, \frac{s^2(\mathbf{x}_2)}{r_2}, \ldots, \frac{s^2(\mathbf{x}_k)}{r_k}\right) \tag{4.1}$$

as the covariance matrix, where $s^2(\mathbf{x}_i)$ is the (sample) variance based on r_i replicated runs of $y(\mathbf{x}_i)$. For SK it is recommend that $r_i \geq 10$. This deals with the heteroscedasticity at the design points since the nugget term depends on $\mathbf{x}$. However, the noise is treated as a nuisance rather than as an object of inference; SK only emulates $\text{E}\{y(\mathbf{x})\}$. Emulating stochastic computer models can be viewed as modelling a multiple output simulator. If both outputs (mean and variance) are important, then we need to explicitly model $\lambda^2(\mathbf{x})$.

Henderson *et al.* (2009) construct two independent GP emulators: one for $\text{E}\{y(\mathbf{x})\}$ and another for $\log \lambda(\mathbf{x})$. Note that Henderson *et al.* (2009) emulate the log standard deviation, rather than log variance, but these are identical up to a multiplicative factor of 2. Their case study employs around 1000 replicates per design point; this does not take too long for their example (they generate their training data in about a day using parallel computing) but for more expensive simulators this may be an unachievable sample size. Simple sample mean and

standard deviation estimates are used as the training data to construct their emulators. This independence assumption is pragmatic; the simulator mean and standard deviations are both functions of $\mathbf{x}$, but it is a practical and useful modelling choice. A very similar approach is adopted by Andrianakis *et al.* (2017*b*) and is shown to be more powerful, in terms of increasing the efficiency of an calibration procedure, than taking $\lambda^2(\mathbf{x}) = \lambda^2$.

Marrel *et al.* (2012) take a similar approach to Henderson *et al.* (2009) but couple together their mean and variance GPs. They first estimate the mean response using a HomGP. Let the posterior mean be $m_0(\mathbf{x})$. Next they estimate $\text{Var}\{y(\mathbf{x})\} \approx \frac{1}{r}\sum_{i=1}^{r}(y_i(\mathbf{x}) - m_0(\mathbf{x}))^2$ and fit a GP to the estimated variances. Let the posterior mean of this GP be $m_V(\cdot)$. They then fit a new GP to $y(\mathbf{x})$ using $m_V(\cdot)$ in place of λ^2: $y(\cdot) \sim \mathcal{GP}\{m(\cdot), C(\cdot,\cdot) + m_V(\cdot)I\}$. The authors suggests that this approach may be improved by alternating between the fitted variance GP and the mean GP until some convergence criterion is met.

4.2.1 *Non-Gaussian noise*

If the stochasticity exhibited by $y(\mathbf{x})$ is not well approximated by a Normal distribution (e.g. asymmetric), then more work may have to be done. Transformation is a simple but effective method for addressing non-Normality. For example, if it is known that $y(\mathbf{x}) > 0$ a log transform may be enough to rectify non-Normality. The availability, $A(\mathbf{x})$, in Athena is constrained to the unit interval thus, constructing an emulator for logit $A(\mathbf{x})$ or probit $A(\mathbf{x})$ would be a sensible strategy.

If the simulator behaviour is too complex to be made (approximately) Gaussian by transformation, Quantile Kriging may be an appropriate tool (Plumlee & Tuo, 2014). Since quantiles are directly estimated, rather than moments, Quantile Kriging is very flexible and can reproduce complex simulator behaviour such as multimodal outputs and discrete outputs without additional modelling. This is perhaps the most flexible approach to modelling stochastic simulators. This flexibility comes at a large cost; hundreds of replicates are required to model non-Gaussian, input dependent noise in computer simulators. Many stochastic simulators, such as Athena, can be computationally expensive. Thus such levels of replication could make emulation of the Athena, and other simulators, infeasible unless huge training budgets are available.

Another approach to using GPs to model simulators which are asymmetric or

otherwise non-Normal is to embed a GP into another statistical model. This is an attractive approach since it may help leverage problem structure. For example, if a model outputs an integer (perhaps, the number of cells that have mutated in a biological system), we may wish to model the response as $y(\mathbf{x}) \sim \text{Poisson}(\gamma(\mathbf{x}))$ with $\log \gamma(\cdot) \sim \mathcal{GP}\{m(\cdot), C(\cdot,\cdot)\}$. Note the lack of nugget term in the GP because the stochasticity is absorbed into the Poisson error structure (this is also true for many non-Gaussian models which utilise a GP). A Bayesian treatment would require intensive numerical methods to integrate out the latent GP to form the posterior density for $y(\mathbf{x})$ in this Poisson model. For this reason, the person constructing the emulator may prefer to employ the tractable and computationally efficient GP over a more accurate, but unwieldy, non-Gaussian emulator. For example, the capture-recapture example illustrated in Baker *et al.* (2022) is non-Gaussian, but the (computational) advantages of employing a GP surrogate outweigh this misrepresentation of the simulator.

If the response is non-Gaussian, but all that is required is a central estimate and quantification of uncertainty, then a Bayes linear approach may be useful. If probabilistic predictions are required, the adjusted moments can be plugged into a parametric form. For example, Jackson & Woods (2019) use a Bayes linear emulator to find the adjusted mean and variance of a simulator, then plug these moments into (i) a Normal distribution and (ii) a uniform distribution to construct probabilistic predictions for a deterministic simulator. This approach of plugging moments into a parametric form can easily be adapted to stochastic simulations. However, those utilising a Bayes linear approach typically prefer not to assume any probabilistic form and report only the adjusted moments.

4.3 *Heteroscedastic Gaussian processes*

The approaches considered so far in this chapter typically rely on using multiple emulators to generate predictions for a single quantity of interest; $y(\mathbf{x})$. There is something inherently dissatisfying about requiring two independent statistical models to make predictive statements about a single quantity of interest. A unified approach is appealing.

Unification of mean and variance emulators motivates the Heteroscedastic Gaussian Process (HetGP), which originated in the machine learning literature as a regression procedure (Goldberg *et al.*, 1998) but has recently gained traction

in the emulation literature for modelling simulators exhibiting input dependent noise (Binois *et al.*, 2018; Baker *et al.*, 2020*a*).

4.3.1 *The HetGP emulator*

If $y(\cdot)$ is a stochastic computer model, then the HetGP emulator is as follows:

$$y(\cdot)|\lambda^2(\cdot) \sim \mathcal{GP}\{\mu(\cdot), C(\cdot,\cdot) + \lambda^2(\cdot)\} \tag{4.2}$$

$$\log \lambda^2(\cdot) \sim \mathcal{GP}\{\mu_V(\cdot), C_V(\cdot,\cdot) + \lambda_V^2 I\}. \tag{4.3}$$

An alternative representation is $y(\cdot) = f(\cdot) + \sqrt{\lambda^2(\cdot)}\varepsilon$ where $f(\cdot) \sim \mathcal{GP}\{\mu(\cdot), C(\cdot,\cdot)\}$, $\log \lambda^2(\cdot) \sim \mathcal{GP}\{\mu_V(\cdot), C_V(\cdot,\cdot) + \lambda_V^2 I\}$ and $\varepsilon \sim \mathcal{N}(0,1)$.

Here, $\mu(\cdot)$ and $C(\cdot,\cdot)$ are mean and covariance functions (respectively) for $f(\cdot) = \mathrm{E}\{y(\cdot)\}$. $\mu_V(\cdot)$ and $C_V(\cdot,\cdot)$ are the mean and covariance functions for $\mathrm{E}\{\log \lambda^2(\cdot)\}$ and $\lambda_V^2 I$ denotes a constant nugget effect for the log variance. Now $\mathrm{Var}(y(\mathbf{x})) = C(\mathbf{x},\mathbf{x}) + \lambda^2(\mathbf{x})I$ depends explicitly on $\mathbf{x}$ even if $C(\cdot,\cdot)$ is a stationary covariance function, thus HetGP is a type of non-stationary GP. Boukouvalas (2010) proposed simplifying HetGP by replacing the GP prior on $\log \lambda^2(\cdot)$ by a known parametric form with parameters to be inferred.

A unique property of HetGP is that it need not require replication, but still allows for it. The allure of HetGP is the promise of a full surrogate; joint prediction of the mean response and the level of noise at any input combination. This is possible via a latent variable formulation which jointly models the simulator mean as a GP and the log noise (to ensure positivity) as a GP. As Gramacy (2020) notes, this coupled GP approach provides smooth estimates of the noise at both within sample and out of sample simulator inputs.

As with the previous chapter, we will assume squared exponential covariance functions for $C(\cdot,\cdot)$ and $C_V(\cdot,\cdot)$, but this is not a requirement. The `hetGP` package allows for other covariance functions such as Matérn (Binois & Gramacy, 2019). Notation for the simulator mean, $f(\cdot)$, will be the same as in the previous chapter. The equivalent notation for $\log \lambda^2(\cdot)$ will have a V subscript. To our knowledge, no previous work on using HetGPs considers parameter uncertainty in a tractable format; Goldberg *et al.* (1998) propose a suitable MCMC scheme for full Bayesian computation. In the next section we outline our partially Bayesian HetGP, which does not require intensive computation for inference or prediction.

4.3.2 *A Bayesian HetGP*

Our approach to a tractable, but Bayesian, HetGP is to integrate out the $\boldsymbol{\beta}$ parameters of all regression components in the two GPs. First consider the latent log-variance GP:

$$\log \lambda^2(\cdot) \mid \boldsymbol{\beta}_V, \Theta_V \sim \mathcal{GP}\{h_V(\cdot)^T \boldsymbol{\beta}_V, C_V(\cdot,\cdot) + \lambda_V^2 I\} \tag{4.4}$$

where Θ_V are the hyperparameters of $C_V(\cdot,\cdot)$ and the nugget variance; that is $\Theta_V = \{\sigma_V, \boldsymbol{\theta}_V, \lambda_V\}$. Since this is a homoscedastic GP, if we take the prior $\boldsymbol{\beta}_V \sim \mathcal{N}\{\boldsymbol{b}_V, B_V\}$ then by standard properties of multivariate Normal random variables we arrive at

$$\log \lambda^2(X) \mid \Theta_V \sim \mathcal{GP}\{H_V \boldsymbol{b}_V, C_V(X,X) + H_V B_V H_V^T + \lambda_V^2 I\} \tag{4.5}$$

where the i-th row of H_V corresponds to $h_V(\mathbf{x}_i)$. As with the GPs presented in Chapter 3 we will use the notation $K(\cdot,\cdot)$ to represent a covariance function after integrating out regression coefficients. A V subscript is used when referring to aspects of $\log \lambda^2(\cdot)$. The posterior moments of $\log \lambda^2(\cdot) \mid \Theta_V$ are a direct consequence of the multivariate Normal equations. Prediction of $\log \lambda^2(X^*)$ at new points X^* with design matrix H_V^* is given by:

$$\log \lambda^2(X^*) \mid \Theta_V, \log \lambda^2(X) \sim \mathcal{N}\{m_V^*(X^*), v_V^*(X)\} \tag{4.6}$$

$$m_V^*(X^*) = H_V^* \boldsymbol{b}_V - K_V(X^*, X) K_V(X, X)^{-1} (\log \lambda^2(X) - H_V \boldsymbol{b}_V) \tag{4.7}$$

$$v_V^*(X^*) = K(X^*, X^*) - K_V(X^*, X) K_V(X, X)^{-1} K_V(X, X^*) + \lambda_V^2 I \tag{4.8}$$

which relies on estimates of $\log \lambda^2(X)$, the values of the (log) latent variance process. In practice, these cannot be observed unless replication is present. Within a Bayesian framework, estimating $\log \lambda^2(X)$ is automatic since it is simply an unknown. As with any other unknown, we can express prior beliefs about $\log \lambda^2(X)$ and then estimate $\log \lambda^2(X)$ using standard estimation techniques. A MAP estimate can be found if a point estimate is sufficient. More intensive methods, such as MCMC, will allow us to obtain a distribution for $\log \lambda^2(X)$ if necessary. In Section 4.5.3 and Section 4.6, we provide some practical information about estimating

$\log \lambda^2(X)$.

Then our model for the observable $y(\mathbf{x})$ is

$$y(\cdot) \mid \boldsymbol{\beta}, \Theta, \lambda^2(\cdot) \sim \mathcal{GP}\{h(\cdot)^T\boldsymbol{\beta}, C(\cdot,\cdot) + \lambda^2(\cdot)I\}, \tag{4.9}$$

where Θ denotes Θ_V and all the hyperparameters of the GP in Equation (4.9), namely, $\Theta = \{\Theta_V, \sigma, \boldsymbol{\theta}\}$. Now we can integrate out $\boldsymbol{\beta} \sim \mathcal{N}(\boldsymbol{b}, B)$ in the usual way leading to

$$y(X) \mid \Theta, \lambda^2(\cdot) \sim \mathcal{N}\{H\boldsymbol{b}, K(X,X) + \lambda^2(X)I\} \tag{4.10}$$

where $K(X,X) = HBH^T + C(X,X)$ is the covariance function of $\mathrm{E}\{y(X)\}$ after integrating out $\boldsymbol{\beta}$. Under this approach, we need to obtain a point estimate of $\lambda^2(\mathbf{x})$ from Equation (4.6). Obvious candidates are the posterior mean, median and mode of $\lambda^2(X^*)$. Since $\log \lambda^2(X)$ is Normal, $\lambda^2(X)$ is log Normal and thus:

$$\mathrm{E}\{\lambda^2(X^*) \mid \mathcal{D}, \Theta\} = \exp\{m_V^*(X^*) + \frac{1}{2}v_V^*(X^*)\} \tag{4.11}$$

$$\text{mode}\{\lambda^2(X^*) \mid \mathcal{D}, \Theta\} = \exp\{m_V^*(X^*) - v_V^*(X^*)\} \tag{4.12}$$

$$\text{median}\{\lambda^2(X^*) \mid \mathcal{D}, \Theta\} = \exp\{m_V^*(X^*)\}. \tag{4.13}$$

The most conservative estimate is the mean, whereas the mode will be the most optimistic. All three will be approximately equal when $|v_V^*(X^*)| << |m_V^*(\mathbf{x})|$. Thus, if the variance is much less than the magnitude of the mean, we would recommend using the median point estimate as this does not rely on $v_V^*(X^*)$ thus is fastest to compute. The speed difference will be greatest for larger sample sizes, and typically $v_V^*(X^*)$ will be small for large sample sizes. Also small values for σ_V and λ_V lead to small $v_V^*(X^*)$.

4.3.3 *Returning to the motorcycle example*

Returning to the motorcycle example, Figure 4.2 presents a HetGP fitted to the motorcycle data. The predictive intervals provided by the HetGP emulator are more appropriate than those from the HomGP emulators. The interval width is small for Time < 15, is wider when 15 < Time < 30 and then shrinks again when Time > 30, mimicking the input-dependent variability in the data. HetGPs are great at detecting non-linear variability in both the mean response surface of a simulator and the variance surface. However, it is incredibly data-hungry; Binois

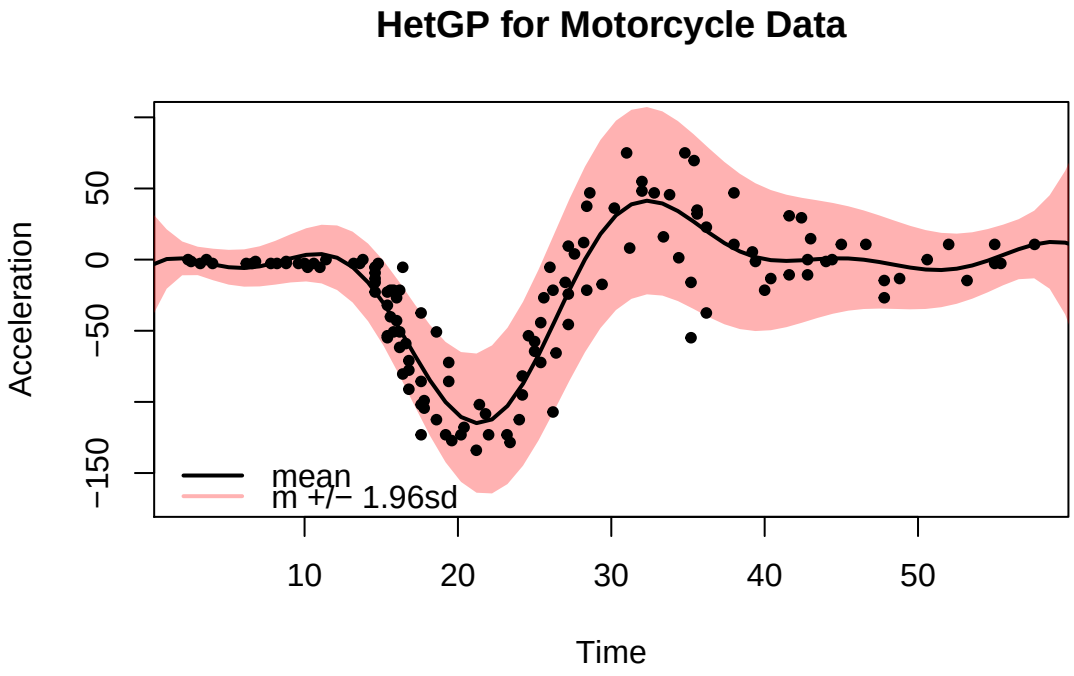

Figure 4.2: The motorcycle data set with a HetGP superimposed. The black line is the emulator mean and the red band corresponds to a 95% posterior predictive interval for $y(x)$.

et al. (2018); Zhang *et al.* (2022) both used hundreds of runs per input dimension; the single input motorcycle example has a sample size of 133.

4.3.4 *Shortcomings of HetGP*

We often construct emulators because the data (simulator runs) are scarce. Let us consider a simple, tractable and heteroscedastic stochastic simulator, defined for $x \in [0, 1]$:

$$y(x) = 4\sin(7\pi x) + 5(2x + 1) + 3\log(x + 0.01) + (5x + 2)\varepsilon \tag{4.14}$$

$$\varepsilon \overset{\text{iid}}{\sim} \mathcal{N}(0, 1). \tag{4.15}$$

This simulator is simple and virtually instantaneous to run, thus is used only for illustrative purposes. We constructed a HetGP emulator for Equation (4.14) using 50 design points. Observing the fit in Figure 4.3, the fitted emulator mean (dashed

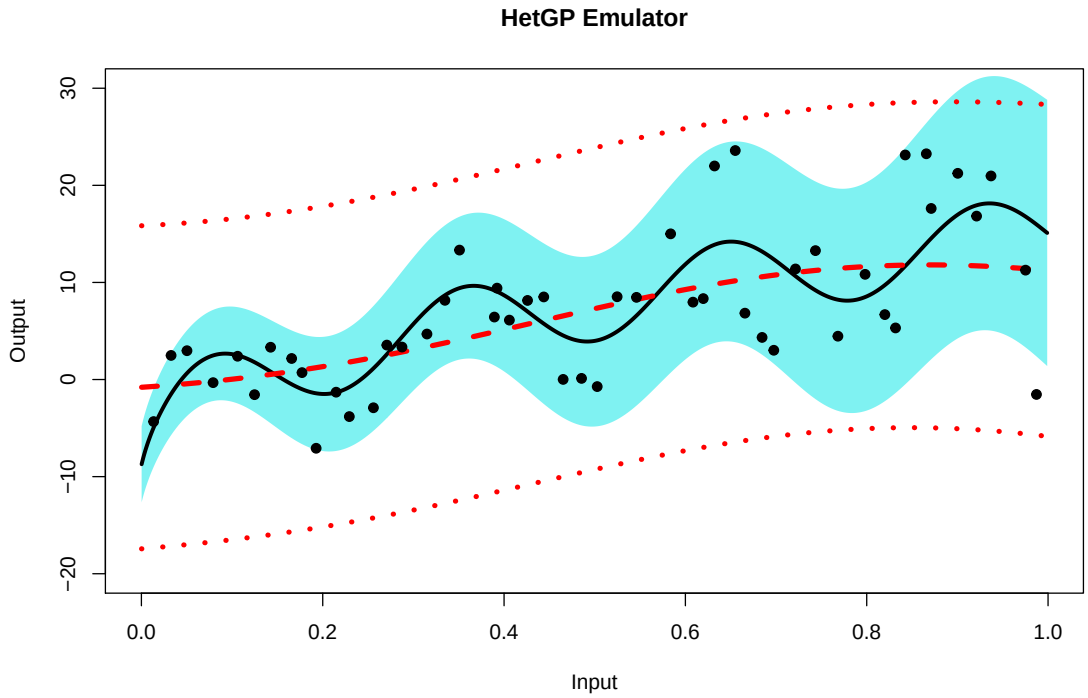

Figure 4.3: A HetGP emulator for Equation (4.14). Black points are the outputs from 50 runs of the simulator. Black line represents the true simulator mean and the blue band represents the mean ±1.96 'true' standard deviations. Red dashed line represents emulator mean with red dotted lines being the emulator mean ±1.96 emulator standard deviations.

red line) does not match up well with the simulator. The emulator predicts an approximately linear response, whereas the simulator is clearly sinusoidal in nature. The emulator is interpreting the systematic sinusoidal variation as noise, rather than signal. Ultimately, this is because emulating a stochastic computer model requires much more information than the standard deterministic problem. However, when provided with adequate amounts of data, HetGP can produce excellent surrogates for complex stochastic computer models (Binois *et al.*, 2018). The confusing of signal and noise is a similar phenomenon to that observed by Andrianakis & Challenor (2012), they observed that (λ, θ) often have a bimodal likelihood when emulating deterministic simulators, with one mode corresponding to large λ. In other words, it is not uncommon for the emulator to favour an overly smooth fit. There are essentially three ways in which a HetGP can interpret variability in a stochastic simulation:

(1) Low signal; high noise. In this case $\lambda^2(\cdot)$ is over-estimated, resulting in an overly-smooth mean response. This is what has happened in Figure 4.3.

(2) Low noise; high signal. This is where $\lambda^2(\cdot) \to 0$ and the mean function interpolates the data as if the simulator were deterministic.

(3) A Goldilocks fit, i.e somewhere in between the other two. In this case, the HetGP will manage to separate the noise from the signal and produce an appropriate fit akin to Figure 4.2.

When simulator runs are sparse, it is quite easy to end up in scenario (1). In our experience, (2) is very rare, but is still possible. To end up in (3), the ideal scenario, we need to either have a relatively large number of simulator runs to allow HetGP to tease the signal from the noise or elicit strong and accurate beliefs about $y(\cdot)$ expressed via $\mu(\cdot)$ and $C(\cdot, \cdot)$. However, if $y(\cdot)$ is a complicated simulator which implements cutting-edge science, there may be limited knowledge of appropriate choices for $\mu(\cdot)$ and $C(\cdot, \cdot)$. It is commonly the case that we are emulating $y(\cdot)$ to perform inference about $\mu(\cdot)$.

4.4 *Stochastic multilevel emulation*

On many occasions, HetGPs require a lot of information, either expressed through a prior specification which accurately reflects simulator behaviour, or in the form of large amounts of training data, to produce an adequate approximation to the simulator under study. To satisfy the appetite of HetGP we can provide HetGP with runs from a related, but cheaper, version of the simulator under study. This motivates the stochastic multilevel (SML) emulator proposed in Kennedy *et al.* (2023), which we will discuss, implement and compare to HetGP. We also compare two estimation routines for the HetGP and SML emulators.

4.4.1 *Motivation and intuition*

In the Athena simulator it is simple to change model features to give us cheap approximations to large offshore wind farm simulations. Since these approximations are relatively computationally cheap, we can use these approximations to get enough training data to construct good emulators. If we can build a good emulator for the cheap simulator, and accurately describe its mean, perhaps we

can utilise this information to build better emulators for more expensive versions of the stochastic computer models.

Exploiting cheap approximations to an expensive simulator has been tackled in the deterministic framework by Kennedy & O'Hagan (2000). The most popular format is their autoregressive structure for functions (Forrester *et al.*, 2007; Singh *et al.*, 2017; Harvey *et al.*, 2018). The autoregressive structure builds a well informed emulator for the cheap simulator and uses this as a "starting point" for the expensive simulator. The main aim of multilevel emulation is an improved emulation of the simulator at a fixed training budget. We extend this approach to the more complex case of stochastic computer experiments and use it to enhance the emulation of the Athena simulator.

In this section, we outline our proposed approach to stochastic multilevel (SML) emulation of stochastic simulators. This naturally extends deterministic emulation techniques and exploits the cheap approximations that are readily available from the Athena simulator. Although our application is within engineering, multilevel emulation for agent based models, which are often heteroscedastic, has been identified as a useful technology (Swallow *et al.*, 2022). We therefore propose a general approach that will apply to many stochastic simulators when cheap approximations are available. Many stochastic computer simulators have a complexity parameter, such as the length of a time step, or granularity of a grid over space, which exchanges simulation accuracy for computational cost; examples include Kennedy & O'Hagan (2000) and Le Gratiet & Garnier (2014). In our wind farm setting this will be the time step, Δt, in the numerical integration used to solve Equation (2.1) many times within each simulation run.

The number of event times is affected by Δt, which generates the random time between events. Accurate runs ($\Delta t = 0.001$) of the Athena simulator take just over 3 minutes for a wind farm with 200 turbines on a desktop PC with $8 \times 3.20\,\text{GHz}$ processors and 16 GB RAM. On the same machine, cheap runs ($\Delta t = 0.1$) take just under 3 seconds. A single expensive run is computationally equivalent to 60 cheap runs. The accuracy required comes at a computational cost which severely limits the size of our computer experiment, reducing the quality of the fitted emulator. We aim to exploit these computational properties in jointly modelling the "cheap" simulator and "expensive" simulator. The outputs from cheap and expensive versions of stochastic simulators will be related. Runs from both versions are combined to build an overall better emulator.

The two levels of the Athena simulator are approximately linearly related; see Figure 4.4. The relationship is not exact, partially due to the stochasticity of the two levels. The relationship flattens off when the probit cheap code exhibits values above about 1.6. We will focus on a two level set up; $y^C(\cdot)$ is the cheap

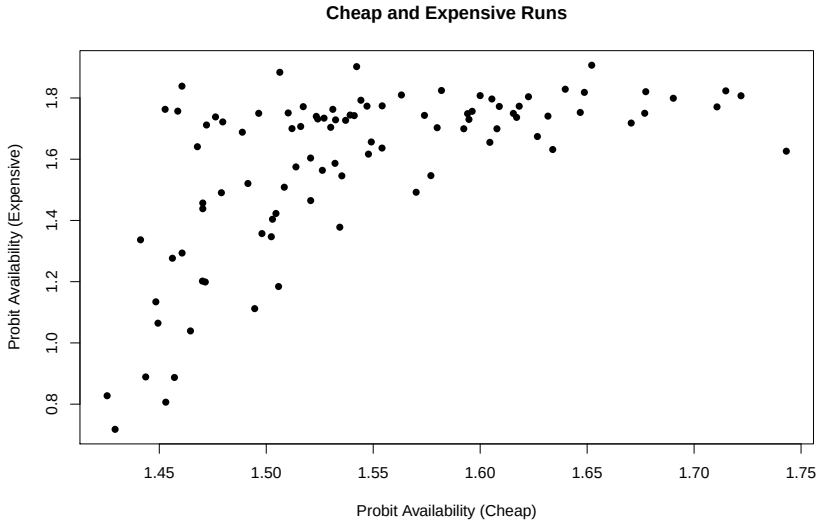

Figure 4.4: Mean probit availability under the expensive version plotted against the cheap version. Each point is computed via 10 replications. We see an approximately linear relationship between the two levels of code but note that the range of the axes is quite different: $0.7 <$ Expensive < 2 but $1.4 <$ Cheap < 1.8.

simulator and $y^E(\cdot)$ is its expensive counterpart. In the motivating example of the Athena simulator $y^C(\cdot)$ is a version of the model with a time step of $\Delta t = 0.1$ years (≈ 1 month). We want to infer $y^E(\cdot)$, which is a version with time step $\Delta t = 0.001$ years (≈ 9 hours).

4.4.2 *Proposed emulation strategy*

We allow for $y^E(\cdot)$ to be heteroscedastic, but if we believe it is homoscedastic we can replace the non-constant variance with a constant term (Roth *et al.*, 2022). Our object of inference is (the distribution of) $y^E(\mathbf{x})$, for any $\mathbf{x}$.

Suppose that the cheap simulator, $y^C(\cdot)$ can be modelled by a homoscedastic (constant noise) GP with mean function $m_C(\cdot)$, covariance function $C_C(\cdot,\cdot)$ and constant variance λ_C^2, that is,

$$y^C(\cdot) \sim \mathcal{GP}\left\{m_C(\cdot), C_C(\cdot,\cdot) + \lambda_C^2 I\right\}, \tag{4.16}$$

or equivalently, $y^C(\cdot) = Z^C(\cdot) + \varepsilon^C(\cdot)$ where $Z^C(\cdot) \sim \mathcal{GP}\{m_C(\cdot), C_C(\cdot,\cdot)\}$ and $\varepsilon^C(\cdot) \sim \mathcal{GP}\{0, \lambda_C^2 \mathbb{I}(\cdot,\cdot)\}$.

We expect that the cheaper simulator's mean is informative for the expensive counterpart and thus, as in Kennedy & O'Hagan (2000), we assume that

$$y^E(\cdot)|\rho, Z^C(\cdot), \delta(\cdot) = \rho Z^C(\cdot) + \delta(\cdot) \tag{4.17}$$

where $y^E(\cdot)$ is the expensive stochastic simulator and $\delta(\cdot)$ is a HetGP such that

$$\delta(\cdot)|\lambda_E^2(\cdot) \sim \mathcal{GP}\left\{m_E(\cdot), C_E(\cdot,\cdot) + \lambda_E^2(\cdot) I\right\} \tag{4.18}$$

$$\log \lambda_E^2(\cdot) \sim \mathcal{GP}\left\{m_V(\cdot), C_V(\cdot,\cdot) + \lambda_V^2 I\right\}, \tag{4.19}$$

where the I are identity matrices of appropriate dimensions. It is also convenient to decompose $\delta(\cdot)$ into a deterministic component and a stochastic component; $\delta(\cdot) = Z^E(\cdot) + \varepsilon^E(\cdot)$ where $Z^E(\cdot) \sim \mathcal{GP}\{m_E(\cdot), C_E(\cdot,\cdot)\}$ is the deterministic component and $\varepsilon^E(\cdot) \sim \mathcal{GP}\{0, \lambda^2(\cdot)\}$ is the stochastic component. In this formulation, $\rho \in \mathbb{R}$ is a regression parameter and $m_E(\cdot)$, $C_E(\cdot,\cdot)$ are mean and covariance functions for $\delta(\cdot)$. The term $\delta(\cdot)$ serves a dual purpose. Firstly, $\delta(\cdot)$ can be viewed as a discrepancy function; the mean of $\delta(\cdot)$ represents the difference in the mean response of the two simulators, or the loss of accuracy from running cheap simulations (with a large time step/coarse grid). Secondly, $\delta(\cdot)$ describes the stochasticity in the expensive simulator. This is a similar structure to that of Bayesian calibration of deterministic computer models (Kennedy & O'Hagan, 2001), however we do not observe data from a physical system — but a computer simulator — and we have noise in both sets of observations.

This joint model for the two simulators allows us to borrow information from the cheaper simulator, but is sufficiently flexible to reject a relationship between the two levels if no such relationship exists. If $\rho = 0$ we recover HetGP.

We express the mean functions in a hierarchical form so that $m_C(\mathbf{x}) = h(\mathbf{x})^T \boldsymbol{\beta}_C$ and $m_E(\mathbf{x}) = h(\mathbf{x})^T \boldsymbol{\beta}_E$. We take $h(\cdot)$ to be a set of known, deterministic basis functions. The mean functions have the same form; the particular parameters of these regression functions are allowed to differ.

We will use squared exponential covariance functions so that

$$C_*(\mathbf{x}, \mathbf{x}') = \sigma_*^2 \exp\left\{-(\mathbf{x} - \mathbf{x}')^T D_*^{-1} (\mathbf{x} - \mathbf{x}')\right\}$$

where $* \in \{C, E\}$, $D_* = \text{diag}\,(\theta_{1,*}^2, \ldots, \theta_{K,*}^2)$ is a diagonal matrix containing the correlation lengthscales and σ_* are scale parameters of the covariance functions. Note that the choice of squared exponential covariance function is not a requirement; the user can specify a different covariance structure as they see fit (Rasmussen & Williams, 2006).

Since we are only interested in the cheap simulator's mean, we do not consider that it is necessary to estimate a surface for its variance. In fact, the homoscedastic GP is quite good at learning the mean response surface, even in the face of heteroscedasticity (see Fig. 5 of Binois *et al.* (2019)). In our model formulation, λ_V is a constant nugget for the latent variance of the expensive simulator. Both λ_C and λ_V smooth the noisy simulator observations. Hence a SML emulator has a similar structure to the standard multilevel emulators presented by Kennedy & O'Hagan (2000), with the addition of a latent variance process ($\lambda_E^2(\cdot)$). We model the log variance as a GP to enforce positivity.

It follows that, conditional on all hyperparameters, $\boldsymbol{Y}^T = \left((\boldsymbol{Y}^C)^T, (\boldsymbol{Y}^E)^T\right) = (Y_1^C, \ldots, Y_{N_C}^C, Y_1^E, \ldots, Y_{N_E}^E)^T$ are multivariate normal where N_C and N_E are the number of runs of the cheap and expensive simulators, respectively. That is,

$$\begin{pmatrix} \boldsymbol{Y}^C \\ \boldsymbol{Y}^E \end{pmatrix} \mid \Theta \sim \mathcal{N}_{N_C+N_E}\left\{ \begin{pmatrix} m_C(X^C) \\ \rho m_C(X^E) + m_E(X^E) \end{pmatrix}, \text{Var}\,(\boldsymbol{Y} \mid \Theta) \right\} \tag{4.20}$$

where X^C and X^E are sets of input vectors of the cheap and expensive versions of the simulator, respectively. Details of the design we use are given in Section 4.5.2.

We now derive the covariance matrix of the response $\boldsymbol{Y}$. We write this covariance matrix in block form

$$\text{Var}\,(\boldsymbol{Y} \mid \Theta) = \begin{pmatrix} \text{Var}\,(\boldsymbol{Y}^C \mid \Theta) & \text{Cov}\,(\boldsymbol{Y}^C, \boldsymbol{Y}^E \mid \Theta) \\ \text{Cov}\,(\boldsymbol{Y}^E, \boldsymbol{Y}^C \mid \Theta) & \text{Var}\,(\boldsymbol{Y}^E \mid \Theta) \end{pmatrix}. \tag{4.21}$$

The auto-covariance of $\boldsymbol{Y}^C$ is

$$\text{Var}\,(\boldsymbol{Y}^C \mid \Theta)_{i,j} = \sigma_C^2 \exp\left\{-(\mathbf{x}_i^C - \mathbf{x}_j^C)^T D_C^{-1} (\mathbf{x}_i^C - \mathbf{x}_j^C)\right\} + \lambda_C^2 \mathbb{I}_{\mathbf{x}_i^C, \mathbf{x}_j^C},$$

where $\mathbb{I}_{i,j}$ is an indicator function equal to 1 when $i = j$ and 0 otherwise. For the auto-covariance of the expensive simulator, we assume the three summed GPs are

all pairwise independent and that the constant variance of the cheap simulator is independent of the variance of the expensive simulator. Further we assume, for $i \neq j$, that

$$\text{Cov}\left(Z^C(\mathbf{x}_i), Z^E(\mathbf{x}_j)\right) = 0 \tag{4.22}$$
$$\text{Cov}\left(Z^C(\mathbf{x}_i), \varepsilon^E(\mathbf{x}_j)\right) = 0 \tag{4.23}$$
$$\text{Cov}\left(\varepsilon^C(\cdot), Z^E(\cdot)\right) = 0 \tag{4.24}$$
$$\text{Cov}\left(\varepsilon^C(\mathbf{x}_i), \varepsilon^E(\mathbf{x}_j)\right) = 0 \tag{4.25}$$
$$\text{Cov}\left(Z^E(\mathbf{x}_i), \varepsilon^E(\mathbf{x}_j)\right) = 0, \tag{4.26}$$

where $Z^C(\mathbf{x}) = \text{E}\left\{y^C(\mathbf{x})\right\}$. Thus we find that

$$\begin{aligned}\text{Var}\left(\boldsymbol{Y}^E \mid \Theta\right)_{i,j} &= \text{Cov}\left(Y^E(\mathbf{x}_i^E), Y^E(\mathbf{x}_j^E) \mid \Theta\right) &(4.27)\\ &= \rho^2\sigma_C^2 \exp\left\{-(\mathbf{x}_i^E - \mathbf{x}_j^E)^T D_C^{-1}(\mathbf{x}_i^E - \mathbf{x}_j^E)\right\} \\ &\quad + \sigma_E^2 \exp\left\{-(\mathbf{x}_i^E - \mathbf{x}_j^E)^T D_E^{-1}(\mathbf{x}_i^E - \mathbf{x}_j^E)\right\} + \lambda_E^2(\mathbf{x}_i^E)\mathbb{I}_{\mathbf{x}_i^E, \mathbf{x}_j^E}, &(4.28)\end{aligned}$$

Finally, the cross-covariance is given by

$$\text{Cov}\left(\boldsymbol{Y}^C, \boldsymbol{Y}^E \mid \Theta\right)_{i,j} = \rho C_C(\mathbf{x}_i, \mathbf{x}_j). \tag{4.29}$$

4.5 *Bayesian approach to SML*

Adopting a multivariate Normal prior for the $\boldsymbol{\beta}$ parameters allows them to be analytically integrated out. For example, if we take

$$\begin{pmatrix}\boldsymbol{\beta}^C \\ \boldsymbol{\beta}^E\end{pmatrix} \sim \mathcal{N}(\boldsymbol{b}, B)$$

then we can write $\boldsymbol{Y} \mid \Theta_{-\boldsymbol{\beta}} \sim \mathcal{N}\left\{H\boldsymbol{b}, K_0\right\}$ as the prior for $\boldsymbol{Y}$ conditional on the GP covariance matrix, where $K_0 = \text{Var}\left\{\boldsymbol{Y} \mid \Theta\right\} + HBH^T$ and H is the design matrix. Details of H are discussed in Section 4.5.3.

4.5.1 *Prior specification*

Since a Bayesian approach to inference is adopted, we assign priors to all GP parameters. We propose that all parameters are assumed independent *a priori* with the following distributions (where the hyperparameters of the prior are chosen by the user),

$$\beta_j^* \sim \mathcal{N}(m_{j,*}, s_{j,*}^2) \qquad \theta_{j,*} \sim Gamma(a_{j,*}, b_{j,*}) \tag{4.30}$$

$$\sigma_* \sim Inv-Gamma(c_{j,*}, d_{j,*}) \qquad \rho \sim \mathcal{N}(m_\rho, s_\rho^2) \tag{4.31}$$

where $* \in \{C, E, V\}$ and $\lambda_*^2 \sim Inv-Gamma(e_{j,*}, f_{j,*})$ for $* \in \{C, V\}$. Note the absence of λ_E^2 since we replace this by a GP to account for heteroscedasticity. For β_j^* we adopt independent $\mathcal{N}(0,1)$ priors. Because our GP is on the probit scale — and our inputs are mean centred and scaled to have unit variance — this prior covers a wide range of observable values; a more diffuse prior (say $s_{j,*}^2 = 10$) would imply that the simulator output will be very close to either 0 or 1 but not between. Our priors on θ_* will be reasonably uninformative, but designed to omit very large lengthscales, therefore we take $a_{j,*} = 2$ and $b_{j,*} = 1$. Fairly weak priors are taken over σ_* $c_{j,*} = d_{j,*} = 2$ and for λ_*^2 we have $e_j = f_j = 2$. In the prior for ρ we are being quite subjective, we take $m_\rho = 1$ and $s_\rho = 1/3$. This specification expresses the belief that the simulator versions are positively correlated with a high probability; this is a reasonable assertion (recall Figure 4.4). If this belief was not held, then there would be little reason to construct a multilevel emulator. This specification is *our* prior specification. In practice, a user can choose a prior that they see as suitable.

4.5.2 *Design*

We require a space filling design for both the cheap and expensive versions of the simulator, hence we will appeal to a nested design based on Latin hypercubes. We generate X^E via a maximin Latin hypercube (Morris & Mitchell, 1995) (using the `lhs` package in `R` (Carnell, 2022)). To generate X^C we make another maximin Latin hypercube and append the two designs together. We run both the cheap and expensive versions of the simulator at X^E, but run only the cheap simulator at X^C.

4.5.3 *Posterior predictive distribution of code output*

Within our Bayesian approach, we will use an empirical Bayes (EB) approach. As with HetGP, we integrate out all $\boldsymbol{\beta}$ parameters analytically. The remaining parameters — those of the GP covariance structure — are found via a numerical optimisation of the log-posterior (up to an additive constant) using the `optimizing` function from `rstan` (Stan Development Team, 2020), which uses the L-BFGS algorithm for numerical optimisation. This effectively gives us a MAP estimate of the GP covariance structure. To prevent a local mode being chosen as the maximiser, we recommend running the `optimizing` function multiple times and selected the best result. In our work, we run `optimizing` three times. Recall that the log posterior density is

$$\log \pi(\Theta \mid \mathcal{D}) = \log \{\pi(\Theta) L(\Theta \mid \mathcal{D})\} - \log \left\{ \int \pi(\tilde{\Theta}) L(\tilde{\Theta} \mid \mathcal{D}) \mathrm{d}\tilde{\Theta} \right\}. \tag{4.32}$$

Note that the intractable log evidence term does not depend on Θ, and thus vanishes upon differentiation by Θ. Therefore maximisation of Equation (4.32) is equivalent to maximising any function of the form $g(\Theta) = \log \{\pi(\Theta) L(\Theta \mid \mathcal{D})\} + C$. This EB approach is not fully Bayesian, however it is computationally thrifty. After integrating out the $\boldsymbol{\beta}$ coefficients, we condition on point estimates of the remaining parameters to obtain the posterior distribution for $\log \lambda_E^2(X^*)$. The posterior at new inputs X^* is Gaussian with mean

$$m_V^*(X^*) = H_V^* \boldsymbol{b}_V + K_V(X^*, X^E) \left\{ K_V(X^E, X^E) + \lambda_V^2 I_E \right\}^{-1} (\log(\lambda_E^2(X^E)) - H_V \boldsymbol{b}_V)$$

where $K_V(\cdot, \cdot)$ is the same as for HetGP.

Prediction of $y^E(X^*)$ is more complex, but is a natural extension of the posterior predictive mean of a two-level code given in Kennedy & O'Hagan (2000). Having

observed code outputs $\boldsymbol{Y}^C$, $\boldsymbol{Y}^E$ at design points X^C, X^E, our design matrix is

$$H = \begin{pmatrix} h(\mathbf{x}_1^C)^T & \mathbf{0} \\ \vdots & \vdots \\ h(\mathbf{x}_{N_c}^C)^T & \mathbf{0} \\ \\ \rho h(\mathbf{x}_1^E)^T & h(\mathbf{x}_1^E)^T \\ \vdots & \vdots \\ \rho h(\mathbf{x}_{N_E}^E)^T & h(\mathbf{x}_{N_E}^E)^T \end{pmatrix} \tag{4.33}$$

and hence the posterior distribution of the output of the expensive simulator at new inputs X, conditional on a point estimate of $\Theta_{-\boldsymbol{\beta}}$, is Gaussian with mean

$$m^*(X^*) = h_0(X^*)^T(\rho\boldsymbol{\beta}^C, \boldsymbol{\beta}^E) + t(X^*)K_0^{-1}\left(\boldsymbol{Y} - H\boldsymbol{b}\right). \tag{4.34}$$

where $h_0(X^*) = (h(X^*), h(X^*))$ and $t(X^*) = \mathrm{Cov}\left(y^E(X^*), \boldsymbol{Y}\right)$. If we take $B = \mathrm{diag}\left(B^C, B^E\right)$ to be a block diagonal matrix of variance matrices where the blocks are of equal dimension, then the posterior variance, conditional on $\Theta_{-\boldsymbol{\beta}}$, can be expressed as

$$\begin{aligned} v^*(X^*) = \rho^2 C_c(X^*, X^*) + C_E(X^*, X^*) + h(X^*)(\rho^2 B^C + B^E)h(X^*)^T \\ + \lambda_E^2(X^*)I - t(X^*)K_0^{-1}t(X^*)^T, \end{aligned} \tag{4.35}$$

To get a flavour for SML emulation we have produced an SML emulator in Figure 4.5 for the simulator described in Section 4.3.4. We used 46 of the runs from the HetGP emulator in Figure 4.3 and replaced the remaining 4 runs with 400 runs from a 'cheap' simulator $y^C(x) = 4\sin(7\pi x) + 4\varepsilon$ with $\varepsilon \sim \mathcal{N}(0, 1)$ and $x \in [0, 1]$. The cheap points have a similarly shaped mean function to the expensive points. This information is utilised by the SML emulator to provide an emulator which closely mimics $y(\cdot)$.

4.6 *Fitting HetGP and SML emulators to Athena*

As we discussed in the first chapter, one task a subjective Bayesian may want to perform is a probabilistic sensitivity analysis (PSA) to better plan an elicitation

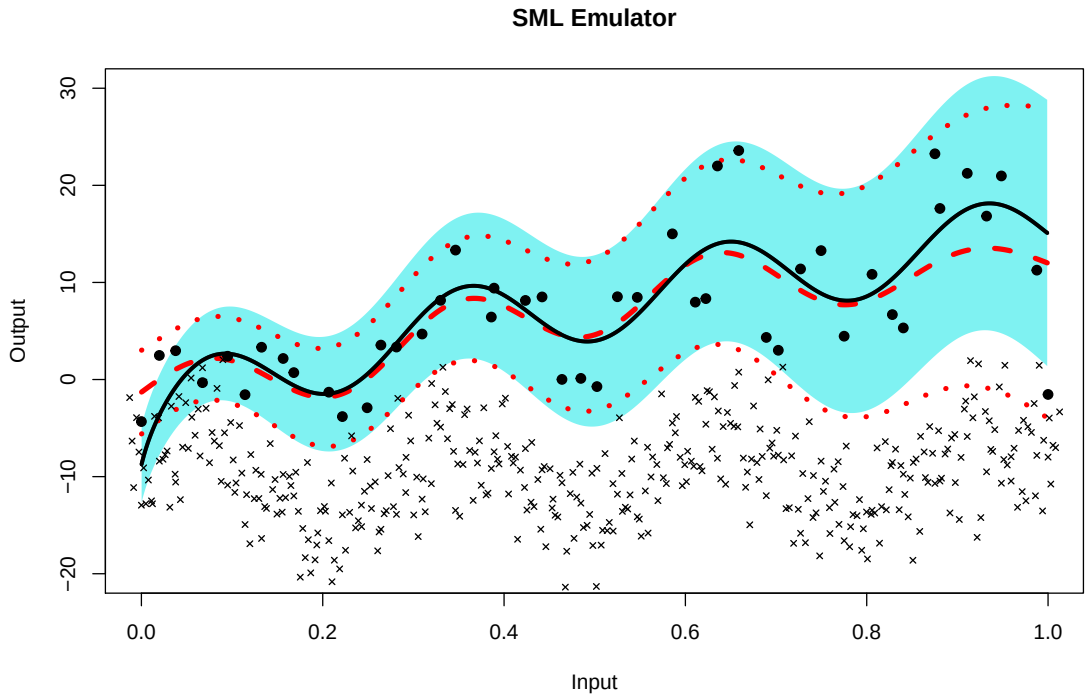

Figure 4.5: A SML emulator for $y(\cdot)$. Expensive runs are black points and cheap runs are black crosses (which are offset by -10 to aid visualisation). The true simulator is represented by the black line (mean function) and blue band (± 1.96 standard deviations). The emulator is represented by the red dashed line (emulator mean) and red dotted lines (± 1.96 emulator standard deviations).

exercise. Since Athena is stochastic, we want to understand

(i) which parameters influence the mean response

(ii) which parameters influence the stochasticity.

To perform such a task, with an expensive simulator such as Athena, we first fit an appropriate emulator. HetGP and SML emulators provide what we need here as they provide flexible estimates of both $\mathrm{E}\{y(\mathbf{x})\}$ and $\mathrm{Var}\{y(\mathbf{x})\}$. A key output of Athena is availability, $A(\mathbf{x}) \in (0, 1)$. Since $A(\mathbf{x}) \in (0, 1)$ we will construct emulators for $y(\mathbf{x}) = \text{probit}\{A(\mathbf{x})\} = \Phi^{-1}\{A(\mathbf{x})\} \in \mathbb{R}$. We will compare a total of 4 approaches; two estimation routines for fitting HetGP to Athena as well as two estimation routines for fitting SML to Athena.

The first approach will be an EB approach; we will integrate out any β terms in the emulator and find point (MAP) estimates of aspects of the covariance

functions. Here we *will not* consider a full-Bayes approach (implemented via MCMC). This is because a full-Bayes approach is incredibly computationally costly due to (a) poor mixing in latent variable models and (b) multiple matrix inversions being required for each iteration of the MCMC scheme (Kersting *et al.*, 2007).

In our second approach, we will fit the emulators via the 'full' MAP approach (which can also be thought as a penalised log-likelihood approach). Point estimates are the standard approach to fitting HetGP models and their variants (Binois *et al.*, 2019). MAP estimates can provide improved estimates of lengthscales, as lengthscale parameters can have a very flat likelihood. Comparing MAP estimates to an EB approach will also allow us to assess how important uncertainty quantification about the β parameters is.

4.6.1 *Emulator construction*

We construct emulators over a 9 dimensional input space. The input spaces consists of the mean time to degradation (start of phase (3)) of a Bathtub hazard function for the 9 sub-assemblies in the wind farm. Recall that the 9 sub-assemblies as follows: 1. gearbox, 2. generator, 3. frequency converter (freq. conv.), 4. transformer, 5. main shaft bearing (MSB), 6. blades, 7. tower, 8. foundations, and 9. catch all. We vary each input over the range $[0.1, 5]$ (years). The Athena simulator is flexible enough to specify unique parameters for every subassembly in every turbine. We give the same parameter values to each subassembly of a given type and allow different types of subassembly to have different parameters. For example, all gearboxes could have a mean time to degradation of 1 year whereas all generators could have a mean time to degradation of 3.2 years.

Design points are chosen via the structure described in Section 4.5.2. To construct the HetGP emulator we ran the Athena simulator at 100 design points. The cheap runs of the simulator were fast enough that we could trade just 5 expensive design points for 295 cheap runs. We used basis functions $h(\mathbf{x}) = (1, \log(\mathbf{x}))$ for the mean functions of the mean response. We arrived at this selection to reflect a prior belief that the mean availability would flatten off at larger values of x_i. The covariance function assumes standardised inputs, x_i^*. Standardisation is achieved by subtracting the sample means and then dividing by the sample standard deviations (of the expensive training data). The latent variance GP has mean function $m_V(\mathbf{x}) = (1, \mathbf{x})\beta_V$ and again, the covariance function assumes standardised inputs.

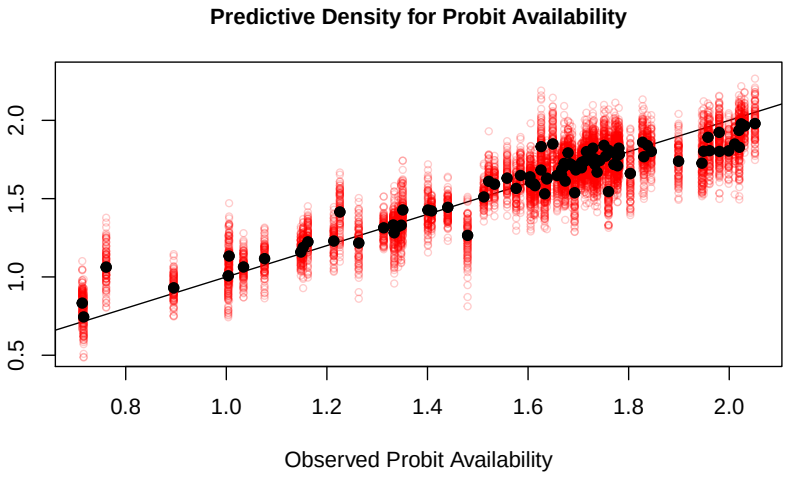

Figure 4.6: SML emulator mean predictions plotted against observed probit availability (training data from expensive runs of Athena; black dots) and 100 realisations from the emulator at each point (translucent red circles).

Figure 4.6 shows the emulator predictions of the training data (probit scale) with realisations from $\mathcal{N}\{m^*(\mathbf{x}), v^*(\mathbf{x})\}$ around each prediction. Large deviations from the unit diagonal are typically accompanied by a more diffuse predictive distribution; the emulator is giving larger variance to the points which are far away from the mean. We also see that the observed probit availabilities are mostly in the region of 0.5 – 2 (availabilities in the region of around 0.76 – 0.98). The full range of observed availabilities is $(0.762, 0.980)$; the vast majority of realisations from the emulator agree with this range.

4.6.2 *Emulator performance comparison*

To judge relative performance of each emulator we use the performance metrics outlined in the previous chapter; RMSE and Score (Equations 3.40 and 3.41). Using 100 independently generated validation data points, the RMSE (on the probit scale) for HetGP was 0.181, whereas SML achieved an RMSE of 0.156. The score for HetGP was 238 and for SML the score was 254 (probit scale). Since we transformed the availability to construct emulators on an unbounded space, we should also check how predictions perform on the $[0, 1]$ scale. Using an inverse-probit transformation on the mean function provides a sensible point estimate of availability. Comparing the RMSE on the original scale we observe an RMSE of

0.0272 for HetGP, and under SML this is reduced to 0.0198. Hence, SML achieves better RMSE and score here than HetGP for the Athena model, suggesting it is a better emulator. Further, our point estimate of ρ is $\hat{\rho} = 0.51$. This suggests a moderate correlation between the two versions of Athena. The additional information extracted from cheap simulations has improved our emulation with little computational cost. It took 5.7 seconds to fit HetGP and 29.6 seconds to fit SML on a laptop with $4 \times 2.40\,\text{GHz}$ processors and 8 GB RAM. Although SML took more time to fit, in real terms this is about 30 seconds of computation time — less than a single expensive run of Athena. Both timings are for a total of 3 fits of the emulator. We performed 3 fits to prevent using a local maximum being chosen for the point estimates.

4.6.3 *Emulator validation*

To validate the emulators, we will implement some graphical diagnostics proposed by Bastos & O'Hagan (2009). Since we model the (transformed) simulator outputs by a Gaussian process, the Cholesky errors (CEs) should form a random sample from a $\mathcal{N}(0,1)$ distribution (approximately). If the posterior mean and variance are well suited to the simulator, the validation data should lie in a horizontal band, centred at 0, with approximately 95% of points in the interval $(-1.96, 1.96)$. We also compare empirical quantiles of the CEs against theoretical quantiles — we do this via coverage plots which compare the proportion of CEs in the $100(1-\alpha)\%$ prediction interval against the expected proportion. In Figure 4.7 the CEs for HetGP have a distinct pattern when plotted against x_6, whereas for SML in Figure 4.8 the points appear to be closer to a random $\mathcal{N}(0,1)$ sample. The coverage plots in Figure 4.9 suggest that for both emulators the coverage is reasonably well calibrated.

4.7 *MAP approach*

This chapter so far has taken a Bayesian approach to HetGP and SML. Although not fully Bayesian, it takes into account some parameter uncertainty without the often large computational cost of MCMC. In the Athena example, the Bayesian SML emulator provides an accurate and efficient approach to emulation. Although the Bayesian HetGP was not adequate for this example, it was very fast to

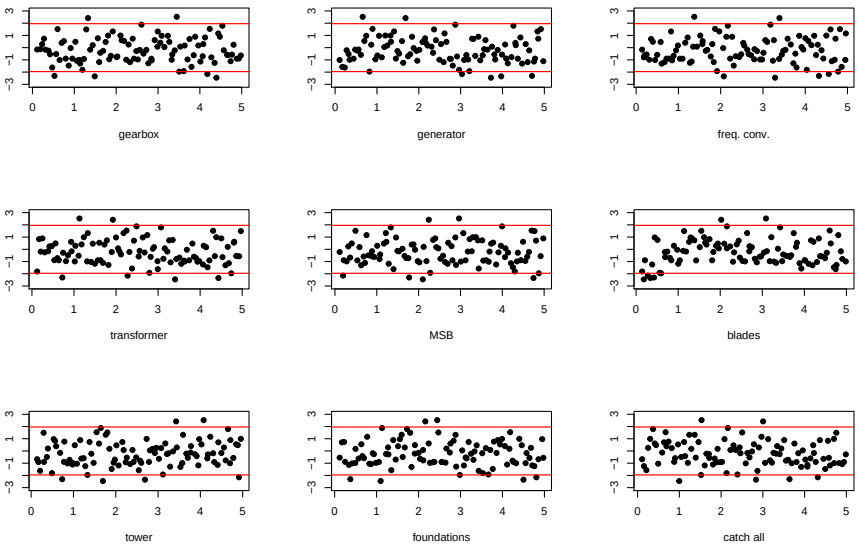

Figure 4.7: Cholesky errors for HetGP, based on 100 "unseen" validation points. Orange lines are at ±1.96.

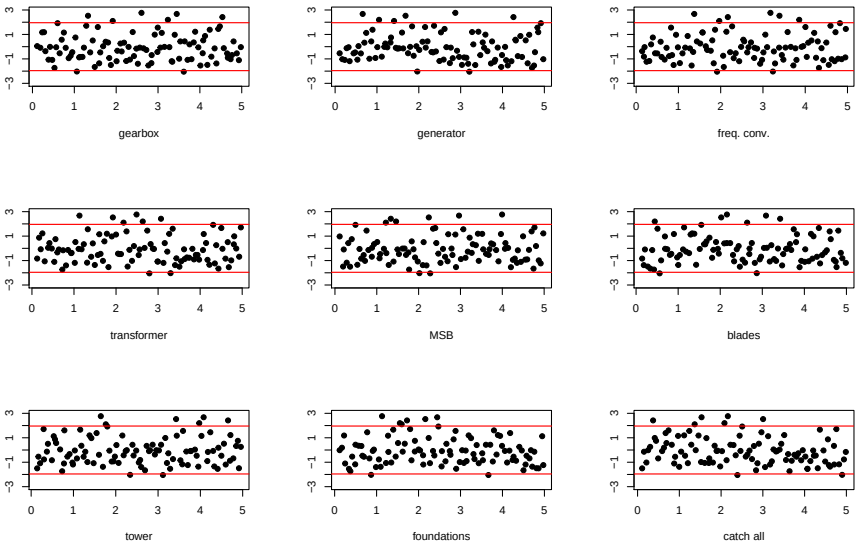

Figure 4.8: Cholesky errors for SML emulation, based on 100 "unseen" validation points. Orange lines are at ±1.96.

fit thus could provide useful, and more interpretable, emulators for other stochastic simulators.

However, the majority of the literature uses a MAP/MLE approach to estimate GP hyperparameters and for prediction of simulator output (Binois *et al.*, 2018;

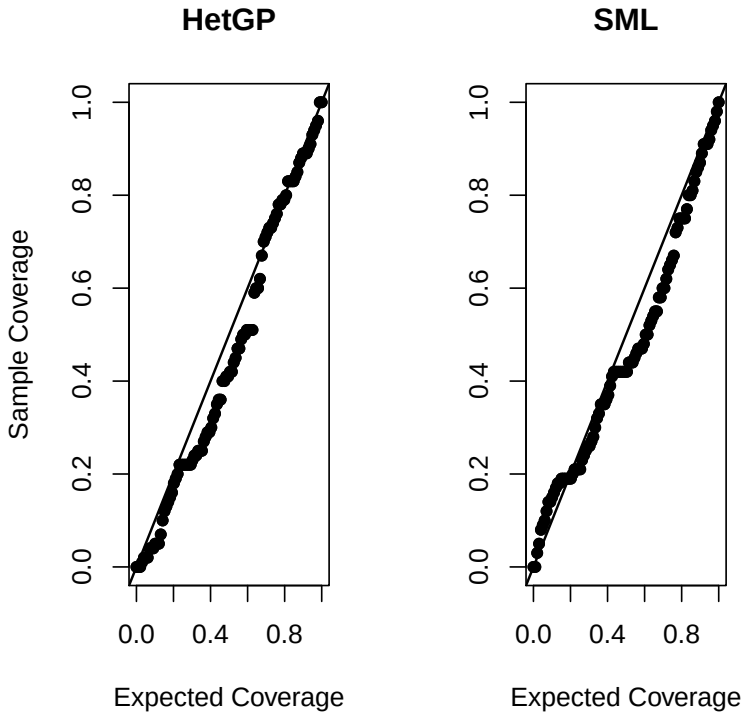

Figure 4.9: Out of sample coverage plots (black dots), using the Cholesky errors of the "unseen" validation data. Black lines represent the unit diagonal.

Baker *et al.*, 2020*a*,*b*, 2022; Zhang *et al.*, 2022). Since this is a mathematically simpler approach, which does not rely on specific forms of prior specification, we will investigate whether using the MAP estimate offers improved emulator accuracy and/or improves the computational properties of the emulator.

4.7.1 *MAP-HetGP prediction*

With the MAP approach, *all* parameters are estimated by their posterior mode. Therefore we will re-define $\Theta := \{\Theta, \boldsymbol{\beta}, \boldsymbol{\beta}_V\}$. Given $\{\mathcal{D}, \Theta\}$, prediction follows in a similar way to the Bayesian approach.

As in the Bayesian approach, we first predict $\log \lambda^2(X^*)$. The predictive distribution of $\log \lambda^2(X^*) \mid \widehat{\Theta}, \log \hat{\lambda}^2(X), \mathcal{D}$ is given by

$$\log \lambda^2(X^*) \mid \widehat{\Theta}, \log \hat{\lambda}^2(X), \mathcal{D} \sim \mathcal{N}\{m_V^*(X^*), v_V^*(X^*)\} \tag{4.36}$$

where the posterior moments are

$$m_V^*(X^*) = H_V^*\hat{\boldsymbol{\beta}}_V + C_V(X^*, X)\left\{C_V(X, X) + \hat{\lambda}_V^2 I\right\}^{-1}(\log\hat{\lambda}^2(X) - H_V\hat{\boldsymbol{\beta}}_V) \quad (4.37)$$

$$v_V^*(X^*) = C_V(X^*, X^*) - C_V(X^*, X)\left\{C_V(X, X) + \hat{\lambda}_V^2 I\right\}^{-1} C_V(X, X^*) + \hat{\lambda}_V^2 I. \quad (4.38)$$

As with the Bayesian approach, there are three plug-in estimates of $\lambda^2(X^*)$ that can be used. For simplicity, we take $\hat{\lambda}^2(X^*) = \exp\{m_V^*(X^*)\}$. With this estimate, we can then write the predictive distribution of $y(X^*)$ as

$$y(X^*) \mid \hat{\Theta}, \hat{\lambda}^2(X), \mathcal{D} \sim \mathcal{N}\{m^*(X^*), v^*(X^*)\} \quad (4.39)$$

where the posterior moments are

$$m^*(X^*) = H^*\hat{\boldsymbol{\beta}} + C(X^*, X)\left\{C(X, X) + \hat{\lambda}^2(X)I\right\}^{-1}(y(X) - H\hat{\boldsymbol{\beta}}) \quad (4.40)$$

$$v^*(X^*) = C(X^*, X^*) - C(X^*, X)\left\{C_V(X, X) + \hat{\lambda}^2 I\right\}^{-1} C(X, X^*) + \hat{\lambda}^2(X^*)I. \quad (4.41)$$

Note that these predictive equations are the same as those derived for the EB approach with $B = 0$, $B_V = 0$, $\boldsymbol{b} = \hat{\boldsymbol{\beta}}$ and $\boldsymbol{b}_V = \hat{\boldsymbol{\beta}}_V$.

4.7.2 *MAP-SML prediction*

Much like MAP-HetGP, MAP-SML prediction is obtained by taking the Bayesian version of the predictive equations but setting prior variance matrices to zero matrices and the regression coefficients to their MAP estimates. The design is independent of the statistical paradigm thus we use exactly the same design as above. Since the predictive equations are a direct consequence of the multivariate Normal equations, we present them without proof.

As usual, we start with prediction of the log variance; $\log\lambda_E^2(X^*) \mid \Theta$ is multivariate normal with the following first two moments

$$m_V^*(X^*) = H_V^*\hat{\boldsymbol{\beta}}_V + C_V(X^*, X)\left\{C_V(X, X) + \hat{\lambda}_V^2 I\right\}^{-1}(\log\hat{\lambda}^2(X) - H_V\hat{\boldsymbol{\beta}}_V) \quad (4.42)$$

$$v_V^*(X^*) = C_V(X^*, X^*) - C_V(X^*, X)\left\{C_V(X, X) + \hat{\lambda}_V^2 I\right\}^{-1} C_V(X, X^*) + \hat{\lambda}_V^2 I. \quad (4.43)$$

The quantity of interest, $y^E(X^*)$, is also multivariate Normal, conditional on

Θ, $\lambda_E^2(X)$ and $\lambda_E^2(X^*)$ with mean and variance

$$m_E^*(X^*) = H_V^*\hat{\boldsymbol{\beta}}_V + C_0(X^*)\text{Var}\left(\boldsymbol{Y} \mid \Theta\right)^{-1}(\log\hat{\lambda}^2(X) - H_V\hat{\boldsymbol{\beta}}_V) \tag{4.44}$$

$$v_E^*(X^*) = C_0(X^*, X^*) - C_0(X^*, X)\left\{C_0(X, X) + \hat{\lambda}_V^2 I\right\}^{-1} C_0(X, X^*) + \hat{\lambda}_E^2(X^*)I \tag{4.45}$$

where $C_0(X', X) = \text{Cov}\left(y^E(X'), y^E(X)\right) = \hat{\rho}^2 C_C(X', X) + C_E(X, X')$.

4.7.3 *Performance of MAP estimation*

The advantage of MAP estimation is that it is simpler and, if the user desired, a suitably diffuse prior would return an approximate maximum likelihood estimate. Ultimately, we want to know which performs best for our Athena example. Using the same priors as the Bayesian approach to produce a MAP estimate, we refit the emulators and re-compute out of sample performance metrics using exactly the same validation data as before. The performance of the MAP-based, and Bayesian, emulators are summarised in Table 4.1. All metrics in this table indicate that the Bayesian SML emulator is the best approximation to the Athena simulator. The Bayesian SML emulator does not have the fastest estimation routine, the Bayesian HetGP is fastest. However, this computation time is trivial relative to the computational cost of uncertainty analysis in a computationally expensive simulator such as Athena. The MAP-SML emulator has comparable performance to the Bayesian HetGP emulator in terms of RMSE (probit scale) and score, but the RMSE ($(0, 1)$ scale) is quite large. The MAP-SML emulator took a relatively long time to fit and will be slower to implement due to the larger design matrix. The MAP-HetGP offers a marginally better RMSE than its Bayesian equivalent, however the uncertainty quantification under the MAP approach is poor. The score is only 136 (the Bayesian version had a score or 238), this poor performance is exemplified by the C.E.s of the Map-HetGP which have a range of $(-3.85, 4.33)$. This stresses that incorporating parameter uncertainty into predictions, especially when the sample size is small, is a wise idea. The Bayesian estimation was much faster than the MAP estimates, this is because the number of parameters to be estimated is much larger in the MAP approach. Assuming each regression function has p inputs and each covariance function has k inputs, the number of parameters for HetGP is $2p + 2k + 3$. For SML there are $3p + 3k + 6$. In our example this equates to 41 parameters for HetGP and 63 for SML. The Bayesian approach

	MAP		Bayes	
Metric	HetGP	SML	HetGP	SML
Time to fit emulator/seconds	37	307	**6**	30
RMSE (probit scale)	0.179	0.171	0.181	**0.156**
RMSE ((0, 1) scale)	0.0248	0.0231	0.0272	**0.0198**
Score	136	248	238	**254**

Table 4.1: Performance summaries for the emulators based on both the MAP and Bayesian versions of the HetGP and SML emulators. Time here denotes the time take for a total of 3 runs of each estimation routine. RMSE and score are correct to 3 s.f., time is correct to the nearest second. For each row, the bold typeface indicates the best value (largest score is best, for all others, smaller is better).

removes the p terms from each number of parameters; this reduces the dimension of the parameter space to 21 for HetGP and 33 for SML.

4.8 *Summary*

We have reviewed some approaches to emulating stochastic computer models which exhibit input dependent noise. We began with approaches which emulate the mean and variance as separate components and then moved to a single, but more complex, emulator approach. We illustrated that HetGPs can suffer from poor predictive performance in the low-data regime, which is unfortunate since emulators are typically constructed when data (simulator runs) are expensive to obtain. We offered a solution to this by adapting the autoregressive model from Kennedy & O'Hagan (2000).

We have introduced a stochastic multilevel emulator, which adopted elements of (i) the autoregressive structure from Kennedy & O'Hagan (2000) to construct a more accurate mean function and (ii) the latent variance structure of HetGP (Goldberg *et al.*, 1998; Binois *et al.*, 2018) to account for the heteroscedastic nature of the Athena simulator. This structure allowed us to link together two versions of the Athena simulator to construct an emulator with improved accuracy in terms of RMSE and also in terms of score. The easy to generate training data allowed us to build an adequate emulator without relying on a large computational budget to generate training data.

It would be interesting to see, from a methodological point of view, how the SML emulator could be improved. One idea would be to implement a sequential

design rule similar to that of Le Gratiet & Cannamela (2015), that is, minimising some design criterion such as integrated mean squared prediction error. Another idea would be to use a preliminary round of simulations to see where cheap simulations might be most beneficial. For example, it may be advantageous to place more cheap points where the two levels agree most and then retain the expensive simulation budget for areas where the two levels disagree. It would also be interesting to see if replicates could improve this type of emulator in the same way that replicates benefit HetGP. Replication could be especially beneficial in the cheap simulator; this would help to reduce the size of computational overheads of a large design matrix, since inference is $O(N^3)$ for HetGP and $O\left((N_C + N_E)^3\right)$ for SML, however Table 4.1 showed that the raw computation times for the Athena simulator were not particularly large. Prediction is $O\left(N^2\right)$ for HetGP and $O\left((N_C + N_E)^2\right)$ for SML. Another possibility would be to link the variance of the two simulators; we chose not to do this as it would involve linking two latent variance processes and would involve inversion of a large matrix, increasing the computational cost of inference and prediction.

A joint emulator is not the only way to use knowledge of cheap simulations to enhance emulation of an expensive simulator. Using a large number of cheap simulations, we can construct an emulator for $y^C(\cdot)$. The joint posterior over the GP hyperparameters and the β coefficients can be used to construct an informative prior for $y^E(\cdot)$. This could be more computationally efficient than the SML emulator, but would likely require more human input. The informative prior would then be used as part of a HetGP emulator for $y^E(\cdot)$. Cumming & Goldstein (2010) use a large number of runs from a cheap, deterministic simulator to build an informative prior for an expensive, deterministic simulator. Their idea could be adapted to the stochastic, heteroscedastic case in the following way: first use the two-emulator approach of Henderson *et al.* (2009) to obtain an emulator for $y^C(\cdot)$. Next use the MCMC samples obtained from $\pi(\Theta_C \mid \mathcal{D})$ where Θ_C are the hyperparameters of the GP fitted to $y^C(\cdot)$ to obtain $\pi(\Theta_E)$, the prior for the hyperparameters of $y^E(\cdot)$. Finally, use $\pi(\Theta_E)$ to fit an emulator to $y^E(\cdot)$. We provide no specific details of how to perform such an elicitation, but when constructing $\pi(\Theta_E)$ the analyst should consider how they think the versions of the simulator differ, rather than just setting $\pi(\Theta_E) = \pi(\Theta_C \mid \mathcal{D})$. Cumming & Goldstein (2010) give advice on how to do this within a Bayes linear framework; much of their approach could be applied to a Full Bayesian approach but are particularly relevant

to our EB approach since both methods focus on uncertainty about regression parameters.

Chapter 5

Global Sensitivity Analysis for Stochastic Simulators

5.1 *Introduction*

A simulator used to make statements about a real-world system needs realistic parameter settings. Some parameter values have little uncertainty about them. It is well understood that, close to the earth's surface, acceleration due to gravity takes the value $g = 9.81ms^{-2}$. For complex engineering projects implementing new technologies, the values parameters should take may not be well understood.

In the wind farm setting, consider the lifetime of a gearbox. We may have some understanding of the gearbox lifetime from laboratory testing, such as step accelerated life testing (Nelson, 1980). This testing will have been performed under laboratory conditions which offer an imperfect replication of real-world conditions. Although the test will provide a useful indication of the lifetime of a gearbox, there will be marked differences between the test environment and real-world conditions. Alternatively, we may be using old technologies in new environments, and so there will exist historical data from which quantities of interest can be inferred under a different set of conditions to the planned use case. For example, we may be employing onshore technology in an offshore environment, or pushing offshore technology much further away from the shore into deeper waters. In either case, we may have a good understanding of how the technology works in a limited range of conditions. There are intricacies to every problem which need to be considered; for example, the location of the wind farm

will have an impact on gearbox lifetimes. Components used in an onshore wind farm in a mild climate will likely last considerably longer than those used in a harsh, offshore environment. This problem can be expressed as trying to obtain the conditional probability $P(T < t \mid E)$ where T is the lifetime of a component such as a gearbox. The event E denotes all relevant aspects of the wind farm, such as the wind farm topology and details of its location. Since there will never have previously been a wind farm with properties described by E, the probability $P(T < t \mid E)$ is difficult to obtain with data. A practical solution to this problem is to elicit probability distributions from experts for uncertain quantities (Syed *et al.*, 2020; Wilson & Farrow, 2021; Dalal *et al.*, 2022).

This input uncertainty means that, even for deterministic computer simulations, the output is also uncertain. Let us return to the gearbox example. Suppose one aspect of E is that the wind farm will be in operation for 20 years. If the lifetime distribution was specified, or estimated, to be $T \mid E \sim \mathcal{N}(40, 5^2)$, then it is reasonable to assume that gearbox lifetime will have negligible impact on wind farm performance. Whereas if the lifetime distribution is given by $T \mid E \sim \mathcal{N}(10, 3^2)$, it is likely that many gearboxes will fail during the operation of the wind farm and thus gearbox performance could have serious implications for availability. When there are many uncertain parameters in a black-box simulator, it is not clear which are the main contributors to output uncertainty. In the context of eliciting parameters for a complex simulator this is important. Probability elicitation is a time intensive task; especially when multiple experts are used (Williams *et al.*, 2021). Although elicitation is a time consuming task, it is typically much faster and cheaper than collecting an equivalent sample of data.

If an input is deemed important in the simulator, we should elicit the probability distribution over this parameter carefully. We should consider details such as the shape of the parameter's probability distribution. Less important parameters can be assigned a distribution less rigorously; for example a Normal distribution with an appropriate mean or mode (possibly truncating the distribution to avoid non-physical values) or a uniform distribution with appropriate limits. The least important parameters can be assigned a point mass at a plausible value, or be assigned simple distributions. This essentially follows the SHELF guidelines of spending the most time on the most important parameters; see the "Many Quantities of Interest" document from Oakley & O'Hagan (2019) for an elaboration on this philosophy. An illustration of how an elicitation for forward

uncertainty propagation can be assisted by emulators and sensitivity analysis is given in Figure 5.1. The first emulator will typically be constructed with all relevant unknowns in mind, whereas the emulator constructed in light of refined uncertainty may be a simpler emulator. In particular, it may be appropriate to construct the emulator over only the more important variables and either condition on a plausible value for unimportant parameters or, if unimportant variables are varied in an experiment, they can be accounted for via the nugget term.

The above discussion is well characterised by the following three questions about output uncertainty induced by input uncertainty in a complex simulator:

1. If $\mathbf{x}$ is uncertain; what is the associated uncertainty about $y(\mathbf{x})$?

2. If $\mathbf{x}$ is a vector of uncertain inputs, which elements of $\mathbf{x}$ are contributing the most uncertainty to $y(\mathbf{x})$?

3. If $y(\cdot)$ is a stochastic simulator, how much output uncertainty is coming from the inherent stochasticity of $y(\cdot)$? How much output uncertainty stems from uncertainty about $\mathbf{x}$?

We will now briefly review some approaches to understanding the impact of uncertain inputs to a simulation model.

5.2 *Approaches to understanding input uncertainty*

5.2.1 *Scenario analysis*

A basic approach to considering the impact of input uncertainty is a scenario analysis (Zhang *et al.*, 2012; Grewal & Grewal, 2013). The goal of scenario analysis is to make statements of the form *"If we assume* $\mathbf{x}$ *happens then* $y(\mathbf{x})$ *will happen"*.

A scenario analysis takes a finite collection of inputs $\mathcal{X}_0 = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ and compares their outputs; $y(\mathbf{x}_1)$, $y(\mathbf{x}_2)$, $\ldots$, $y(\mathbf{x}_n)$. The set $\mathcal{X}_0$ might be chosen as the sets of inputs corresponding to a desirable scenario, an undesirable scenario and an intermediate scenario. $\mathcal{X}_0$ may also be chosen as a set of moderate inputs and some extreme inputs. If $y(\cdot)$ is stochastic the analyst would analyse the distribution generated by each $\mathbf{x}_i$. A discussion of scenario analysis is given by Wheatcroft *et al.* (2019); they provide details of choosing $\mathcal{X}_0$.

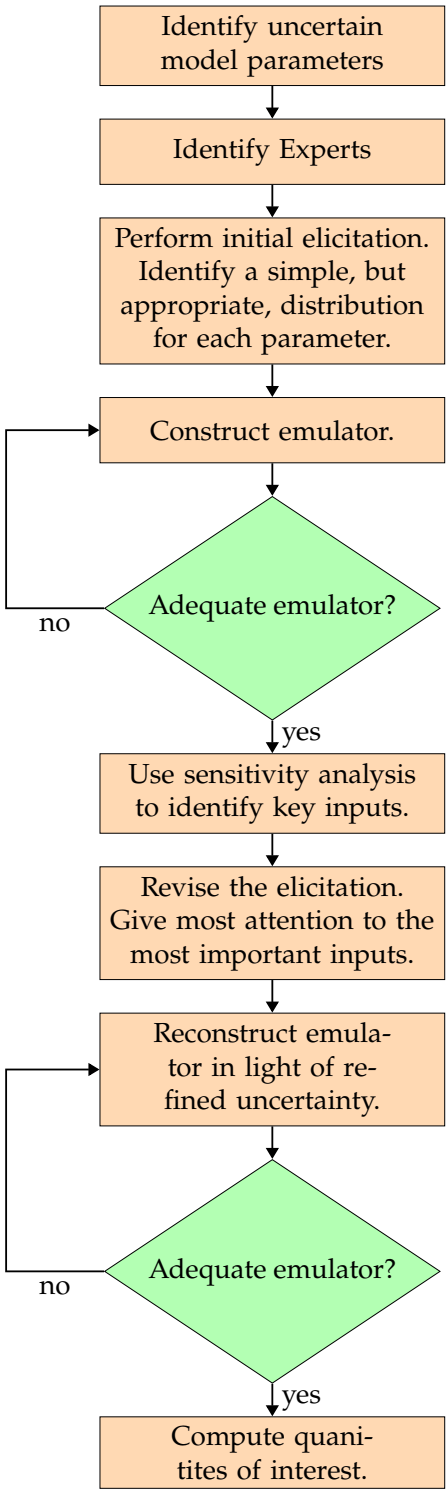

Figure 5.1: Illustration of how emulators and sensitivity analysis can be used to assist the elicitation for forward uncertainty propagation.

One criticism of this approach is that it is not very exploratory. Scenario analysis only considers a small number of values of $\mathbf{x}$ and thus we gain a limited understanding of how $\mathbf{x}$ impacts $y(\mathbf{x})$. Even for input vectors of moderate dimension (say 4), examining the outputs of just a handful of input configurations gives virtually no knowledge of the underlying behaviour of $y(\mathbf{x})$.

Another criticism of this approach is that it does not consider how likely each scenario, and hence each $\mathbf{x}_i$, is; if $y(\mathbf{x}_1)$ suggests that the wind farm is destined to fail but an expert assessed that $\mathbf{x}_1$ has negligible density, we might not worry too much about the implications of this scenario.

Although scenario analysis is not well suited to constructed a detailed understanding of complex simulations, it does serve a role in understanding broad, less well defined questions (Criqui & Mima, 2012; Samsó *et al.*, 2020).

5.2.2 *OAT analysis*

One-at-a-time (OAT) analysis aims to generate understanding of $y(\mathbf{x})$ (or $f(\mathbf{x})$) by varying only the ith inputs, x_i over some pre-specified range whilst keeping the remaining inputs, $\mathbf{x}_{-i}$, fixed at some nominal values. Plotting $f(x_i \mid \mathbf{x}_{-i})$ allows us to study the influence of x_i on $f(\mathbf{x})$ or $y(\mathbf{x})$. This type of analysis is very popular outside of the statistical community (Holvoet *et al.*, 2005; Khalid *et al.*, 2016; Saraiva *et al.*, 2017). OAT analysis is attractive since it is conceptually simple. In varying just one input at a time, we can easily see how x_i impacts the simulator output. There is no noise introduced by varying additional variables (unless $y(\cdot)$ is stochastic). However, a very simple example shows a shortcoming in fixing $\mathbf{x}_{-i}$ at some nominal value. We argue that this is a naive approach to understanding input importance via the following example.

Consider the following simulator

$$f(x_1, x_2) = x_1 x_2. \tag{5.1}$$

Equation (5.1) is overly-simplistic and known, but we will imagine it is a black-box. Suppose a modeller conducts an OAT analysis and they choose to vary x_1 and x_2 over the range $(-1, 1)$. The analyst may then vary x_1 over this range whilst fixing $x_2 = 0$, the midpoint of the range. Since $f(x_1, 0) = 0$ for any x_1 it appears that x_1 has no impact whatsoever on $f(\cdot)$. No simulator of practical interest will be as simple as this, but Equation (5.1) displays an important flaw in OAT analysis: an

OAT analysis *cannot* detect interactions. Another important criticism of OAT is that, asymptotically, it is a local sensitivity analysis (Saltelli & Annoni, 2010). The argument is constructed by considering the volume of the input space explored compared to the total volume of the input space. Suppose that $\mathbf{x} \in [0,1]^m$. If we vary inputs individually, we only explore points within the m dimensional hypersphere of radius $\frac{1}{2}$, rather the full hypercube of input configurations. The relative volume of the hypersphere of radius $\frac{1}{2}$ to the unit hypercube is

$$RV(m) = \frac{\pi^{\frac{m}{2}}}{2^m \Gamma\left(1 + \frac{m}{2}\right)}. \tag{5.2}$$

This expression is fairly complex but, by observing that $\frac{\sqrt{\pi}}{2} < 0.9$ we see that

$$RV(m) < \frac{0.9^m}{\Gamma\left(1 + \frac{m}{2}\right)} \to 0 \text{ as } m \to \infty \tag{5.3}$$

since 0.9^m is a monotonically decreasing function of m, and $\Gamma(1 + \frac{m}{2})$ is monotonically increasing when $m \in [1, \infty)$. The effects of this asymptotic result are felt at small m, thus the idea that OAT analysis is 'global' is a bit of a myth. For a 9 dimensional input configuration, like our Athena example from the previous chapter, $RV(9) = 0.0064$. Thus if we were to perform an OAT analysis, we would be exploring less that 1% of the parameter space. $RV(m)$ is plotted in Figure 5.2. This plot illustrates that, even for problems of small-to-moderate dimension, an OAT analysis is severely under-exploring the input space. This leads to poor inferences about $f(\cdot)$. To consider the full hypercube of input configurations (or a more general shape when inputs are constrained (Kucherenko *et al.*, 2017; Kotidis *et al.*, 2019)) we need to perform integration. One integration based method is probabilistic sensitivity analysis (PSA). This is frequently called variance based sensitivity analysis and we use the terms interchangeably.

5.3 *Probabilistic sensitivity analysis for deterministic simulators*

5.3.1 *Some notation*

Throughout the remainder of this chapter, we will be working with the expectation and variance with respect to

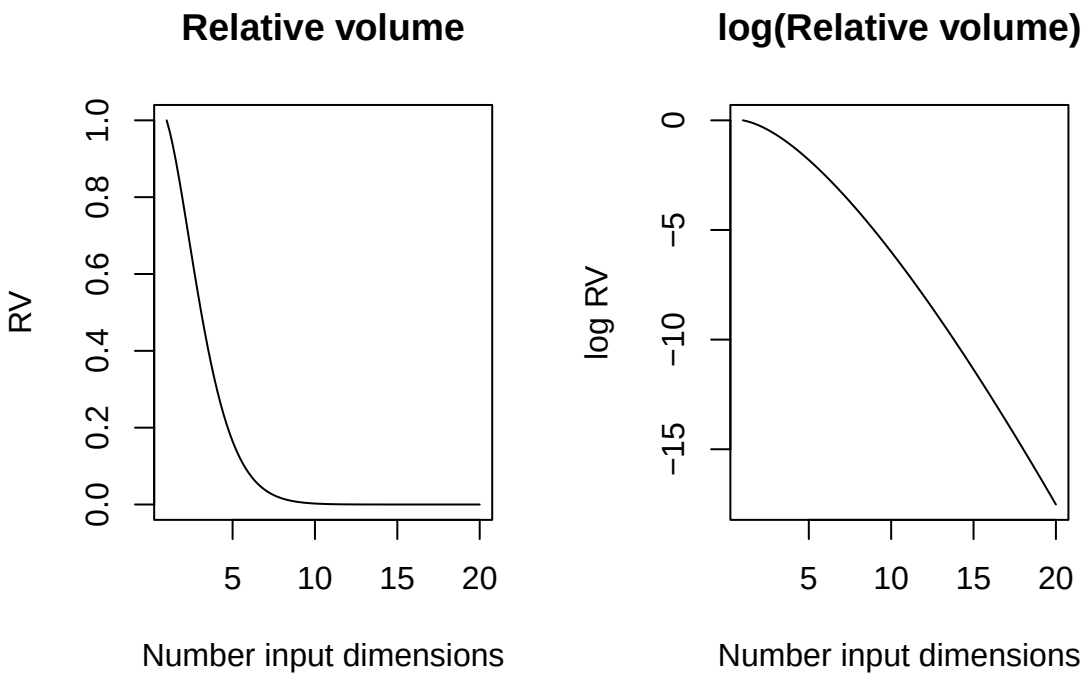

Figure 5.2: Plots showing $RV(m)$ (left) and $\log RV(m)$ (right) for $1 \leq m \leq 20$. Once $m \geq 10$ the relative volume is essentially 0.

(i) $G(\mathbf{x})$ – the distribution characterising uncertainty about simulator inputs, $\mathbf{x}$.

(ii) $\pi(y(\cdot) \mid \mathcal{D})$ and $\pi(f(\cdot) \mid \mathcal{D})$ – the posterior distribution of simulator output for stochastic and deterministic simulators respectively.

(iii) When $y(\cdot)$ is a stochastic simulator, it will have its own expectation, $\mathrm{E}\{y(\cdot)\} = f(\cdot)$ and variance $\mathrm{Var}\{y(\cdot)\} = \lambda^2(\cdot)$.

To avoid confusion between the expectation and variance induced by $G(\mathbf{x})$ and the expectation and variance induced by $\pi(y(\cdot) \mid \mathcal{D})$, we will use $\mathrm{E}_{\mathbf{x}}$ and $\mathrm{Var}_{\mathbf{x}}$ to denote moments with respect to $G(\mathbf{x})$. Moments with a $*$ superscript (E^*, Var^*) denote the moments of posterior distributions of simulator outputs. In other words, with respect to $\pi(y(\cdot) \mid \mathcal{D})$ or $\pi(f(\cdot) \mid \mathcal{D})$. We will retain the m^* and v^* notation used in previous chapters when referring to the posterior mean and variance of $y(\cdot)$. We will use operators without a subscript (E and Var) to denote the "true" moments of $y(\cdot)$ and $\log \lambda(\cdot)$.

This section relies on sampling many collections of inputs, $\mathbf{x} \sim G(\mathbf{x})$. We also condition on elements of $\mathbf{x}$ on many occasions. We will use subscripts, x_i, $\mathbf{x}_J$, to

denote elements and sub-vectors of $\mathbf{x}$. When drawing values of $\mathbf{x}$ from a (joint) distribution, we will use the notation $\mathbf{x}^{(j)}$ to denote the jth sample of $\mathbf{x}$.

5.3.2 *Probabilistic sensitivity analysis*

Within the statistical community, PSA is the go-to method for global sensitivity analysis for two reasons:

1. PSA is a global method.

2. The output of PSA is straightforward to interpret.

One output of PSA is a collection of sensitivity indices (sometimes called Sobol' indices) representing the proportion of uncertainty induced on simulator output by collections of uncertain inputs. The main drawback is that probabilistic sensitivity analysis can be computationally expensive. Thousands of simulator runs are required to obtain reliable estimates of quantities of interest.

When emulators are used in place of a deterministic simulator, $f(\cdot)$, the computational cost is manageable Becker *et al.* (2012); Overstall & Woods (2016). With emulator assisted probabilistic sensitivity analysis, the majority of the computational budget is spent on training an appropriate emulator, which will typically be of the order of a few hundred runs of the simulator. This may be more expensive than a scenario or OAT analysis, but allows for full parameter space exploration and inference about interactions of any order between inputs.

PSA proceeds by specifying a distribution over the inputs $G(\mathbf{x})$ and propagating this distribution, and various conditional distributions based on $G(\mathbf{x})$, through $f(\mathbf{x})$. Sensitivity indices tell us what proportion of the variance in the model output, $V = \text{Var}(Y)$, where $Y = f(\mathbf{x})$ is the uncertain simulator output, is induced by the uncertainty in (subsets of) inputs $\mathbf{x}$. First we consider sensitivity analysis for a deterministic simulator and then extend this to the stochastic setting.

5.3.3 *The functional ANOVA decomposition*

PSA relies on the functional ANOVA decomposition. If $\mathbf{x}$ is a m-dimensional input vector then the functional ANOVA decomposition of a function $f(\mathbf{x})$ is

$$f(\mathbf{x}) = f_0 + \sum_i f_i(x_i) + \sum_{i<j} f_{i,j}(\mathbf{x}_{i,j}) + \sum_{i<j<k} f_{i,j}(\mathbf{x}_{i,j,k}) + \ldots + f_{1,2,\ldots,m}(\mathbf{x}) \qquad (5.4)$$

which is unique when $\mathbf{x}$ has probabilistically independent components. In Equation (5.4), $f_0 = \mathrm{E}_{\mathbf{x}}(Y)$, and

$$f_i(x_i) = \mathrm{E}_{\mathbf{x}_{-i}}(Y \mid x_i) - f_0 \tag{5.5}$$

is the main effect of x_i on $f(\mathbf{x})$. The first order interaction effect of x_i and x_j (where $i < j$) is

$$f_{i,j}(\mathbf{x}_{i,j}) = \mathrm{E}_{\mathbf{x}_{-(i,j)}}(Y \mid x_i, x_j) - f_i(x_i) - f_j(x_j) - f_0. \tag{5.6}$$

This is the effect due to *only* the interaction between x_i and x_j, as their main effects are subtracted. Higher order effects for some non-empty subset J of $\{1, 2, \ldots, m\}$ can be extracted from the functional ANOVA decomposition by finding $\mathrm{E}_{\mathbf{x}_{-J}}(Y \mid \mathbf{x}_J)$ and then subtracting relevant interaction and main effect terms. Plotting $f_i(x_i)$ allows us to understand how x_i influences $f(\mathbf{x})$. Likewise, plotting first order effects allows us to understand how pairs of inputs influence $f(\mathbf{x})$. This is similar to an OAT analysis but allows for interaction terms and averages over $\mathbf{x}_{-i}$, rather than conditioning on a single value.

5.3.4 *Sensitivity indices*

Main effect and interaction plots allow us to gain an intuition of how influential $\mathbf{x}_J$ is on $f(\mathbf{x})$. It is often useful to assign an interpretable value to the relative influence of $\mathbf{x}_i$ on $f(\mathbf{x})$. In PSA, this is achieved by considering the variance of Equation (5.4).

If $G(\mathbf{x}) = \prod_i G_i(x_i)$ then taking the variance of both sides of Equation (5.4) leads to

$$\begin{aligned} \mathrm{Var}\,(Y) &= \sum_{J \subseteq \{1,2,\ldots,m\}} \mathrm{Var}_{\mathbf{x}_J}\{f_J(\mathbf{x}_J)\} && (5.7) \\ &= \sum_{J \subseteq \{1,2,\ldots,m\}} \mathrm{Var}_{\mathbf{x}_J}\{\mathrm{E}_{\mathbf{x}_{-J}}[Y \mid \mathbf{x}_J]\} && (5.8) \end{aligned}$$

The above sum has $2^m - 1$ terms; one for each non-empty subset of $\{1, 2, \ldots, m\}$.

The first type of sensitivity index is

$$S_i = \frac{V_i}{V} \tag{5.9}$$

$$V_i = \text{Var}_{x_i}\{\text{E}_{\mathbf{x}_{-i}}[y(\mathbf{x}) \mid X_i]\}. \tag{5.10}$$

Here, S_i is the proportion of $\text{Var}\{Y\}$ induced by the uncertainty about x_i. This tells us that, if we were to learn the 'true' value of x_i, then we would expect $\text{Var}\{Y\}$ to decrease by $100S_i\%$. This can be generalised to any subset J;

$$S_J = \frac{1}{V}\sum_{J' \subseteq J} V_{J'} \tag{5.11}$$

where S_J is the amount of variability explained by the variables corresponding to J.

The second type of sensitivity index is

$$S_{T_i} = \frac{V - \text{Var}_{x_{-i}}\{\text{E}_{x_i}(Y \mid \mathbf{x}_{-i})\}}{V} \tag{5.12}$$

which represents the proportion of $\text{Var}_{\mathbf{x}}\{Y\}$ remaining when all inputs apart from x_i have been learnt. This can be generalised to S_{T_J}, which represents the remaining uncertainty when all inputs apart from those corresponding to J have been learnt. In this chapter we focus mainly on S_J rather than S_{T_J} since we are interested in using sensitivity analysis for probability elicitation. We need to deduce how to reduce uncertainty about $\mathbf{x}$ in a way which efficiently reduces uncertainty about Y, which is what S_i tells us. The S_i and S_J have a natural decision theoretic justification as measures of importance, which we now discuss.

5.3.5 *Linking PSA to EVPI: decision theoretic justification for* S_i

Let $u(a, \mathbf{x})$ be a utility function for decision $a \in \mathcal{A}$ when the (random) consequence $\mathbf{x}$ occurred. Let $U(a) = \text{E}_{\mathbf{x}}\{u(a, \mathbf{x})\}$ be the expected utility of decision a. Let $U^* = U(a^*)$ where $a^* = \arg\max_{a \in \mathcal{A}} U(a)$ is the optimal decision.

In decision theory, EVPI (expected value of perfect information) is the price the decision maker would pay to obtain the precise value of an uncertain quantity

x_i. The EVPI for input i is given by

$$\text{EVPI}_i = \text{E}_{x_i}\left\{\max_{a\in\mathcal{A}} \text{E}_{\mathbf{x}|x_i} u(a,\mathbf{x})\right\} - U^*. \tag{5.13}$$

EVPI is interpreted as the price a decision maker would pay to have access to perfect information; the true value of x_i. In our context, 'perfect information' is the true value of an uncertain parameter to a simulator. Now suppose our goal is to obtain the true value, or the best possible point estimate, of $f(\mathbf{x})$ when $\mathbf{x}$ itself is uncertain. To define the 'best possible' point estimate, a Bayesian would specify a utility function describing how (un)desirable using a as an estimate of $f(\mathbf{x})$ is. A commonly used utility function in estimation routines is squared loss. Thus we take our utility function to be

$$\begin{aligned} U(a) &= -\text{E}_{\mathbf{x}}\{(a - f(\mathbf{x}))^2\} && (5.14)\\ &= -a^2 + 2a\text{E}_{\mathbf{x}}\{f(\mathbf{x})\} + \text{E}_{\mathbf{x}}\{f(\mathbf{x})^2\} && (5.15)\end{aligned}$$

which is maximised at $a^* = \text{E}_{\mathbf{x}}\{f(\mathbf{x})\}$, the expected simulator output. Thus the EVPI, under this utility function is,

$$\begin{aligned} \text{EVPI}_i &= \text{E}_{x_i}\{-\text{Var}_{X_{-i}}[f(\mathbf{x}) \mid X_i]\} - (-)\text{Var}_{\mathbf{x}}\{f(\mathbf{x})\} && (5.16)\\ &= \text{Var}_{\mathbf{x}}\{f(\mathbf{x})\} - \text{E}_{X_i}\{\text{Var}_{\mathbf{x}_{-i}}[f(\mathbf{x}) \mid x_i]\} && (5.17)\\ &= \text{Var}_{x_i}\{\text{E}_{\mathbf{x}_{-i}}[f(\mathbf{x}) \mid x_i]\} && (5.18)\\ &= V_i. && (5.19)\end{aligned}$$

The sensitivity index $S_i = V_i/V$ improves the interpretability of EVPI; S_i represents the proportion of variation in $f(\mathbf{x})$ explained by x_i. This relationship between S_i and EVPI tells us that, if S_k is the largest sensitivity index and we are able to learn the precise value of only one input, then we should learn x_k. If we can learn two inputs, then we should learn the pair (k,l) which maximise $S_k + S_l + S_{k,l}$. If we want to learn Q inputs, we should learn the set J of size Q which maximises $\sum_{J'\subseteq J} S'_J$.

5.3.6 *Computation of sensitivity indices*

Sobol' (1993) provides Monte Carlo based algorithms for computing sensitivity

indices. Many runs are required to compute the indices, which is not problematic for computationally cheap simulators. Slow run times of complex simulators induce a computational bottleneck in PSA. Replacing the simulator by an emulator provides huge computational savings without much loss of accuracy when the simulator is computationally expensive. In the deterministic case, particular forms of $G(\mathbf{x})$ lead to a tractable emulator-based PSA (Oakley & O'Hagan, 2004). We will present the more general case via Monte Carlo simulation, which does not rely on the form of emulator or specification of $G(\mathbf{x})$. The Monte Carlo case also allows for distributional estimates of S_J, whereas Oakley & O'Hagan (2004) produce only $\mathrm{E}^*\{V_J\}$ and $\mathrm{Var}^*\{V_J\}$.

Constructing main effects plots

To begin the analysis we must estimate f_0. This is achieved via the usual estimate of the mean;

$$f_0 = \int f(\mathbf{x})\, \mathrm{d}G(\mathbf{x}) \tag{5.20}$$

$$\approx \frac{1}{N} \sum_{j=1}^{N} f(\mathbf{x}^{(j)}) \tag{5.21}$$

$$= \hat{f}_0 \tag{5.22}$$

where $\mathbf{x}^{(j)} \overset{\text{iid}}{\sim} G(\mathbf{x})$. Simple i.i.d. sampling returns an unbiased estimate for f_0. If each $x_i \sim U(a, b)$ then Latin Hypercube sampling offers an estimate that is still unbiased but more efficient (McKay *et al.*, 1979; Stein, 1987).

Estimates of main effects are given by

$$f_i(x_i) = \int f(\mathbf{x})\, \mathrm{d}G(\mathbf{x}_{-i} \mid x_i) - f_0 \tag{5.23}$$

$$\approx \frac{1}{N} \sum_{j=1}^{N} f(\mathbf{x}^{(j)} \mid x_i) - \hat{f}_0. \tag{5.24}$$

Now, $f_i(x_i)$ is a function; plots of $f_i(x_i)$ are constructed by evaluating Equation (5.24) over a sequence of values for x_i.

Estimation of the functions implied by interaction effects is as follows:

$$\hat{f}_{i,j}(x_{i,j}) = \frac{1}{N}\sum_{k=1}^{N} f(\mathbf{x}^{(k)} \mid x_{i,j}) - \hat{f}_i(x_i) - \hat{f}_j(x_j) - \hat{f}_0. \tag{5.25}$$

The above estimation procedures assume that we are directly using the simulator. If we are using a GP emulator to facilitate computation then we have posterior distributions for these quantities. The posterior mean of $\hat{f}_i(x_i)$ is obtained by replacing $f(\mathbf{x})$ by $m^*(\mathbf{x})$ in Equation (5.24) and Equation (5.25). The posterior variance can be found by replacing $f(\mathbf{x})$ by $v^*(\mathbf{x})$, and turning subtractions into additions. Monte Carlo estimates can be obtained by drawing random functions from $\pi(f(\cdot) \mid \mathcal{D})$ to produce a probability distribution for $f(x_i)$.

Estimating sensitivity indices

To estimate V, assuming direct use of the simulator, we can use the standard variance estimate;

$$\hat{V} = \frac{1}{N-1}\sum_{j=1}^{N}(f(\mathbf{x}^{(j)}) - f_0)^2. \tag{5.26}$$

To compute V_i we need to find the variance of an expectation. Some manipulation of expectations gives us a more attractive way forward. Starting from the definition of V_i we have

$$\begin{aligned} V_i &= \mathrm{Var}_{x_i}\left\{\mathrm{E}_{\mathbf{x}_{-i}}[f(\mathbf{x}) \mid x_i]\right\} && (5.27)\\ &= \mathrm{E}_{x_i}\left\{\mathrm{E}_{\mathbf{x}_{-i}}[f(\mathbf{x}) \mid x_i]^2\right\} - \mathrm{E}_{x_i}\left\{\mathrm{E}_{\mathbf{x}_{-i}}[f(\mathbf{x}) \mid x_i]\right\}^2 && (5.28)\\ &= \mathrm{E}_{x_i}\left\{\mathrm{E}_{\mathbf{x}_{-i}}[f(\mathbf{x}) \mid x_i]^2\right\} - \mathrm{E}_{\mathbf{x}}\{f(\mathbf{x})\}^2. && (5.29) \end{aligned}$$

Now a Monte Carlo estimate of $E_{\mathbf{x}}\{f(\mathbf{x})\}^2$ is straight forward; $(\hat{f}_0)^2$. This will be slightly biased for $\mathrm{E}_{\mathbf{x}}\{f(\mathbf{x})\}^2$, but with sufficiently large N, the bias will be negligible. The use of an emulator means obtaining sufficiently large N is not a problem. Estimating $T_i = \mathrm{E}_{x_i}\left\{\mathrm{E}_{\mathbf{x}_{-i}}[f(\mathbf{x}) \mid x_i]^2\right\}$ is a bit more of a challenge. An efficient estimation method — under the assumption that $\mathbf{x}$ is uniformly distributed on $[0,1]^m$ is given in Chapter 8 of Gramacy (2020). We illustrate this algorithm in Algorithm 1 under the assumption that $G_i(x_i) = \mathcal{U}(0,1)$.

Bayesian estimation is based on having many posterior draws of $f(\cdot)$. If draws

Algorithm 1 Point estimation of S_i

Require: Estimates $\hat{f}_0 \approx \mathrm{E}_{\mathbf{x}}\{f(\mathbf{x})\}$, $\hat{V} \approx \mathrm{Var}_{\mathbf{x}}\{f(\mathbf{x})\}$.
  for $i = 1, 2, \ldots, m$ **do**
    Construct an $N \times m$ Latin hypercube, X
    $\tilde{X} \leftarrow X$
    for $j = 1, 2, \ldots, N$ **do**
      $\tilde{x}_{i,j} \sim G_i(x_i)$ ▷ Re-draw the ith elements of $\tilde{\mathbf{x}}_j$
    end for
    $\hat{T}_i \leftarrow \frac{1}{N-1} \sum_{j=1}^{N} m^*(\mathbf{x}_j) m^*(\tilde{\mathbf{x}}_j)$
    $\hat{S}_i \leftarrow \frac{\hat{T}_i - \hat{f}_0^2}{\hat{V}}$
  end for
  Return estimated sensitivity indices $\hat{S}_1, \hat{S}_2, \ldots, \hat{S}_m$.

are available, then for each draw we execute Algorithm 1 and return the set of point estimates as a single draw of each value. Many draws are used to approximate the joint posterior distribution of the S_i.

It is straightforward to adapt Algorithm 1 can be adapted to when $\mathbf{x}$ is not uniformly distributed on $[0, 1]^m$ but does have probabilistically independent elements. We replace $G_i(x_i)$ by the distribution of x_i and the ith column of X must be N i.i.d. samples from $G_i(x_i)$, this column must also be independent of the other columns.

The above MC estimates are often relying on one or more previously found MC estimates. The convergence of these 'nested' MC schemes (NMC) has been analysed in the machine learning literature. For example, Rainforth *et al.* (2018) provide optimal convergence rates for NMC schemes. In general, convergence is slower as the number of nested approximations increases. In this work, we only have two levels of nesting (the variance of the mean), and our emulators are sufficiently cheap that achieving negligible bias with a small wall-clock CPU time is achievable.

5.4 *Extending PSA to the stochastic case*

The Athena simulator is stochastic so the illustrated methodology needs to be adapted. Re-applying the above methodology to the emulator mean will allow us to perform inference about $\mathrm{E}\{y(\cdot)\}$. Because the illustrated approach was devised with deterministic simulators in mind it cannot address two important aspects of

sensitivity analysis in stochastic simulation:

1. Quantifying the amount of output uncertainty induced by the stochastic nature of the simulator.

2. Understanding which input variables are driving the stochasticity.

Fortunately, Marrel *et al.* (2012) present an appropriate solution to both the above concerns which places few restrictions on the form of the emulator. The core requirements are a model for the mean and another for the variance (or other measure of dispersion). Therefore, their approach applies to our HetGP and SML emulators constructed in Chapter 4. They compartmentalise the simulator; $y_m(\mathbf{x}) = \mathrm{E}\{y(\mathbf{x})\}$ is the mean component of the simulator and $\lambda^2(\mathbf{x}) = y_d(\mathbf{x}) = \mathrm{Var}\{y(\mathbf{x})\}$ is the dispersion component. We will retain our notation from Chapter 4 using $f(\cdot)$ and $\lambda^2(\cdot)$ to denote the 'true' mean and variance of $y(\cdot)$. The approach of Marrel *et al.* (2012) formulates a stochastic computer model $y(\cdot)$ as a function of $\mathbf{x}$, the simulator inputs, but also x_ε, the state of the pseudo random number generator of the simulator. Although, in theory, we could 'open up' the black box simulation and observe the state of the random number generator, we treat x_ε as an unknown quantity. We refer to $x_\varepsilon \in \mathbb{N}$ as the "seed variable".

5.4.1 *Determining input importance for* $y(\mathbf{x})$

Since $f(\mathbf{x}) = \mathrm{E}\{y(\mathbf{x})\}$ is a deterministic function, the ANOVA decomposition is precisely the same as Equation (5.4). In addition, $y(\mathbf{x})$ has a similar functional ANOVA decomposition which takes x_ε into consideration

$$y(\mathbf{x}) = f(\mathbf{x}) + f_\varepsilon(\mathbf{x}) + \sum_{J \subseteq \{1,2,\ldots,m\}} f_{\varepsilon,J}(\mathbf{x}) \tag{5.30}$$

where $f_\varepsilon(\mathbf{x})$ is the main effect of the seed variable and $f_{\varepsilon,J}(\mathbf{x})$ represents the interaction term between the seed variable and the subset of variables attributed to the set J. As with the deterministic case, V_J/V, determines how much uncertainty of the subset of variables $J \subseteq \{1, 2, \ldots, m\}$ contributes to the total variance in $y(\mathbf{x})$. However, because $y(\mathbf{x})$ is stochastic, the total variance formula tells us that

$$V = \mathrm{Var}_{\mathbf{x}}\{\mathrm{E}[y(\mathbf{x}) \mid \mathbf{x}]\} + \mathrm{E}_{\mathbf{x}}\{\mathrm{Var}[y(\mathbf{x}) \mid \mathbf{x}]\} \tag{5.31}$$

$$= \mathrm{Var}_{\mathbf{x}}\{f(\mathbf{x})\} + \mathrm{E}_{\mathbf{x}}\{\lambda^2(\mathbf{x})\}. \tag{5.32}$$

The definition of V has been adjusted to account for stochasticity, therefore,

$$\sum_{J \subseteq \{1,2,\ldots,m\}} V_J < V. \tag{5.33}$$

The additional variability comes from x_ε. We now have

$$V = \left(\sum_{J \subseteq \{1,2,\ldots,m\}} V_J \right) + V_{T_\varepsilon} \tag{5.34}$$

with $V_{T_\varepsilon} = V \times S_{T_\varepsilon}$ being the total variance induced by x_ε.

5.4.2 *Determining input importance for* $\log \lambda^2(\mathbf{x})$

Under the HetGP and SML assumptions, $\log \lambda^2(\mathbf{x})$ is an arbitrary function. To avoid cumbersome notation, we define $\ell(\mathbf{x}) = \log \lambda^2(\mathbf{x})$. A functional ANOVA decomposition of $\ell(\mathbf{x})$ is also possible. We adopt the convention used in Chapter 4 and place a λ superscript on any quantities related to $\ell(\mathbf{x})$. The functional ANOVA decomposition of $\ell(\mathbf{x})$ is

$$\ell(\mathbf{x}) = f_0^\lambda + f_\varepsilon^\lambda(\mathbf{x}) + \sum_{J \subseteq \{1,2,\ldots,m\}} f_J^\lambda(\mathbf{x}). \tag{5.35}$$

There are no interaction terms between $\mathbf{x}$ and x_ε in Equation (5.35) because λ_V^2 is assumed constant. It would be possible to construct an ANOVA decomposition of the variance of $\ell(\mathbf{x})$, however, this is a fourth order statistic. Emulation of this quantity is discouraged since estimates of fourth order statistics can be highly unstable and it is not clear what the benefits of such an analysis would be (Andrianakis *et al.*, 2017*b*). Sensitivity indices for $\log \lambda^2(\mathbf{x})$ proceed in a similar way to $y(\mathbf{x})$. The total variance of $\ell(\mathbf{x})$ is given by

$$\mathrm{Var}_{\mathbf{x}}\{\ell(\mathbf{x})\} = \lambda_V^2 + \mathrm{Var}_{\mathbf{x}}\{\mathrm{E}\left[\ell(\mathbf{x})\right]\} \tag{5.36}$$

where $\mathrm{E}\left\{\ell(\mathbf{x})\right\}$ is the 'true' value of $\ell(\mathbf{x})$. The S_J^λ have definitions which follow naturally from S_J and and S_{T_ε}. In the log variance context, $S_{T_\varepsilon}^\lambda$ is the amount of uncertainty about $\log \lambda^2(\mathbf{x})$ that is present due to the stochasticity of simulations. We can also compute main effects in an analogous way to $y(\mathbf{x})$.

5.4.3 *Computation of sensitivity indices*

We discuss some details of the computation of the sensitivity indices in the stochastic case as well as understanding the importance of S_{T_ε}. We only provide details of point estimates of the quantities of interest; however our results later give posterior distributions for the sensitivity indices. The posterior distributions are obtained by drawing random functions from the emulator and computing a corresponding point estimate for each draw. Computation of S_{T_ε} is fairly simple. First we need to estimate V. To do this, we use

$$\hat{V} = \widehat{\mathrm{Var}}_{\mathbf{x}}\{f(\mathbf{x})\} + \bar{\lambda}^2(\mathbf{x}) \tag{5.37}$$

where

$$\bar{\lambda}^2(\mathbf{x}) = \frac{1}{N}\sum_{j=1}^{N}\hat{\lambda}^2(\mathbf{x}) \tag{5.38}$$

$$\widehat{\mathrm{Var}}_{\mathbf{x}}\{f(\mathbf{x})\} = \frac{1}{N-1}\sum_{j=1}^{N}(m^*(\mathbf{x}^{(j)}) - \hat{f}_0)^2 \tag{5.39}$$

and then

$$\hat{S}_{T_\varepsilon} = \frac{\hat{V} - \widehat{\mathrm{Var}}_{\mathbf{x}}\{y_m(\mathbf{x})\}}{\hat{V}}. \tag{5.40}$$

Introduction of S_{T_ε} is one of the novel contributions from Marrel *et al.* (2012); it is the total proportion of variability attributed to the stochastic nature of $y(\cdot)$. The physical interpretation of S_{T_ε} is the total sensitivity attributed to all stochastic processes in the simulator. It essentially measures how random a simulator is. Then estimates of S_i follow directly from estimates of T_i, $\mathrm{E}_{\mathbf{x}}\{y(\mathbf{x})\}$ and V and have the usual interpretation. Marrel *et al.* (2012) suggest estimating S_{T_ε} by Equation (5.40) to ensure that $S_{T_\varepsilon} + \sum S_J = 1$. We cannot compute S_ε or any $S_{\varepsilon,J}$; they are non-identifiable because x_ε is not observed. Thus we have a limited understanding of how important the stochasticity in the simulator is or how the seed interacts with elements of $\mathbf{x}$.

5.5 *Application of PSA to the Athena simulator*

We apply the above methodology to the Athena simulator. To facilitate the computation, we use our SML emulator from Chapter 4 which emulated probit $[A(\mathbf{x})]$ for a 9 dimensional, uncertain input vector $\mathbf{x}$. The elements of $\mathbf{x}$ are exactly the same as in the previous chapter. As a reminder, they are the mean time to failure for the 9 sub-assemblies; 1. gearbox, 2. generator, 3. frequency converter (freq. conv.), 4. transformer, 5. main shaft bearing (MSB), 6. blades, 7. tower, 8. foundations, and 9. catch all.

We also revisit the HetGP emulator. We know that the SML emulator improved RMSE and score. The black-box nature of emulators means it is difficult to see *why* one emulator offers a better representation of the simulator. Since main effect plots give an impression of the shape of the simulator we may be able to understand why one emulator has outperformed the other. The results here are also presented in Kennedy *et al.* (2023).

Our choice of $G(\mathbf{x})$ is given by $x_i \overset{\text{iid}}{\sim} \mathcal{U}(0.1, 5)$, which is uniform over the range of inputs our emulators were constructed. This is a useful and practical distribution for PSA when we have limited access to experts (Saisana *et al.*, 2005; Overstall & Woods, 2016). Our estimation approach is Bayesian; we draw 1000 different functions from the posterior distribution and then for each draw compute Sobol' sensitivity indices based on Latin hypercube samples of size $N = 10^4$. Boxplots of first order indices based on both HetGP and SML are given in Figure 5.3. In both cases, $\sum_{i=1}^{9} S_i + S_{T_\varepsilon} \approx 1$ suggesting Athena is approximately additive in the inputs. Some realisations of S_i are negative, this is a symptom of Monte Carlo error and happens when $S_i \approx 0$ (Gramacy, 2020). Note that all posterior means and medians of S_i are positive, and the negative draws are usually identified as outliers, indicated by a $\times$ on the boxplots. We should not worry about small differences amongst the S_i since they are functionals of $G(\mathbf{x})$, which is a rough approximation to an expert's beliefs. We should focus on larger differences. Estimated first order sensitivity indices for the mean give us more-or-less the same interpretation about the Athena simulator. We see in both cases that $S_6 > S_{T_\varepsilon} > S_2$ are the three most important indices, and the rest have mean values comfortably under 5%. Since S_{T_ε} is clearly larger than all but one first order effect, this suggests the stochasticity in the Athena simulator is an important part of the model (inherent stochasticity contributes to slightly more output uncertainty than x_2). The most dramatic

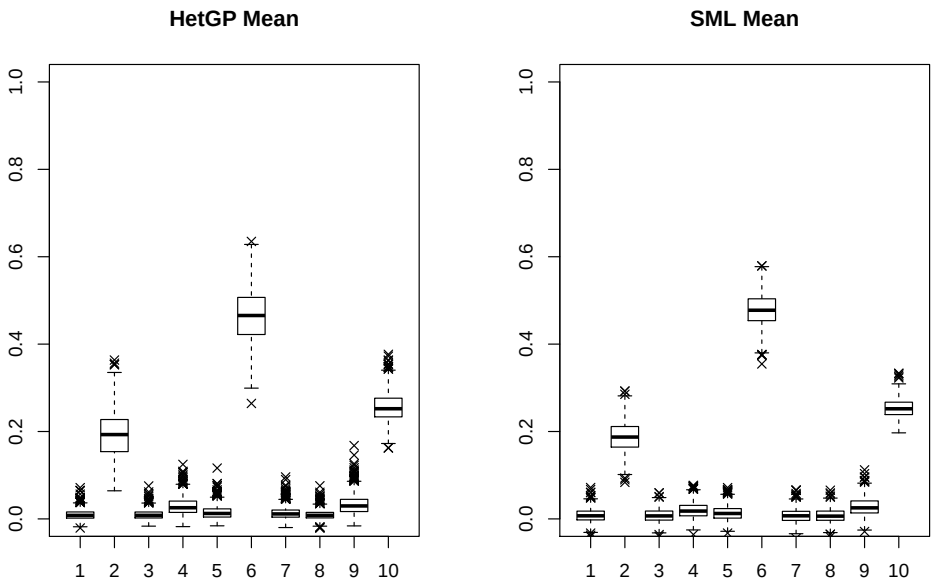

Figure 5.3: Boxplots representing the posterior distribution of S_i; $i = 1, 2, \ldots, 9$. The 10^{th} index corresponds to S_{T_ε}. The left hand plot corresponds to HetGP; the right hand to SML.

difference between the approaches is the estimates of S_i^λ and $f_i^\lambda(x_i)$. Observing Figure 5.4 we can see that the HetGP estimate of S_6^λ is very large (around 50%) whereas under SML the estimate is less than 10%. We suspect that HetGP is interpreting a large proportion of the systematic variation due to x_6 as noise, this is because S_6^λ is estimated to be large via HetGP. SML estimates that S_6^λ is fairly small and provides a slightly larger estimate of S_6 than HetGP.

We see that qualitatively, the plots of main effects (Figure 5.5) agree with the estimated values of S_i and S_i^λ. That is, an input with high S_i exhibits a large range in the main effect plot. The main effect plots for the mean are quite similar with the exception of $f_6(x_6)$. Under SML $f_6(x_6)$ has a much larger range and the shape of $f_6(x_6)$ is different under the two approaches. The HetGP estimate looks very close to $\log(x_6)$; the chosen mean function. This suggests that SML is borrowing information from the cheap simulator to inform the mean response of the expensive simulator and marries up with the (lack of) structure in the residual plots seen earlier (Figure 4.7, Figure 4.8). The main effects for the log variance are quite different under the two emulators. Under HetGP, x_6 is highly influential for the log variance, whereas x_6 is much less influential under the SML emulator.

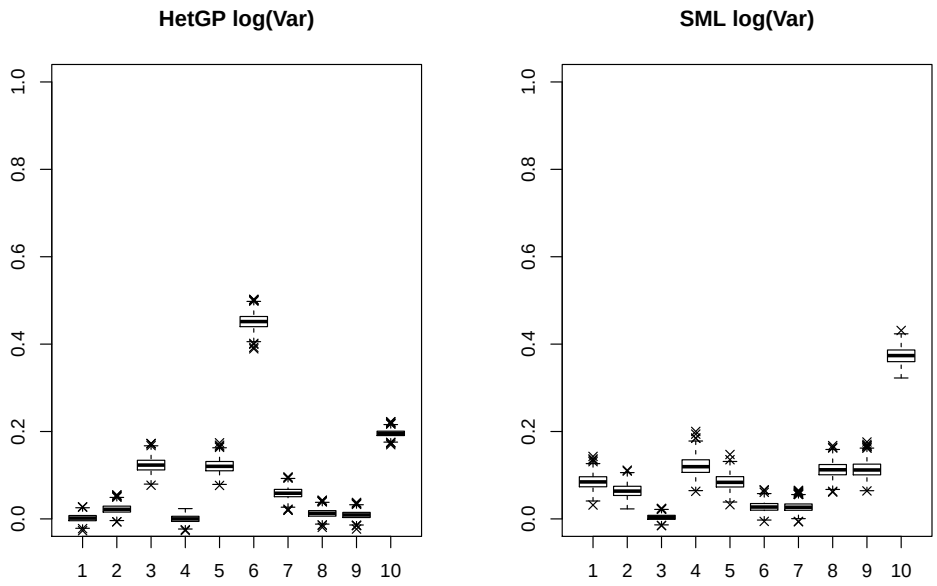

Figure 5.4: Boxplots representing the posterior distribution of S_i^{λ}; $i = 1, 2, \ldots, 9$. The 10^{th} index corresponds to S_{T_ε}. Left hand plot corresponds to HetGP; right hand to SML.

A stark difference is that the slopes of many of the $f_i^{\lambda}(x_i)$ are of opposite sign under the two emulators. For example, the slope of $f_2^{\lambda}(x_2)$ under SML is positive which matches up with empirical estimates given in Figure 5.6. We believe this is due to SML resolving a kind of weak identifiability issue; when insufficient data is fed to a HetGP emulator, systematic variation can be interpreted as noise. Figure 5.6 shows logged sample estimates $\text{Var}\{y(\mathbf{x})\}$. Namely, we have plotted $\log s^2(\mathbf{x})$ where

$$s^2(\mathbf{x}) = \frac{1}{N-1} \sum_{i=1}^{N} \left\{ \text{probit}\,[A(\mathbf{x})]_i - \overline{\text{probit}}\,[A(\mathbf{x})] \right\}^2 \tag{5.41}$$

is the (estimated) variance of the probit availability, and availability is defined at the mean point-wise availability over the first 5 years of the wind farm's life time. The i subscript is used to denote replicates at the same value of $\mathbf{x}$. The data plotted in Figure 5.6 are based on a round of simulations independent of training data of any emulators. We only varied x_2 to construct this plot. Although we were critical of OAT analyses earlier, we can justify fixing the other inputs here as our sensitivity analysis showed the interaction terms were negligible thus varying

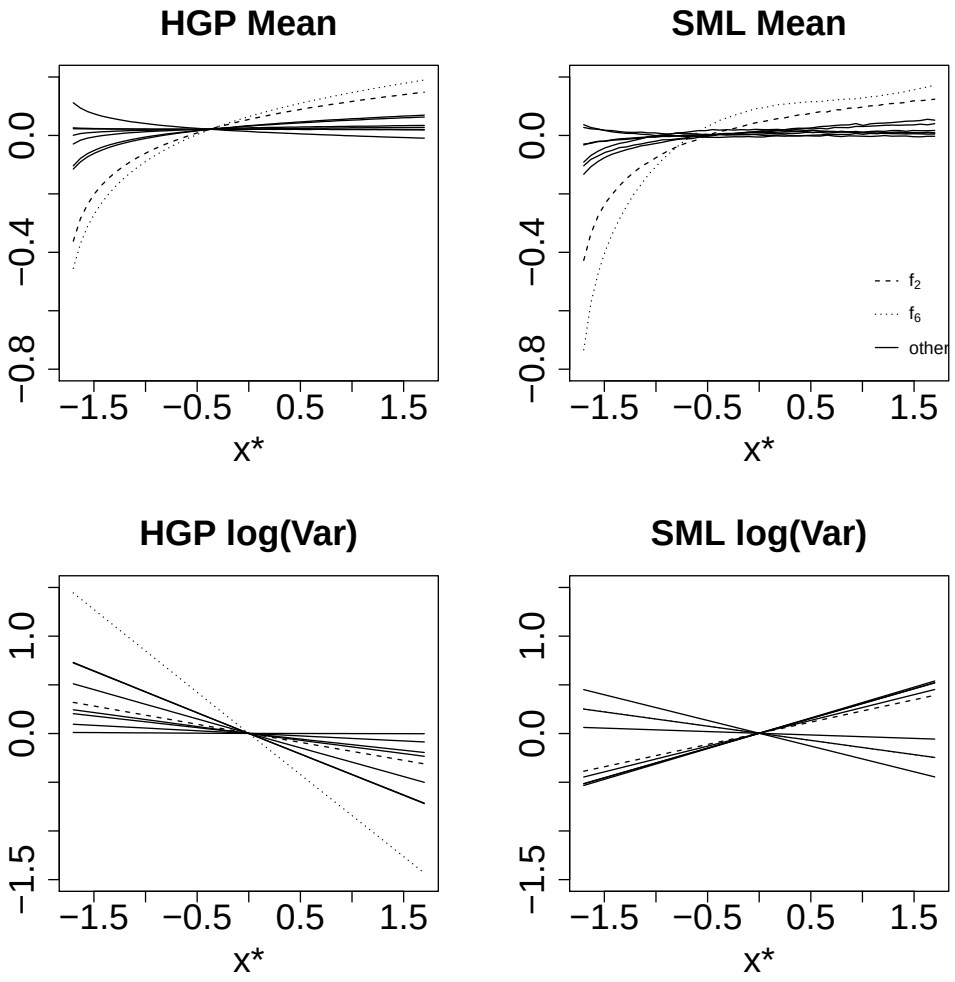

Figure 5.5: Main effect plots under HetGP (left) and SML (right). Top plots correspond to the mean surface and bottom plots to the log variance surface. The dashed lines correspond to $f_2(x_2)$ and $f_2^{\lambda}(x_2)$, dotted lines to $f_6(x_6)$ and $f_6^{\lambda}(x_6)$. The solid lines represent all other main effects. The x axis is on the standardised scale that the emulators were fitted on. Note that the scale of the y axis on the mean plot differs from that of the variance plot.

inputs one-at-a-time is a reasonable way to sense-check the main effect plots.

5.6 *Conclusions*

We have now performed the necessary analyses to set up a future elicitation procedure. Using the results from either the SML or HetGP emulators the largest contributions to output uncertainty were the failures of the blades (x_6) and gen-

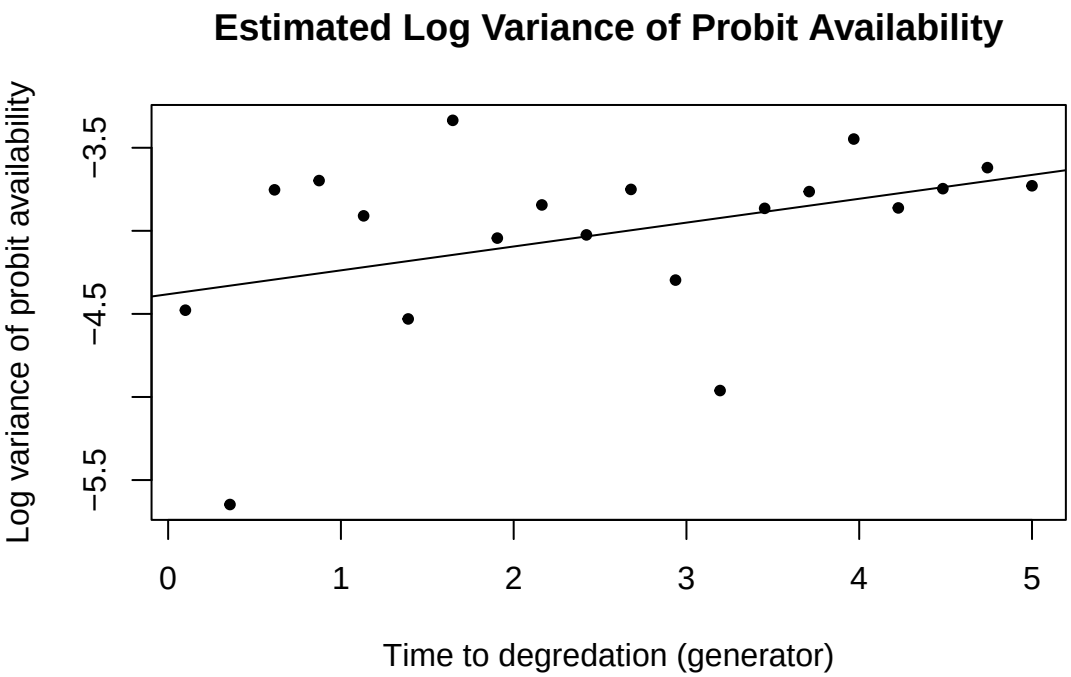

Figure 5.6: Log sample variances of probit$[A(\mathbf{x})]$ at equally spaced points on $[0.1, 5]$. Each estimate is based on 10 replicates of the expensive version of Athena considered in Chapter 4 and generated independently of any emulator training data.

erator (x_2). An equi-tailed 95% credible interval (computed via SML) for $S_2 + S_6$ is $(59, 73)$% of input uncertainty. Note that this estimate does not include $S_{2,6}$, the contribution of uncertainty due to the interaction between x_2 and x_6. The simulator appears to be approximately additive, thus $S_{2,6} \approx 0$. The results of the sensitivity analysis were also used to help understand why one of two competing emulators performed worse than the other. A second round of elicitation of $\mathbf{x}$ would focus mainly on the lifetime parameters of the generator (x_2) and the turbine blades (x_6), since they jointly contribute to over half of output uncertainty. The other inputs would not be completely neglected since they contribute roughly equal amounts of uncertainty to the log variance. Without an emulator this sensitivity analysis would have taken many months of CPU time. Our SML emulator allowed us to further reduce the amount of time required to construct an adequate emulator by exploiting a computationally cheaper version of the Athena simulator.

One issue that we have not explored here is understanding which input(s) have

the largest overall impact on $y(\mathbf{x})$. Suppose that, for an m dimensional simulator, the largest S_i is S_1 and the smallest is S_m. Further suppose that S_1^λ is the *smallest* sensitivity index for the log variance and S_m^λ is the *largest*. In such a scenario it is not clear whether x_1 is more or less important than x_m. In a deterministic setting, if we know the precise value of $\mathbf{x}$ then we can find $f(\mathbf{x})$. In the stochastic case, knowing $\mathbf{x}$ means we can find the distribution of $y(\mathbf{x})$. One way to do this would be to formulate a utility function $U(\mathbf{x}) = \tilde{U}(y(\mathbf{x}))$ and perform an EVPI analysis, on $U(\mathbf{x})$. $U(\mathbf{x})$ should reflect which aspects of $y(\mathbf{x})$ are most critical to our application. For example, if our aim is to find input settings which lead to the average availability, $A(\mathbf{x})$, being above some threshold A^* then $u(\mathbf{x}) = \mathbb{I}(A(\mathbf{x}) > A^*)$ would be an appropriate utility function (remembering that $\mathrm{E}\{u(\mathbf{x})\} = U(\mathbf{x})$). This would be equivalent to emulating, and performing an EVPI analysis on $\Pr(A(\mathbf{x}) > A^*)$.

If the analysis is concerned with understanding which inputs are driving the entire distribution (rather than a single summary), it may be advantageous to borrow ideas from the Bayesian design of experiments literature. In Bayesian design of experiments, one will have a prior specification for the parameters of a statistical model, and the design should aim to maximise a summary of the posterior, such as some notion of difference between prior and posterior (Prangle *et al.*, 2022). A concrete idea would be to construct a HetGP emulator for $y(\cdot)$ then use Kullback-Leibler divergence as a utility function within an EVPI framework. Since, under GP assumptions, $y(\mathbf{x}) \mid \mathbf{x} \sim N(\mu(\mathbf{x}), \lambda^2(\mathbf{x}))$ we may be able to obtain a tractable analysis by assuming $y(\mathbf{x}) \sim \mathcal{N}(M, V)$. In particular, if $\mathbf{x}'$ is a known simulator input, then assuming $y(\mathbf{x})$ and $y(\mathbf{x}')$ are Normally distributed we have

$$U(\mathbf{x}') = \mathrm{KL}(y(\mathbf{x}')||y(\mathbf{x})) = \log\left(\frac{\hat{\lambda}(\mathbf{x}')}{\sqrt{V}}\right) + \frac{\hat{\lambda}^2(\mathbf{x}') + (m^*(\mathbf{x}') - M)^2}{2V} - \frac{1}{2}. \tag{5.42}$$

Then, by an EVPI analysis, the input i with largest EVPI would be the most influential input on the distribution of $y(\cdot)$. Since, with an emulator, m^* and $\lambda^2(\mathbf{x})$ are quick to compute, this analysis should be tractable within a frequentist framework or an Empirical Bayes framework like the emulators described in Chapter 4. A fully Bayesian equivalent to the analysis would have to average over $\pi\{y(\cdot) \mid \mathcal{D}\}$. This is similar to the approach taken by Oakley (2009), who performs an EVPI analysis, assisted by emulators, to deduce which of three medical treatments offers the most viable solution. EVPI analysis comes into play as a way to check the

robustness of the optimal decision. This is perhaps more useful than performing PSA on the mean and log variance separately. Of course, performing PSA on the mean and variance has many valid use cases. It allows us to investigate the behaviour of a complex, black-box stochastic simulator. We used the results of PSA to diagnose potential issues with an emulator We used the fact that the main effect plots of the log variance generated by HetGP were inconsistent with our knowledge of Athena to understand that the HetGP emulator was confusing systematic and random variation. In other words, we extended this idea to *emulator* validation. On a similar vein, Kennedy *et al.* (2006) used the main effect plots to verify that a vegetation model produced sensible outputs. The outputs were deemed unusual which led to the discovery of a bug in the simulator under study.

Chapter 6

Decision Making with Complex Models

In many settings, a model — of any complexity — can be constructed and then deployed to aid decision making. Bayesian decision theory gives us a mature and coherent framework for combining the decision maker's prior beliefs and preferences with a model, including complex stochastic simulators. Recall that if the decision maker represents their preferences for a decision $\mathbf{x} \in \mathcal{X}$ by a utility function $u(\mathbf{x}, \boldsymbol{\theta})$, they represent their beliefs about all uncertain quantities by a prior distribution $\boldsymbol{\theta} \sim \pi(\boldsymbol{\theta})$ and the expected utility of a decision is $U(\mathbf{x}) = \mathrm{E}_{\boldsymbol{\theta}}\{u(\mathbf{x}, \boldsymbol{\theta})\}$, then the optimal decision is defined as

$$\mathbf{x}^* = \arg\max_{\mathbf{x} \in \mathcal{X}} U(\mathbf{x}). \tag{6.1}$$

Prior beliefs, $\pi(\boldsymbol{\theta})$, should be replaced by posterior beliefs, $\pi(\boldsymbol{\theta}|\mathbf{y})$, when relevant data, $\mathbf{y}$, are available.

Once the decision problem has been defined by an appropriate belief structure as above, the decision problem is reduced to an optimisation problem. Although the problem is 'reduced', the enormous literature on mathematical and numerical optimisation should serve as a warning that the reduced problem is usually far from trivial. The difficulty is amplified when the utility function is expensive to compute, is noisy and no gradient information is available. This can be because $U(\mathbf{x})$ itself is expensive to compute or depends on the output of one or more computationally expensive simulators (Williamson & Goldstein, 2012). Both situations are commonly seen in Bayesian design of experiments or subjective Bayesian anal-

yses when viewing a large collection of samples from the posterior distribution with negligible autocorrelation as the output of a computationally expensive simulation (e.g. a long run of an MCMC scheme) (Ryan *et al.*, 2016; Vernon & Gosling, 2022). Our analysis will use relatively simple utility functions; however, the utility function itself depends on output of the Athena model. Therefore, $U(\mathbf{x})$ will be expensive to compute.

We now discuss the various types of optimisation problem that arise and methods to solve them.

6.1 *Discrete problems*

6.1.1 *Small, discrete problems*

Suppose that there is no obvious structure to the decision problem, or the number of decisions that can be taken is small. In a wind farm setting this could be considering where to place the farm from a small number of candidate locations. If we consider n locations $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots \mathbf{x}_n\}$ then the best we can do is just evaluate $U(\mathbf{x}_i)$ for each i.

When the decision is made under uncertainty, the simplest approach is to use a Monte Carlo approximation of $U(\mathbf{x}_i)$. If evaluating $u(\mathbf{x}, \boldsymbol{\theta})$ is expensive because, for example, the utility of a decision depends on the result of a complex computer model, we may wish to construct an emulator for $u(\cdot)$ as a function of uncertain quantities $\boldsymbol{\theta}$ and decision variables $\mathbf{x}$. That is, construct a surrogate, $\hat{u}(\mathbf{x}, \boldsymbol{\theta})$ and propagate beliefs $\pi(\boldsymbol{\theta})$ through $\hat{u}(\mathbf{x}, \boldsymbol{\theta})$. This is precisely what Oakley (2009) does, and achieves a tractable approximation to $U(\mathbf{x}_i)$ via a GP emulator.

Another option for discrete problems is Thompson Sampling (TS). If the number of simulator evaluations we can make is limited, we may want to carefully allocate simulator runs in a sequential manner. TS originated in two-armed bandit problems (Thompson, 1933) and has enjoyed recent success in multi-armed bandit problems since it is simple but effective (Scott, 2010; Chapelle & Li, 2011). The approach to solving a multi-armed bandit problem via TS is to randomly allocate simulator runs one-by-one to a pre-defined set of inputs. This sequential allocation balances exploitation of inputs which are likely to give rise to good solutions whilst acknowledging that simulator outputs are uncertain, thus the uncertainty should be resolved via exploring the decision space.

When the decision set is small and discrete we may be able to phrase the problem as a multi-armed bandit problem. For example, suppose we have $U(\mathbf{x}) = \mathrm{P}(A(\mathbf{x}) > A^*) = \mathrm{E}\{\mathbb{I}(A(\mathbf{x}) > A^*)\}$, and we have a small number of distinct values of $\mathbf{x}$ corresponding to discrete decisions. These decisions, for example, could be a small collection of different locations at which the wind farm could be constructed. Running the Athena simulator at a candidate decision $\mathbf{x}$ will give us an availability which is either above A^* or not. After n simulator runs at different values of $\mathbf{x}$ (with some replication) we are able to obtain probabalistic estimates for each $U(\mathbf{x})$. Since, in this example, $U(\mathbf{x})$ is a probability, a Beta distribution would be an appropriate probabalistic representation of $U(\mathbf{x})$. The Beta prior distribution allows us to estimate $U(\mathbf{x}) = P(A(\mathbf{x}) > A^*)$ via conjugate Bayesian inference. If, prior to running any simulations, $U(\mathbf{x}_i) \sim Beta(a_{i,0}, b_{i,0})$, then once we have evaluated $u(\cdot)$ $n_i > 0$ times at simulator input $\mathbf{x}_i$, the posterior distribution will be $Beta(a_{i,n_i}, b_{i,n_i})$ where

$$a_{i,n_i} = a_{i,0} + \sum_{j=1}^{n_i} \mathbb{I}\left\{A(\mathbf{x}_i)_j > A^*\right\}$$

$$b_{i,n_i} = b_{i,0} + \sum_{j=1}^{n_i} \mathbb{I}\left\{A(\mathbf{x}_i)_j \leq A^*\right\}$$

where $A(\mathbf{x}_i)_j$ is the jth draw of $A(\mathbf{x}_i)$. If $n_i = 0$ then the prior cannot be updated; our prior beliefs will be the representation of our knowledge. TS tells us to next run the simulator at the input which maximises

$$\mathbf{x}_j = \underset{i \in \{1,2,\ldots,n\}}{\arg\max}\ \tilde{U}(\mathbf{x}_i) \tag{6.2}$$

where $\tilde{U}(\mathbf{x}_i)$ are independent, random draws of $U(\mathbf{x}_i) \sim Beta(a_{i,n_i}, b_{i,n_i})$. The random draws induce an exploration-exploitation trade-off. Randomness allows for exploration as it accounts for uncertainty about the $U(\mathbf{x}_i)$. Exploitation comes into play though the posterior distributions. If the posterior density of $U(\mathbf{x}_i)$ is concentrated about large values of $U(\mathbf{x}_i)$, then the probability that the simulator is run at $\mathbf{x}_i$ will increase. We can also allow for exploitation of prior knowledge via the prior; $a_{i,0} \in \mathbb{N}$ and $b_{i,0} \in \mathbb{N}$ can be interpreted as *a priori* counts for the events $A(\mathbf{x}_i) > A^*$ and $A(\mathbf{x}_i) \leq A^*$ respectively. This approach can be extended to continuous problems, which is considered later.

6.1.2 *Structured, discrete problems*

Some decision problems are discrete in nature but far too complex for a brute force computation. The lack of smoothness in such problems makes emulation difficult. In some situations we cannot solve the decision problem exactly because of these computational constraints. We must settle for the best solution possible in the time available.

In complex, discrete problems the expected utility surface can be very rough with respect to the decision space. A well-known example of such a problem is the Travelling Salesman Problem (TSP) in which a salesman aims to visit a fixed set of locations, by traversing the edges of a graph, in such a way that costs are minimised. Phrasing as a decision problem, if $\ell(\mathbf{x})$ is the cost of the path taken by the salesman, we would take $U(\mathbf{x}) \propto \ell(\mathbf{x})$. In this case, a small change in the path might lead to a not so small change in $U(\mathbf{x})$, that is, if $||\mathbf{x} - \mathbf{x}'||$ is small, there is no reason to believe that $||U(\mathbf{x}) - U(\mathbf{x}')||$ is also small (Gutin, 2007).

A mathematically simple approach is a random search. Take a random sample $\mathcal{X}_0 = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ from $\mathcal{X}$. Then an approximation to the optimal decision is $\hat{\mathbf{x}} = \arg\max_{\mathbf{x} \in X_0} U(\mathbf{x})$. If the computational budget is large enough, we would find $\mathbf{x}^*$. In practice, this can be a big "if". Convergence to the optimum requires many evaluations of $U(\mathbf{x})$ when knowledge of $U(\mathbf{x})$ is ignored (Kan & Timmer, 1989). Beyond this, random search serves as a benchmark: if random search does better than a seemingly more intelligent method, then perhaps the method is not as intelligent as first thought.

An adaptation of (purely) random search is stochastic optimisation. A common example is simulated annealing (SA), which is inspired by the process of annealing in metallurgy (Kirkpatrick *et al.*, 1983; Schneider & Kirkpatrick, 2006). SA is one of many stochastic optimisers, and can perform well when the decision surface is very rough. Given a candidate solution $\mathbf{x}'$ and a 'current' solution $\mathbf{x}_t$ we take $\mathbf{x}_{t+1} = X_{t+1}$ where X_{t+1} is a random variable which takes the value $\mathbf{x}'$ with probability $\alpha(\mathbf{x}' \mid \mathbf{x}_t, t)$ and $\mathbf{x}_t$ otherwise. The candidate decision will typically be a modified version of $\mathbf{x}_t$. If the decision space is a subset of $\mathbb{R}^k$, we might form $\mathbf{x}'$ by setting $\mathbf{x}' = \mathbf{x}_t + \mathcal{N}(0, \sigma^2 I_k)$ where σ^2 is a tuning parameter. For discrete problems, we may randomly change individual aspects of $\mathbf{x}$ to construct $\mathbf{x}'$. For example, if $\mathbf{x}$ is a vector of binary values, we may set $x'_i = 1 - x_i$ for some randomly chosen i. Typically $\alpha(\mathbf{x}' \mid \mathbf{x}_t, t)$ should favour $\mathbf{x}'$ for small values of t, corresponding to

exploratory behaviour in early stages, but as t increases, $\alpha(\mathbf{x}' \mid \mathbf{x}_t, t)$ should begin to favour $\arg\max\{U(\mathbf{x}_t), U(\mathbf{x}')\}$, corresponding to increasing levels of exploitation. A common form for alpha is

$$\alpha(\mathbf{x}' \mid \mathbf{x}_t, t) = \min\left\{\exp\left[-\frac{U(\mathbf{x}_t) - U(\mathbf{x}')}{Kt}\right], 1\right\} \tag{6.3}$$

where K is a tuning parameter which controls the rate of the annealing schedule. SA can be thought of as a random walk which is designed to maximise $U(\mathbf{x})$. SA cannot not really exploit any structure in $U(\mathbf{x})$ (and is designed for situations where $U(\mathbf{x})$ has little structure) thus may require many evaluations of $U(\mathbf{x})$ to find a close-to-optimal solution. Within the emulation literature, SA has been successful in providing optimal Latin hypercube designs (Morris & Mitchell, 1995; Pholdee & Bureerat, 2015).

A discrete structure which can be exploited through surrogate modelling is permutations. Consider a set of tasks $\{T_1, T_2, \ldots, T_J\}$. If there are J tasks to complete, then $J!$ permutations need to be considered. For moderately small J, $J!$ can be very large and grows incredibly quickly (3! = 6, 6! = 720, 10! = 3628800). Computing $U(\mathbf{x})$ for all possible $\mathbf{x}$ can quickly become very time consuming, especially if $U(\cdot)$ is expensive. To alleviate this, Wilson *et al.* (2018*c*) use a surrogate inspired by the Benter model (Benter, 1994) to efficiently find an approximate solution to their permutation-based problem.

6.1.3 *Continuous problems*

If the decision space is continuous, e.g. $\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^n$ and we have relatively easy access to $U(\mathbf{x})$, we can use standard optimisation routines like `optim` in `R`. Continuity allows us to embrace calculus when we have easy access to $U(\mathbf{x})$. By 'easy access' we mean that $U(\mathbf{x})$ is available in closed form, or we have access to highly accurate approximations such as Monte Carlo estimates with negligible standard errors.

However, evaluation of U might be sufficiently difficult to make standard optimisation methods cumbersome. When evaluations of $U(\mathbf{x})$ are limited for computational or financial reasons, such as running simulations in a pay-per-usage cloud server, requires collection of primary data, human interaction or is otherwise computationally expensive, we may use a surrogate to reduce optimisation

costs (Tresidder *et al.*, 2012; Astudillo & Frazier, 2020).

Broadly, we can use a surrogate to perform optimisation in two ways. The first way is to construct a one-shot design (via, for example, a Latin Hypercube) to construct an emulator or surrogate $\hat{U}(\mathbf{x})$. Maximising $\hat{U}(\mathbf{x})$ will provide an approximate solution to the decision problem. For examples of this approach see (Wilson *et al.*, 2018*c*; Overstall & Woods, 2017). The second way is to view optimisation as a sequential design problem, which we explore in the next section.

6.2 *Bayesian optimisation*

Instead of directly maximising the surrogate, $\hat{U}(\mathbf{x})$, it may be advantageous to use our knowledge of $U(\mathbf{x})$, expressed as a posterior distribution, to guide us towards values of $\mathbf{x}$ which are likely to improve on the largest value of $U(\mathbf{x})$ seen so far. This is essentially what numerical optimisation does. A standard Newton-Raphson scheme uses the gradient of $U(\mathbf{x})$ to speculate where a larger value of $U(\mathbf{x})$ may lie. Newton-Raphson uses only local knowledge of $U(\mathbf{x})$. Gaussian processes have a global *and* a local understanding of $U(\cdot)$, this can be leveraged.

An approach known as *Bayesian Optimisation* (BayesOpt) uses global and local knowledge, expressed via a posterior distribution, to maximise $U(\mathbf{x})$. The core idea behind BayesOpt is to construct an emulator for $U(\mathbf{x})$ and use this to guide our search towards $\mathbf{x}^*$ (Frazier, 2018). Although much of the BayesOpt language is plagued by machine learning jargon, BayesOpt is actually a form of sequential design. BayesOpt has been found to be useful in a variety of optimisation problems where $U(\mathbf{x})$ has some, or even all, of the following properties

- $U(\mathbf{x})$ is costly to evaluate
- $U(\mathbf{x})$ possess multiple local optima
- derivatives, $\frac{\partial U(\mathbf{x})}{\partial x_i}$, are not available
- $U(\mathbf{x})$ is thought to be quite smooth
- $\mathbf{x}$ is of not too large dimension (less than about 20).

The Athena simulator possesses many of these properties. In particular, we know that it is expensive to evaluate, has no closed form gradient expressions and we think that many quantities output by Athena are reasonably smooth. We do

not know how many local optima Athena has, and that may depend on which input-output combinations are used for optimisation.

BayesOpt is a promising framework for solving decision problems with the Athena simulator. Not all of these properties have to hold for BayesOpt to be a suitable approach. For example, knowledge about maxima or gradient information can be leveraged (Wu *et al.*, 2017; Nguyen & Osborne, 2020). The key properties are that $U(\mathbf{x})$ is costly and smooth; if it were not, other methods could be more appropriate. A common application of BayesOpt is tuning machine learning models (Joy *et al.*, 2016; Snoek *et al.*, 2012). BayesOpt dates back to at least Močkus (1975) who used (Bayesian) linear models as the surrogate. The modern incarnation of BayesOpt has stemmed from Jones *et al.* (1998) who popularised the use of more flexible covariance structures (for example, the squared exponential) within BayesOpt. BayesOpt can be viewed as a sequential design strategy for global maximisation (or minimisation) of expensive black-box functions, see Frazier (2018) for an overview.

We will take a diversion from the Athena simulator and decision making to consider BayesOpt for arbitrary functions, rather than utility functions. Suppose we observe $y(\mathbf{x}) = f(\mathbf{x}) + \varepsilon(\mathbf{x})$ where $f(\cdot)$ is the function to be maximised and $\varepsilon(\mathbf{x})$ is (possibly input dependent) noise, which is assumed to be independent of other error terms, Gaussian and additive. We start by specifying the joint model for $(\mathbf{y}^T, f(\mathbf{x}))$ and use the conditional Normal equations to provide the posterior moments for $f(\mathbf{x}) \mid \mathbf{y}$

$$\begin{pmatrix} \mathbf{y} \\ f(\mathbf{x}) \end{pmatrix} \sim \mathcal{N}\left\{ \begin{pmatrix} \mu(X) \\ \mu(\mathbf{x}) \end{pmatrix}, \begin{pmatrix} \Sigma_{X,X} & \Sigma_{X,\mathbf{x}} \\ \Sigma_{\mathbf{x},X} & \sigma^2 \end{pmatrix} \right\} \tag{6.4}$$

$$f(\mathbf{x}) \mid \mathbf{y} \sim \mathcal{N}(m^*(\mathbf{x}), v^*(\mathbf{x})) \tag{6.5}$$

where X are the locations at which the simulator has been run, $\mu(\cdot)$ is the prior mean function, $\Sigma_{X,X} = \mathrm{Var}\{\mathbf{y}\}$, $\sigma^2 = \mathrm{Var}\{f(\mathbf{x})\}$, $\Sigma_{X,\mathbf{x}} = \Sigma_{\mathbf{x},X}^T = \mathrm{Cov}\{y(X), f(\mathbf{x})\}$ and $m^*(\mathbf{x})$ and $v^*(\mathbf{x})$ are the posterior mean and variance, as found by the multivariate Normal equations; Equation (3.19) and Equation (3.20).

6.2.1 *Acquisition functions*

Once an emulator has been fit to a small collection of initialisation runs, we can start to use the emulator to make intelligent choices about where to next run the simulator so that $f(\mathbf{x})$ can be efficiently optimised. The device that allows us to make decisions about where to next evaluate $f(\mathbf{x})$ is an *acquisition function*. The acquisition function is a function of $\mathbf{x}$ and the posterior distribution on $f(\mathbf{x})$ which aims to find large values of $f(\mathbf{x})$. We introduce new notation for the posterior on $f(\cdot)$ which is better suited to online learning/sequential inference. Suppose we have observed n runs of the simulator: $\mathcal{D}_n = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$. The 'time n' posterior for $f(\cdot)$ (after observing n runs) is denoted $\pi_n(f(\cdot) \mid \mathcal{D}_n, \Theta_n)$ where Θ_n are the time n GP hyperparameters. We will abuse notation here and use $\pi_n(f(\cdot))$ as shorthand for $\pi_n(f(\cdot) \mid \mathcal{D}_n, \Theta_n)$. We then define, using the dummy variable $y = f(\mathbf{x})$,

$$\mathrm{E}_n\{g(f(\mathbf{x}))\} = \int_{-\infty}^{\infty} g(v)\pi_n(y)\,\mathrm{d}v \tag{6.6}$$

as the expected value of $g(f(\mathbf{x}))$ with respect to $\pi_n(f(\mathbf{x}))$, for any function $g(\cdot)$ at a known input $\mathbf{x}$. This additional notation is useful since acquisition functions can often be expressed as expectations of a function of $f(\mathbf{x})$, having observed n simulator runs. This means that many acquisition functions have a decision theoretic justification because they are expressed as expectations of utility functions. Having observed n simulator runs, and assuming that $\mathrm{E}_n(f(\mathbf{x}))$ has a single global optimum, the time n optimal solution is

$$\hat{\mathbf{x}}_n = \arg\max_{\mathbf{x} \in X_n} \mathrm{E}_n\{f(\mathbf{x})\} \tag{6.7}$$

where X_n is the set of input configurations at which the simulator has been run. When $f(\cdot)$ is a deterministic function, $\hat{\mathbf{x}}$ corresponds to the input which, so far, has given the largest function output. This is precisely the value that would be used under any other optimiser. When $f(\cdot)$ is observed with noise, Equation (6.7) is the value of $\mathbf{x}$ leading to the largest posterior mean. This follows naturally from Bayesian decision theory. We now introduce some more notation to allow a fluent discussion of common acquisition functions. Let $y_n^* = \mathrm{E}_n\{f(\hat{\mathbf{x}}_n)\}$, let $\mu_n(\mathbf{x}) = \mathrm{E}_n\{f(\mathbf{x})\}$ and let $\sigma_n^2(\mathbf{x}) = \mathrm{Var}_n\{f(\mathbf{x})\} = \mathrm{E}_n\left\{[f(\mathbf{x}) - \mu_n(\mathbf{x})]^2\right\}$, where $\mathbf{x} \in \mathcal{X}$ is any valid input.

Probability of improvement

The first acquisition function we encounter is the probability of improvement (PI) denoted α_{PI}. Consider the following utility function over possible values of $f(\mathbf{x})$:

$$\tilde{\alpha}_{PI}(\mathbf{x}) = \begin{cases} 1, & f(\mathbf{x}) > y_n^* \\ 0, & f(\mathbf{x}) \leq y_n^* \end{cases} \tag{6.8}$$

that is, an indicator utility function returning 1 when we improve upon y_n^* and returning 0 when there is no improvement. This expresses the belief that all improvements, regardless of size, are equally preferable. We then have $\alpha_{PI}(\mathbf{x}) = \mathrm{E}_n\{\tilde{\alpha}_{PI}(\mathbf{x})\}$. The expectation of an indicator function is the probability of the event in question, thus we have

$$\alpha_{PI}(\mathbf{x}) = \mathrm{P}\left(f(\mathbf{x}) > y_n^*\right) \tag{6.9}$$

$$= \Phi\left(\frac{\mu_n(\mathbf{x}) - y_n^*}{\sigma_n(\mathbf{x})}\right). \tag{6.10}$$

Regardless of the chosen statistical model, Equation (6.9) provides the probability of improvement; this will be available in closed form for a large class of statistical models. If the closed form is not available, a highly accurate approximation is likely to be built into popular scientific computing libraries. Under GP assumptions, the probability of improvement can be expressed as Equation (6.10) which, technically, is not tractable but good approximations are widely available, for example, `pnorm()` from `R`.

Expected improvement

A criticism of PI as an acquisition function is that PI sees all possible improvements as equally good, regardless of the magnitude of the improvement. Suppose that $y_n^* = 0$ and further suppose $f(\mathbf{x}_1) \sim N(0, 1)$ and $f(\mathbf{x}_2) \sim N(0, 10)$. We see that $\alpha_{PI}(\mathbf{x}_1) = \alpha_{PI}(\mathbf{x}_2) = 0.5$, therefore, this BayesOpt scheme views $\mathbf{x}_1$ and $\mathbf{x}_2$ as equally promising. However, the potential gains under $\mathbf{x}_2$ are much larger than the potential gains under $\mathbf{x}_1$. The nature of PI can be investigated via partial derivatives. We see that $\frac{\partial \alpha_{PI}(\mathbf{x})}{\partial \mu_n(\mathbf{x})} = \frac{1}{\sigma_n(\mathbf{x})}\phi\left(\frac{\mu_n(\mathbf{x}) - y_n^*}{\sigma_n(\mathbf{x})}\right) > 0$ thus, increasing $\mu_n(\mathbf{x})$ increases $\alpha_{PI}(\mathbf{x})$. Therefore, inputs corresponding to function values with large

predictive means are favoured. However, $\frac{\partial \alpha_{PI}(\mathbf{x})}{\partial \sigma_n(\mathbf{x})} = -\frac{\mu_n(\mathbf{x})-y_n^*}{\sigma_n(\mathbf{x})^2}\phi\left(\frac{\mu_n(\mathbf{x})-y_n^*}{\sigma_n(\mathbf{x})}\right)$. The sign of this expression depends on $\mu_n(\mathbf{x}) - y_n^*$. In particular, sign $\left\{\frac{\partial \alpha_{PI}(\mathbf{x})}{\partial \sigma_n(\mathbf{x})}\right\}$ = sign $\{y_n^* - \mu_n(\mathbf{x})\}$, therefore increasing $\sigma_n(\mathbf{x})$ increases $\alpha_{PI}(\mathbf{x})$ whenever $\mu_n(\mathbf{x}) < y_n^*$ but decreases $\alpha_{PI}(\mathbf{x})$ when $\mu_n(\mathbf{x}) > y^*(\mathbf{x})$ and $\alpha_{PI}(\mathbf{x}) = 0.5$ when $\mu_n(\mathbf{x}) = y^*(\mathbf{x})$. This means that we tend not to explore areas in which $f(\mathbf{x})$ is uncertain but has moderately large values of $\mu_n(\mathbf{x})$. The way PI deals with uncertainty is unsatisfactory; if $f(\mathbf{x})$ is thought to be large but Var $\{f(\mathbf{x})\}$ is also large, this is a region of the input space that should be explored further as the potential gains are large. This criticism can be boiled down to the fact that PI does not consider how large the potential change in y_n^* is.

This motivates *Expected Improvement* (EI) which gives a reward proportional to the improvement. First we define the improvement by

$$\tilde{\alpha}_{EI}(\mathbf{x}) = \max\{0, f(\mathbf{x}) - y_n^*\}. \tag{6.11}$$

This is intuitive: if $f(\mathbf{x}) \leq y_n^*$ our best value has not improved. But if $f(\mathbf{x}) > y_n^*$ then we have improved our current best value of $f(\cdot)$ by $f(\mathbf{x}) - y_n^*$. Now since $f(\mathbf{x})$ is unknown, we integrate out $f(\mathbf{x})$, which has density $\pi_n(f(\mathbf{x}))$, and thus the EI is

$$\alpha_{EI}(\mathbf{x}) = \mathrm{E}_n\{\tilde{\alpha}_{EI}(\mathbf{x})\}. \tag{6.12}$$

Under standard GP assumptions, we have an analytic expression for EI in terms of the Normal CDF and PDF. We aim to maximise $f(\cdot)$, so the acquisition function can be expressed as (Pourmohamad & Lee, 2021):

$$\alpha_{EI}(\mathbf{x}) = \begin{cases} \sigma_n(\mathbf{x})\phi\left(\frac{\mu_n(\mathbf{x})-y_n^*}{\sigma_n(\mathbf{x})}\right) + (\mu_n(\mathbf{x}) - y_n^*)\Phi\left(\frac{\mu_n(\mathbf{x})-y_n^*}{\sigma_n(\mathbf{x})}\right), & \sigma_n(\mathbf{x}) > 0 \\ 0, & \sigma_n(\mathbf{x}) = 0 \end{cases}. \tag{6.13}$$

The EI is 0 when $\sigma_n(\mathbf{x}) = 0$ because $\sigma_n(\mathbf{x}) = 0$ if, and only if, we know the value of $f(\mathbf{x})$ precisely. $f(\mathbf{x})$ will either be y_n^* or be smaller than y_n^* and hence offers no improvement. Although EI was constructed with deterministic problems in mind, it generalises naturally to the stochastic case. The exploitation-exploration trade-off under EI is fairly transparent. The first term of EI is multiplied by $\sigma_n(\mathbf{x})$, thus large uncertainty leads to larger values of α_{EI}. The second term is multiplied by $\mu_n(\mathbf{x}) - y_n^*$, thus large mean predictions typically increase EI.

Applying basic calculus rules to Equation (6.13) leads to $\frac{\partial \alpha_{EI}(\mathbf{x})}{\partial \mu_n(\mathbf{x})} = \Phi\left(\frac{\mu_n(\mathbf{x}) - y_n^*}{\sigma_n(\mathbf{x})}\right) > 0$ and $\frac{\partial \alpha_{EI}(\mathbf{x})}{\partial \sigma_n(\mathbf{x})} = \phi\left(\frac{\mu_n(\mathbf{x}) - y_n^*}{\sigma_n(\mathbf{x})}\right) > 0$, thus EI increases with both $\mu_n(\mathbf{x})$ and $\sigma_n(\mathbf{x})$. That is, for fixed $\mu_n(\mathbf{x})$, a more uncertain value is a preferred input to run the simulator at, whilst if $\sigma_n(\mathbf{x})$ is fixed we will run the simulator at the input with greatest predicted value.

There are many extensions to EI which take into account problem structure. Examples include optimisation under unknown (black-box) constraints which learns the optimum input/output and the constraints (Gramacy & Lee, 2011). Batch (or "multi-point") EI runs the simulator at multiple, promising solutions simultaneously to take advantage of abundant cores in a parallel computing setting (Marmin *et al.*, 2015; Diessner *et al.*, 2022).

Upper credible bound

Another common acquisition function is the upper credible bound, or upper confidence bound, depending on the chosen interpretation of probability (Srinivas *et al.*, 2009). Both can be abbreviated to UCB. The UCB acquisition function is, under a GP model for $f(\cdot)$,

$$\alpha_{UCB}(\mathbf{x} \mid \nu_n) = \mu_n(\mathbf{x}) + \nu_n \sigma_n(\mathbf{x}). \tag{6.14}$$

where $\nu_n > 0$ are a sequence of tuning parameters controlling the exploration-exploitation trade-off. UCB was proposed with maximisation in mind but if minimisation is required we can use the lower credible/confidence bound instead.

$$\alpha_{LCB}(\mathbf{x} \mid \nu_n) = \mu_n(\mathbf{x}) - \nu_n \sigma_n(\mathbf{x}). \tag{6.15}$$

For many statistical models, Equation (6.14) is an upper credible/confidence bound. We can adjust ν_n as appropriate. This means that in principle, UCB gives a tractable acquisition function for $f(\mathbf{x})$, whenever the chosen emulator has a tractable mean and variance. If $f(\mathbf{x})$ is not modelled *a priori* by a GP, the expected improvement would likely be intractable. If desired, we can take $\nu_n = \nu$ for all n. Now since UCB is a linear combination of the posterior mean and standard deviation, the exploration-exploitation trade-off is transparent. Increasing $\mu_n(\mathbf{x})$ or $\sigma_n(\mathbf{x})$ whilst the other remains fixed clearly increases $\alpha_{UCB}(\mathbf{x} \mid \nu_n)$. The trade off is controlled by ν_n. Setting $\nu_n = 0$ leads to a pure exploitation regime: max-

imise the posterior mean. Letting $\nu_n \to \infty$ leads to picking the point with highest uncertainty, which is essentially ALM (Equation (3.28)). Choosing a moderate value of ν_n leads to an exploration-exploitation compromise. The UCB acquisition function is sometimes called an 'optimistic' approach; we are choosing $\mathbf{x}_{n+1}$ based on an upper estimate for $f(\mathbf{x}_{n+1})$.

Although not immediately obvious, we can actually write $\alpha_{UCB}(\mathbf{x})$ as the expected value of a random variable. Wilson *et al.* (2018*b*) show that

$$\alpha_{UCB}(\mathbf{x} \mid \nu_n) = \int_{-\infty}^{\infty} \frac{(\mu_n(\mathbf{x}) + |f(\mathbf{x}) - \mu_n(\mathbf{x})|)}{\sqrt{2\pi\tilde{\sigma}^2}} \exp\left\{-\frac{(f(\mathbf{x}) - \mu_n(\mathbf{x}))^2}{2\tilde{\sigma}^2}\right\} \mathrm{d}f(\mathbf{x}) \quad (6.16)$$

where $\tilde{\sigma} = \sqrt{2\pi\nu_n}\sigma$. This means that the UCB acquisition can be written as the following expectation

$$\alpha_{UCB}(\mathbf{x} \mid \nu_n) = \mathrm{E}_{\tilde{y}}\{\tilde{y}\} \quad (6.17)$$

where $\tilde{y} \sim tr\mathcal{N}(\mu(\mathbf{x}), \tilde{\sigma}^2(\mathbf{x}); \mu(\mathbf{x}), \infty)$ where $tr\mathcal{N}(m, s^2; a, b)$ denotes a Normal distribution truncated below at a, above at b and the distribution to be truncated is $\mathcal{N}(m, s^2)$. Therefore $\alpha_{UCB}(\mathbf{x})$ can be interpreted, in a decision theoretic context, as the expected value of a linearly transformed and truncated version of $f(\mathbf{x})$. This is quite difficult to interpret but this representation of UCB was constructed with computation, not interpretation, in mind (Wilson *et al.*, 2017).

Thompson sampling

There is a stochastic approach to BayesOpt; Thompson sampling (TS). By stochastic, we mean that the acquisition function is randomly generated. Whether the simulator is stochastic or deterministic is not critical in this context. The idea behind TS is to draw a random function $g(\cdot) \sim \pi_n(f(\cdot))$ and then use $g(\cdot)$ as an acquisition function. The next point at which to run $f(\mathbf{x})$ is then given by the maximiser of this drawn function:

$$\mathbf{x}_{n+1} = \arg\max_{\mathbf{x} \in \mathcal{X}} g(\mathbf{x}). \quad (6.18)$$

We discussed earlier that TS is useful in multi-armed bandit problems. With a GP framework, for problems defined over a continuous domain, a full draw from $\pi_n(f(\cdot))$ is impossible; this in an object of infinite dimension. The best we can do is draw $g(\tilde{X})$ from $\pi_n(f(\tilde{X}))$ where $\tilde{X}$ is a large, but finite, collection of

inputs. This can cause problems in high dimensional settings due to the curse of dimensionality; $\tilde{X}$ will almost always be sparse within the input space. This can be combated by 'extending' the realised function by adding rows to the lower Cholesky decomposition of $\text{Var}_n\{f(\tilde{X})\}$. Algorithms for extending the Cholesky decomposition are discussed by Higham (2009). This means that maximising $g(\cdot)$ when the decision space is either continuous or large and discrete is difficult. However, the three earlier acquisition functions all need to be maximised via some inner optimisation routine like `optim`. The TS acquisition is trivially maximised via a lookup table.

The discussed acquisition functions are plotted in Figure 6.2. They all correspond to the function

$$f(x) = 0.6\sin(2\pi x) + 2\cos(3\pi x) + \sin(4\pi x) + 2x \tag{6.19}$$

and corresponding emulator in Figure 6.1. In this particular example, PI, EI and UCB are all maximised in roughly the same place (in the region of 0.6–0.8), which happens to be close to the true maximiser. One TS acquisition function (red) is maximised quite far away from the true maximiser ($\hat{x} \approx 0.5$). This is because the emulator is exhibiting large uncertainty across much of the input space. The other two TS acquisition functions (black and green) and maximised close to the true maximiser. Remember that the acquisition functions are suggesting suitable locations to run the simulator rather than a prediction of where the maximum is.

6.2.2 *Online GP updates for efficient BayesOpt*

BayesOpt is a sequential design problem. The emulator is updated as simulator runs arrive. GP prediction depends on the inversion of an $n \times n$ matrix. Matrix inversion is well known to be a slow (usually $O(n^3)$) and numerically unstable operation.

By constructing a partition of Σ_{n+1}, the time $n+1$ precision matrix Ω_{n+1} can be expressed in terms of $\Omega_n = \Sigma_n^{-1}$ and the partitioned Σ_{n+1}. This partitioned representation is numerically stable and fast to compute. Consider the following partition of Σ_{n+1}

$$\Sigma_{n+1} = \begin{pmatrix} \Sigma_n + \lambda^2(X_n)I_n & c(\mathbf{x}_{n+1}, X_n) \\ c(\mathbf{x}_{n+1}, X_n)^T & \sigma^2(\mathbf{x}_{n+1})) \end{pmatrix} \tag{6.20}$$

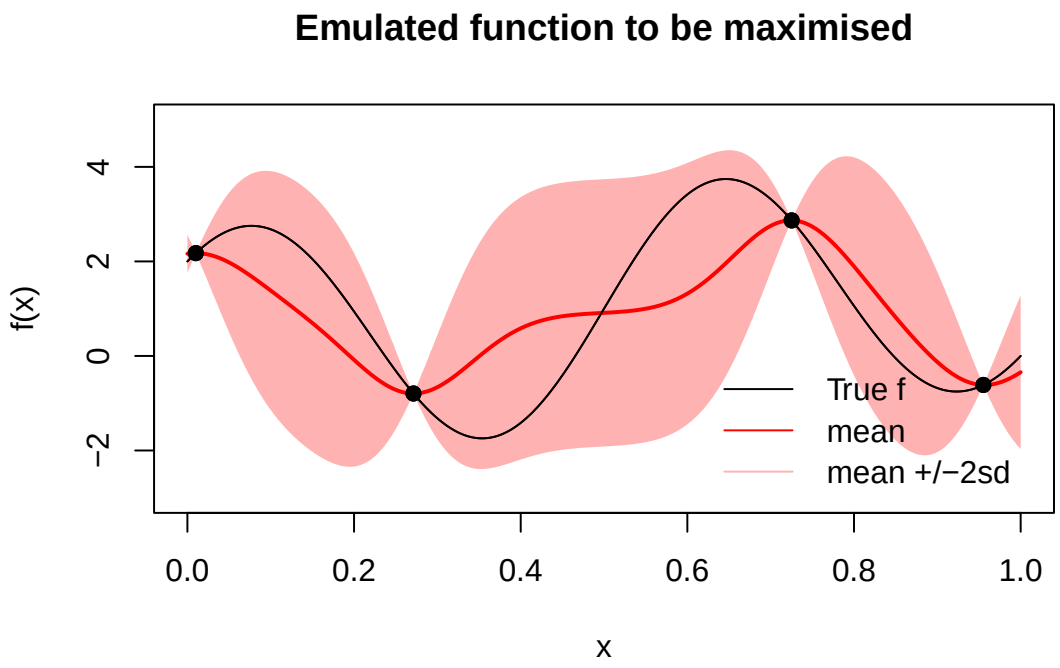

Figure 6.1: A function, f(x), given by Equation (6.19), to be optimised via BayesOpt with a corresponding emulator. The unknown function to be optimised is deterministic and given by the black line. The emulator mean is given by the solid red line and the translucent red region represents a 95% probability band for $f(x)$. Plots of acquisition functions corresponding to this emulator are in Figure 6.2.

where $c(\mathbf{x}_{n+1}, X_n)$ is the column vector of covariances between $\mathbf{x}_{n+1}$ and the rows of X_n and $\sigma^2(\mathbf{x}_{n+1}) = \text{Var}\{y(\mathbf{x}_{n+1})\}$. We then have

$$\Omega_{n+1} = \begin{pmatrix} \Omega_n + kk^T/s & -k/s \\ -k^T/s & 1/s \end{pmatrix} \tag{6.21}$$

where $k = \Omega_n c(\mathbf{x}_{n+1}, X_n)$ is a length n column vector and $s = \sigma^2(\mathbf{x}_{n+1}) - c(\mathbf{x}_{n+1}, X_n)^T k$ is a scalar quantity. This result can be confirmed by checking $\Sigma_{n+1}\Omega_{n+1} = \Omega_{n+1}\Sigma_{n+1} = I_{n+1}$; see Section 3.4 of Gentle (2007) for the result and the Exercises of Section 3 for a sketch proof. The only new information we need is $c(\mathbf{x}_{n+1}, X)$ and $\lambda^2(\mathbf{x}_{n+1})$, and if $\lambda^2(\mathbf{x})$ is assumed constant then it is already known. This amounts to calculating a length n vector of covariances. Efficient updating via Equation (6.21) works only if we condition on GP hyperparameters. This can be implemented if GP hyperparameters are only updated occasionally, or the results

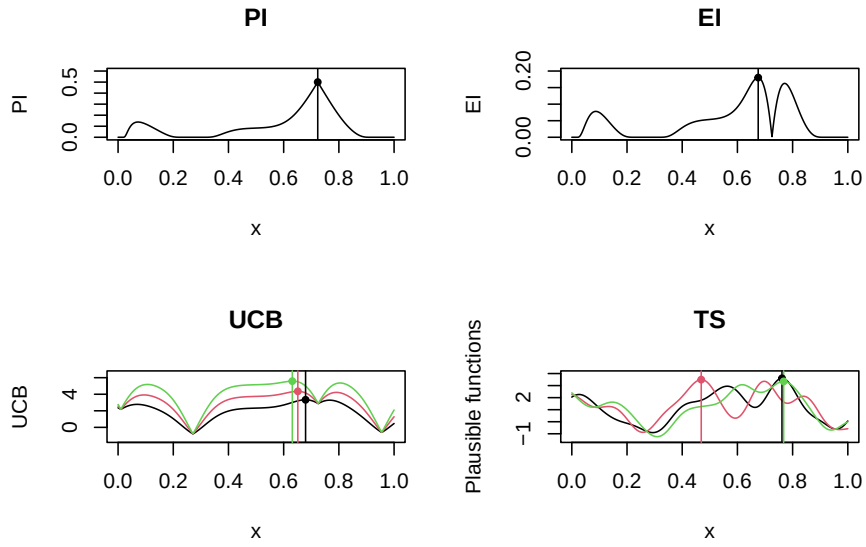

Figure 6.2: Plots of acquisition functions for the function and emulator in Figure 6.1. Vertical lines correspond to the maximiser of the acquisition function with the bullet being the the coordinates of the maximum. Three possible acquisition functions for UCB are shown corresponding to $\nu = 1, 2, 3$ and three random draws of $f(\cdot)$ are shown for TS.

of a previous computer experiment may be used to estimate Θ. We may also use the initialisation runs of BayesOpt to estimate Θ. Sequential Monte Carlo can be used to perform efficient updates of a fully Bayesian posterior, under some appropriate assumptions (Gramacy & Polson, 2011).

6.3 *Decision making or decision support?*

We return to thinking about utility functions and the role of BayesOpt in making decisions. Once a BayesOpt scheme has been run, we have an emulator for the expected utility function, $U(\cdot) \mid \mathcal{D} \sim \mathcal{GP}(\cdot, \cdot)$ and an *approximate* maximiser $\hat{\mathbf{x}} = \arg\max_{\mathbf{x} \in X_n} \mu_n(\mathbf{x})$. In almost all applications of BayesOpt, it is common to stop here and report only $\hat{\mathbf{x}}$. This is perhaps a symptom of the typical use case. A common use of BayesOpt is tuning hyperparameters of machine learning models where parameter uncertainty (decision uncertainty) is usually not regarded as important, or is difficult to incorporate (Bergstra *et al.*, 2011; Swersky *et al.*, 2013; Kim *et al.*, 2020).

When making decisions for complex problems, one must possess great opti-

mism to believe that the single, optimal decision can be uncovered when several necessary approximations to the truth have been forced upon us. It is myopic to report only the 'best' decision. Many simplifications and approximations will have been used. Simulators are flawed representations of physical systems. The simulators will be a misrepresentation of the mathematical methods, for example, a numerical differential equation solver will be used. This will return a solution which is incorrect but close to the 'true' answer implied by the underlying mathematics. Utility functions and prior beliefs will be misspecified to some degree and emulators provide a further, but necessary, approximation of the expected utility function. Therefore, $\hat{\mathbf{x}}$ should not be taken as the ground truth; we should investigate other values of $\mathbf{x}$ which lead to values of $U(\mathbf{x})$ which are 'close' to $U(\hat{\mathbf{x}})$. There is also an aspect of responsibility to consider. An analyst, who is junior to the decision maker, will typically be the individual tasked with running the BayesOpt scheme and reporting results, such as $\hat{\mathbf{x}}$, to the decision maker, who has higher responsibility. If only $\hat{\mathbf{x}}$ is reported, then one might ask what is the point of the decision maker? Another argument which favours decision support is the uncertainty about the utility function used. A comprehensive list of uncertainties in decision making is given by Dent *et al.* (2020); some key aspects are:

1. Uncertainty associated to the simulator. Choice of input variables; choice of model fidelity; model discrepancy.

2. Uncertainty about the simulator. When the simulator is expensive and/or stochastic, we rarely know the true output value or output distribution.

3. Uncertainty about the utility function. The utility function is elicited and thus is prone to error induced by the elicitation process (e.g. biases or 'random error' exhibited by the decision maker). This also includes the elicitation of appropriate decision attributes, as well as the parameters and functional form of the utility function.

In decision support, we should report more than just the (approximate) best solution to the problem. We should present multiple attractive solutions to the decision maker which take into account the above uncertainties.

6.3.1 *The Pareto front*

One approach to presenting multiple solutions, which is often utilised in engineering, is the Pareto front. The Pareto front is a collection of decisions which, in some sense, cannot be bettered. The notion of the Pareto front comes from multi-objective optimisation where the task is to maximise several single-attribute utility functions, $U_i(\mathbf{x})$, $i = 1, 2, \ldots, m$, that is, solve the following problem:

$$\text{maximise}\, U_1(\mathbf{x}), \text{maximise}\, U_2(\mathbf{x}), \ldots, \text{maximise}\, U_m(\mathbf{x}), \mathbf{x} \in \mathcal{X}. \tag{6.22}$$

It is unlikely that the maximiser of $U_1(\mathbf{x})$ also maximises all the other $U_i(\mathbf{x})$. A trade-off is required. One example is the amount of time spent by a company constructing a wind farm. The decision maker can choose to employ more construction workers to improve the efficiency of the build, but if too many are employed at once or poorly managed, the build will be inefficient and costly. An analyst may calculate the monetary and time cost of each decision and report to the DM the decisions that form the *Pareto front*. To define the Pareto front we first need to define a *Pareto dominant* decision.

A decision $\mathbf{x} \in \mathcal{X}$ *Pareto dominates* an alternative decision $\mathbf{x}' \in \mathcal{X}$ if, and only if, every attribute of $\mathbf{x}$ is *at least* as preferable as the corresponding attribute of $\mathbf{x}'$ and at least one attribute of $\mathbf{x}$ is preferred to the corresponding attribute of $\mathbf{x}'$. That is

$$\mathbf{x} \succ_P \mathbf{x}' \iff \forall\, i\, x_i \geq x'_i \text{ and } \exists\, j \text{ such that } x_j \succ x'_j.$$

Now, the Pareto front of $\mathcal{X}$, $PF(\mathcal{X})$, is the set of all decisions in $\mathcal{X}$ which are not Pareto dominated:

$$PF(\mathcal{X}) = \left\{ \mathbf{x} : \mathbf{x} \in \mathcal{X}\, \neg \exists\, \mathbf{x}' \in \mathcal{X} \text{ such that } \mathbf{x}' \succ_p \mathbf{x} \right\}.$$

These definitions of Pareto dominant and Pareto front have been adapted from (Roijers & Whiteson, 2017). The Pareto front for the time and money example is given in Figure 6.3. In this example, a set of possible decisions are plotted. It is cognitively complex for the DM to weigh up the relative merits of such a large set of decisions. It may also be computationally costly to investigate the consequences of each decision. Thus, only investigating the decisions which form the Pareto front simplifies the analysis.

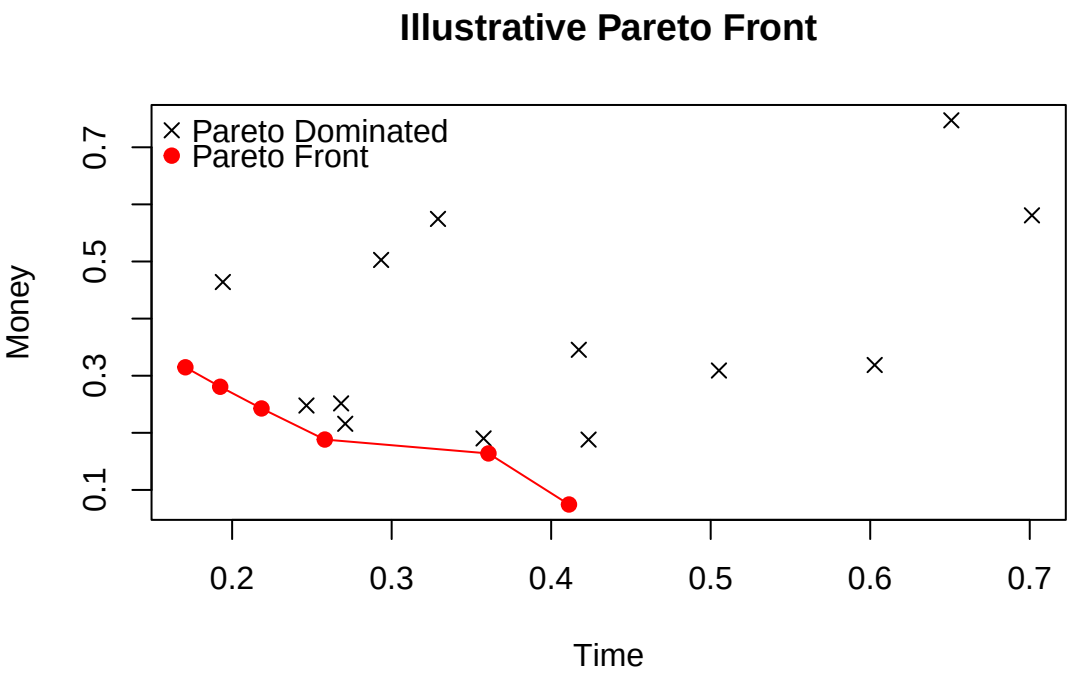

Figure 6.3: An illustrative Pareto front for a collection of decisions. In this example, smaller values of the attributes Time and Money are preferred. The Pareto front is depicted by the red dots, which are connected by the red line, and consists of decisions for which no other decision offers smaller values of Time *and* Money. The decisions which are Pareto Dominated are illustrated by black crosses. Any decision which is above/to the right of the Pareto front is a Pareto Dominated decision.

A Pareto front is useful when we want to optimise multiple competing objectives, but it is not entirely clear how objectives compete. If the DM can settle on a utility function to express their preferences, the expected utility surface can (in theory) be maximised. When the utility function depends on the outputs of an expensive and stochastic simulator, maximisation in practice is incredibly difficult. BayesOpt methodology is well developed and can be used to find an approximate maximiser. However, there is only a relatively small amount of work on quantifying uncertainty about the optimal decision. A promising approach borrows ideas from the History Matching (HM) literature (Lawson *et al.*, 2016).

6.4 *History matching*

HM is a procedure used to identify parameter settings of for a simulator which are consistent with observations from the corresponding physical system (Craig *et al.*, 1997; Domingo *et al.*, 2020). Suppose we have observed data $\mathbf{y}$, are able to specify a bounded domain $\mathbf{X} \subset R^k$ of simulator inputs, and wish to infer parameters $\mathbf{x}$ of a model, statistical or otherwise, $y(\cdot)$.

The usual Bayesian and frequentist inference procedures aim to identify values of $\mathbf{x}$ which best match y_{obs}, the observed data. This rests on the assumption that data y_{obs} could be generated from the physical process which the simulator, $y(\cdot)$, aims to mimic. HM allows for many structures for y_{obs}. The nature of the work in this thesis means we only consider $y_{\text{obs}} \in \mathbb{R}$. HM addresses a more critical question than inference; are there *any* values of $\mathbf{x} \in \mathcal{X}$ which are consistent with y_{obs}? As a corollary, if there are values of $\mathbf{x}$ which are consistent with the observed data (allowing for various sources of uncertainty) then the HM toolkit allows us to find these values. These values are referred to as "Not Ruled Out Yet" (NROY). Let us consider an illustrative example. Consider the following simulator

$$y(x) = 0.6\sin(2\pi x) + 2\cos(3\pi x) + \sin(4\pi x) + 2x + \varepsilon^*,\ x \in [0,1] \tag{6.23}$$

where $\varepsilon^* \sim \mathcal{N}(0,1)$ is an error term to account for all possible sources of uncertainty. Note that the deterministic part of Equation (6.23) is identical to Equation (6.19). We later decompose and discuss sources of error and uncertainty. This simulator is tractable and simple, we use this in place of a computationally expensive and complex simulator. Now suppose that we have observed the value $y = 2.5$ from the corresponding physical system. A frequentist could find the best input, $\hat{x}$, by minimising MSE or by specifying and maximising an appropriate likelihood function. Minimising MSE leads to one solution; $\hat{x} = 0.647$. The lack of uncertainty quantification about appropriate values of $\mathbf{x}$ is a concern.

Uncertainty quantification is at the heart of Bayesian inference thus is a promising alternative. A Bayesian would specify their prior $x \sim \pi(x)$ then perform the usual Bayesian update. Since $y(\cdot)$ is thought to be complex, no conjugate form will exist thus a numerical scheme such as MCMC will be used, possibly assisted by a surrogate. Using Stan (Stan Development Team, 2020) and adopting the prior $x \sim \mathcal{U}(0,1)$ we constructed an MCMC sampler. The MCMC sampler aimed to

obtain 10^4 un-autocorrelated draws. With a burn in period of 10^8 iterations, a further 10^8 iterations were performed and thinned by a factor of 10^4 to obtain 10^4 draws from $\pi(x \mid y)$. Observing the histogram of posterior samples (right hand sub-figure of Figure 6.4). The two modes are centred on the two local maxima of $f(\mathbf{x})$, thus the posterior densities seem reasonable. However, the trace plot (left hand sub-figure of Figure 6.4) appears to be 'sticky' suggesting that inferences are not completely reliable and the lag 1 autocorrelation is 0.919. This is a common computational problem in Bayesian inference, despite our extensive thinning and burn-in periods.

Although we have experienced computational issues, which could be solved by running the Markov chain for a very long time, the Bayesian approach has an interpretation issue which is larger than any computational problem. What exactly does a prior, or posterior density, mean when trying to deduce appropriate decisions? Further, neither the likelihood nor posterior density are telling us which candidate decisions are sensible, given all biases and uncertainties. Allowing for biases and uncertainties within emulator assisted calibration frameworks is well understood (Kennedy & O'Hagan, 2001; Goldstein & Rougier, 2009) but if there is serious simulator-reality discrepancy, this may be due to a combination of misspecification of the underlying science, an error in the code or poor choice of tuning parameters in numerical solvers. Ultimately, this application of Bayesian inference does not tell us how to make sensible decisions under uncertainty.

6.4.1 *How can history matching help?*

HM finds appropriate simulator inputs based on how close $\mathrm{E}\{y(\mathbf{x})\}$ is to observed data y_{obs}, relative to all uncertainties that the DM is willing to consider. What we want to do is find decisions (parameters) which give an expected utility (simulator output) which is close to the maximum expected utility (observed data), whilst allowing for various uncertainties. These uncertainties may include the uncertainty in the utility function, which is characterised by the emulator, the uncertainty in the maximum expected utility and the uncertainty in the utility function and any simulators which feed into the utility function.

HM is based on a 'best input' approach (Kennedy & O'Hagan, 2001; Williamson *et al.*, 2013). Suppose that there exists a 'best input' (or optimal decision) $\mathbf{x}^*$ such

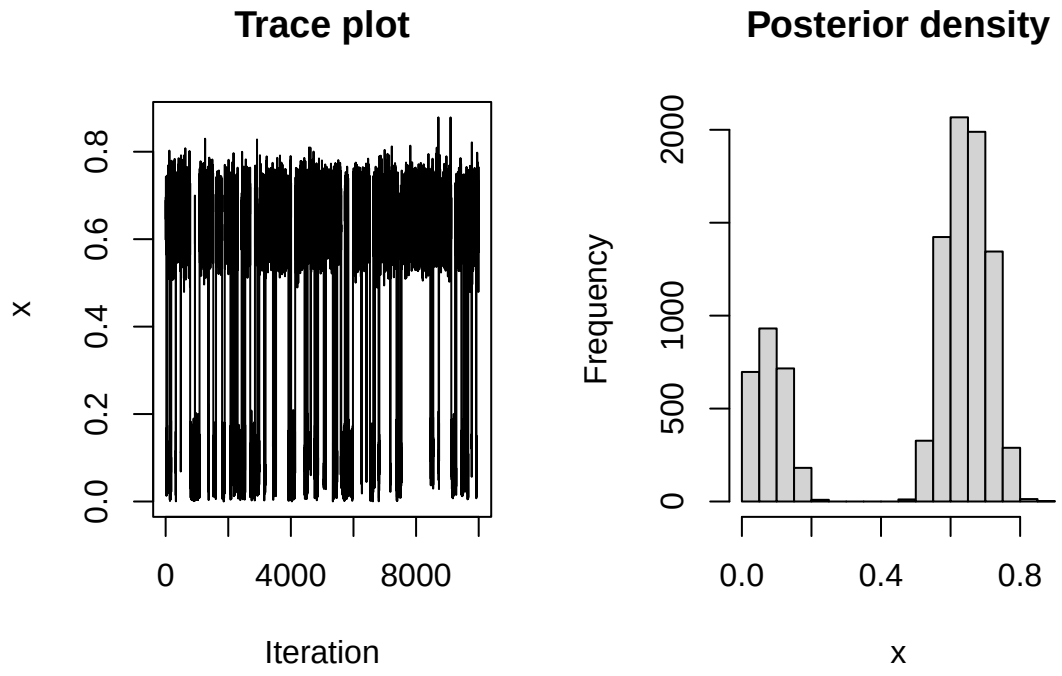

Figure 6.4: Trace plot (left) and posterior draws (right) from an MCMC sampler to infer appropriate value of x in the illustrative example.

that

$$y_{\text{obs}} = y(\mathbf{x}^*) + \varepsilon \tag{6.24}$$

where ε is an error term independent of $\mathbf{x}^*$ and $y(\cdot)$. The goal of history matching is to find inputs that return outputs which are "close" to $y(\mathbf{x}^*)$, relative to all the uncertainties in the problem that we are willing to specify. An implausibility measure, $I(\mathbf{x})$, is used to describe how appropriate $\mathbf{x}$ is for data y_{obs}. Since most common emulation methods (GPs or Bayes linear methods) use expectations and variances for prediction and uncertainty quantification, implausibility measures often have the form in Equation (6.25).

$$I(\mathbf{x}) = \frac{|y_{\text{obs}} - \mathrm{E}\{y(\mathbf{x})\}|}{\sqrt{\mathrm{Var}\{y_{\text{obs}}\} + \mathrm{Var}\{y(\mathbf{x})\} + \mathrm{Var}\{\varepsilon_{\text{MD}}\}}} \tag{6.25}$$

where ε_{MD} is a mean-zero error-like term which accounts for model discrepancy. Structured model discrepancy has not been included in Equation (6.25). Where appropriate, structured model discrepancy could be removed via the numerator

of the given implausibility function. This proposed implausibility measure is well suited to simulators with a single output. Multivariate implausibility measures have also been proposed and are usually based on the Mahalanobis distance or by specifying an implausibility function for each simulator output, $I_j(\mathbf{x})$, and taking $I(\mathbf{x}) = \max_j I_j(\mathbf{x})$. The Mahalanobis distance deems an input as implausible if, jointly, the set of outputs is unusual. The maximum implausibility approach deems an input as implausible if at least one of the outputs is unusual. Both types of multivariate implausibility have been shown be effective in practical examples (Vernon *et al.*, 2010; Williamson *et al.*, 2017).

We consider $\mathbf{x}$ to be an appropriate value, termed 'not implausible', when $I(\mathbf{x}) < I_0$. Often $I_0 = 3$. This value stems directly from Pukelsheim's 3σ rule (Pukelsheim, 1994). The 3σ rule states that, if Z follows a unimodal, continuous distribution with $\mathrm{E}(Z) = \mu < \infty$ and $\mathrm{Var}(Z) = \sigma^2 < \infty$ then

$$\mathrm{P}\left(|Z - \mu| \geq 3\sigma\right) < 0.05. \tag{6.26}$$

Thus using the 3σ rule means we incorrectly rule out a value of $\mathbf{x}$ no more than 5% of the time. Pukelsheim actually proved a slightly tighter bound than this, but $\mathrm{P}(|Z - \mu| \geq 3\sigma) \leq 4/81 \approx 0.04938$ fails to roll off the tongue so smoothly. If we are to make distributional assumptions about $f(\cdot)$ and y_{obs}, we can make this bound tighter. In the Gaussian case, it is well known that approximately 5% of values lie outside of $\mu \pm 2\sigma$ and fewer than 1% of values lie outside of $\mu \pm 3\sigma$. Using the 3σ rule even when probabalistic assumptions are made about $y(\cdot)$ may offer some robustness to misspecification of such assumptions. This is not the only form of implausibility function; any function which suitably measures the difference between the data and the simulator output, relative to all sources of uncertainty is a candidate implausibility function. For example, Wang *et al.* (2018) use p-values to quantify implausibility.

Returning to Equation (6.25); $\mathrm{E}\{y(\mathbf{x})\}$ and $\mathrm{Var}\{y(\mathbf{x})\}$ are critical to many applications of HM. These expectations and variances may be the true mean and variance of $y(\mathbf{x})$ when the moments are known. If $y(\mathbf{x})$ is intractable, then these can be replaced with the posterior (or adjusted) mean and variance of $y(\mathbf{x})$; the inclusion of emulators into the HM framework is natural.

A graphical implementation of HM is given in Figure 6.5. Here we have emulated $y(\cdot)$ and assumed that $\mathrm{Var}\{\varepsilon\} = 0.75$, $\mathrm{Var}\{y_{\text{obs}}\} = 0.25$ and further

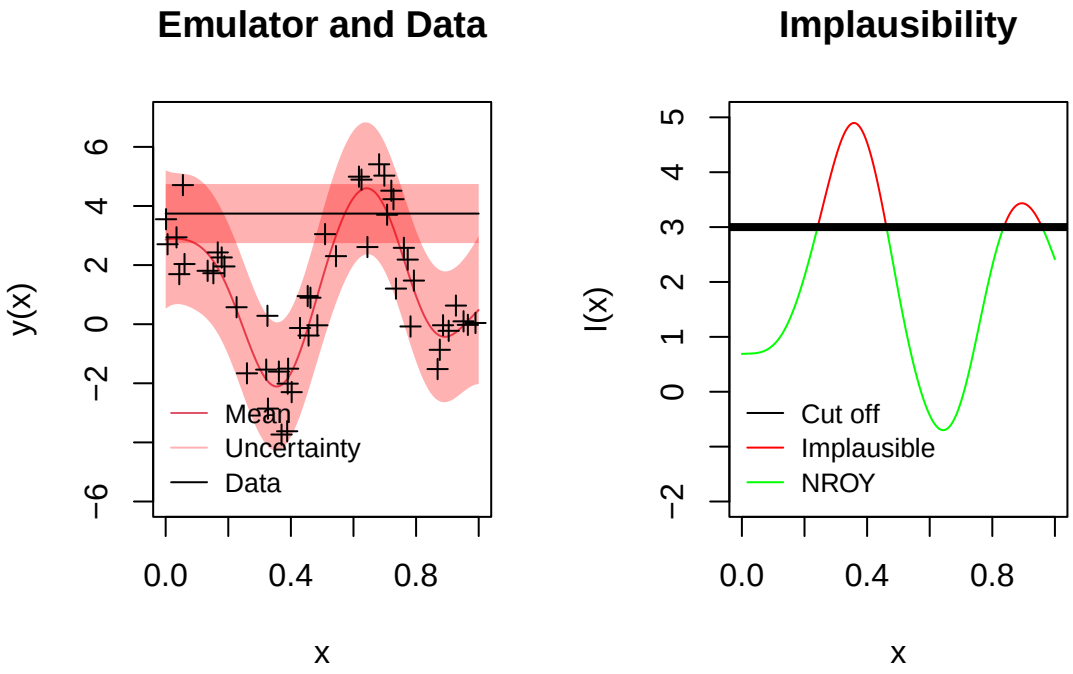

Figure 6.5: Illustrative HM procedure for the simulator given in Equation (6.23). The left hand plot shows an emulator, whose training data are the black '+' symbols and y_{obs} is given as the black horizontal line. The emulator mean has an approximately sinusoidal shape, with pink uncertainty bands about the observed data as well as the emulator. The right hand plot shows the corresponding implausibility function which is colour coded. The red parts of the function ($I(x) \geq 3$) correspond to implausible values of x whilst green regions ($I(x) < 3$) correspond to NROY regions. The black line is the cut off; $I(x) = 3$.

assumed $\text{Var}\{\varepsilon_{\text{MD}}\} = 0$ (i.e. the simulator is a perfect representation of the physical system). We see that using a cutoff of 3, some regions of the input space are incompatible with the data. The green regions of the right hand plot of Figure 6.5 represent the (wave 1) NROY space. A mathematical description of this space is

$$\mathcal{X}_{\text{NROY}} = \{x \in \mathcal{X} : I(x) < I_0\} \tag{6.27}$$

where we have chosen $I_0 = 3$, justified by Pukelsheim's 3σ rule. This first wave of HM has led to 34.2% of the original space being ruled out as inconsistent with y_{obs}. HM is well suited to sequential analysis. We can construct new emulators in the NROY space. These new emulators should have reduced uncertainty about them because emulators typically exhibit reduced uncertainty in regions where

simulator runs are dense. Since this is an illustrative example, we construct one emulator over the entire NROY space, which is the union of three distinct subspaces. When the NROY space is a union of two, or more, disjoint subsets of $[0, 1]$, it may be advantageous to construct an emulator on each disjoint subset. This will result in faster matrix calculations. Matrix multiplication is typically $O(n^2)$ and inversion is typically $O(n^3)$, now $a(n^k) < (an)^k$ for $a, k > 1$ thus if we have a distinct regions, emulation will be faster if we construct distinct emulators over each distinct region. If the function is non-stationary (which is often the case, but computationally complex to implement over a non-disjoint spaces), allowing for different parameters in different regions of the input space should produce emulators which are a better representation of $y(\cdot)$ (Gramacy & Lee, 2008).

A second wave of runs was produced. This second wave of runs and emulation allowed us to rule out a further 13.4% of the remaining space. The revised NROY space is now 57% of the original space. We see that each of the two NROY regions in Figure 6.6 corresponds to a neighbourhood around each of the two local modes of $f(\mathbf{x})$ since $y_{\text{obs}} = \max f(x)$. In this instance, we generated inputs uniformly from each sub-region of the NROY space and allowed an equal number of samples in each sub-region. Because this is a one dimensional problem, it is easy to identify the NROY space by inspection. For higher dimensional problems, the only way to identify the NROY region is by evaluation of the implausibility function (Williamson & Vernon, 2013). This is due to the black-box nature of emulators. When multiple waves of emulators have been produced, we may have to evaluate many implausibility functions. If we have performed k waves of emulation, the wave $k + 1$ space is given by

$$\mathcal{X}_{k+1} = \{\mathbf{x} \in \mathcal{X}_k : I^k(\mathbf{x}) < I_0^k\} \tag{6.28}$$

where $I^k(\cdot)$ is the wave k implausibility function and I_0^k is the wave k cut off. Note that k here is a superscript and not an exponent. This recursive definition means we *might not* have to evaluate all k implausibility functions to know if $\mathbf{x} \notin \mathcal{X}_{k+1}$. Since this requires multiple matrix calculations, this can become computationally costly. To speed up the computation, we should start by evaluating $I^1(\mathbf{x})$, then $I^2(\mathbf{x})$ and so on. If any $I^j(\mathbf{x}) \geq I_0^j$, then we should stop evaluation of further implausibility functions since $\mathbf{x}$ has been ruled out.

Adapting HM to optimisation is a promising tactic when an analyst wants to

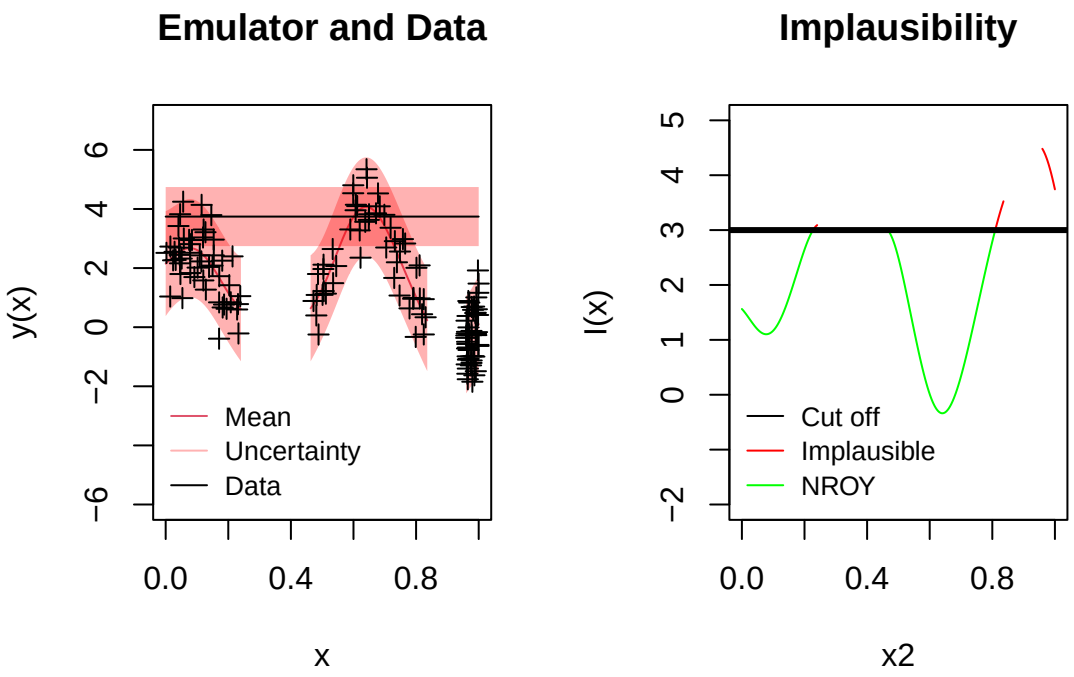

Figure 6.6: Illustrative wave 2 HM procedure for the simulator given in Equation (6.23). The left hand plot shows an emulator, whose training data are the black '+' symbols; y_{obs} is given as the black horizontal line. The emulator has an approximately sinusoidal shape, with pink uncertainty bands about the observed data as well as the emulator. The right hand plot shows the corresponding implausibility function which is colour coded. The red parts of the function ($I(x) \geq 3$) correspond to implausible values of x whilst green regions ($I(x) < 3$) correspond to NROY regions. The black line is the cut off; $I(x) = 3$. Gaps in both graphs represent the space ruled out after the first wave of HM. These are automatically ruled out at this stage, thus are not considered.

perform decision support. HM provides a solution to the problem of constructing an interpretable set of decisions which are acceptable, whilst allowing for all relevant uncertainties and biases. The interpretation here is that these decisions are within a given tolerance of the current best decision. This also allows for uncertainty about the best decision seen so far, as, when $y(\cdot)$ is stochastic, there will typically be uncertainty about $U(\mathbf{x})$. The analyst can then report $\mathcal{X}_{k+1}$ to the decision maker; it represents a set of decisions which are not clearly worse than the reported optimal decision, given all the uncertainties considered.

6.4.2 *Drawbacks of HM*

HM is not a perfect approach, there are potential drawbacks that we should be aware of.

The first potential drawback is that, once a point has been deemed NROY, it is NROY for all future waves. This means, if we were to rule out $\mathbf{x}^*$, no future waves would contain the optimal decision, or best input, as appropriate.

Another potential drawback is that, at the number of waves grows, we are increasingly likely to rule out good points. The error rate for a history matching is often defined at the wave-level. Over multiple waves of HM, the proportion of incorrectly ruled out points is compounded. For a large number of waves, this means we are in danger of ruling out large regions of the input space that leads to good decisions, or is compatible with observed data.

A third drawback is that characterising the NROY space is achieved by random sampling. This means, it is not possible to fully understand the NROY region. For example, if a sampling scheme used to describe the NROY space missed a disconnected region of the NROY space, we might not know. In practice it is hard to tell if disconnected regions exist, and even harder to tell if we have captured all of them.

6.4.3 *HM for decision support*

When using HM techniques for optimisation or decision support, the implausibility measure used must be different from Equation (6.25). This is because, (i) we have not observed data from a physical system and (ii) the modulus operation in the numerator will cause problems if the emulator predicts a value larger than the current largest value. Recall that the goal is to maximise $f(\mathbf{x})$, thus, within decision support, authors typically use an implausibility measure of the form

$$I(\mathbf{x}) = \frac{y^* - \mathrm{E}\left\{f(\mathbf{x})\right\}}{\sqrt{\mathrm{Var}\left\{y^*\right\} + \mathrm{Var}\left\{f(\mathbf{x})\right\} + \mathrm{Var}\left\{\varepsilon_{\mathrm{MD}}\right\}}}. \tag{6.29}$$

Owen *et al.* (2020) uses this implausibility function but set $I(\mathbf{x}) = 0$ whenever $\mathrm{E}\left\{f(\mathbf{x})\right\} > y^*$. Baker *et al.* (2020*a*) retains the exact value of $I(\mathbf{x})$ to communicate which decisions are much likely to be better than our reference value. If $I(\mathbf{x}) \leq -3$, then $\mathbf{x}$ is 'ruled in' as a good candidate decision, since it is much larger than the

reference value y^*. The $\mathbf{x}$ satisfying $I(\mathbf{x}) \geq 3$ are ruled out, as is standard. Then, the next wave is constructed in the range for which $|I(\mathbf{x})| < 3$ to allow the analyst to explore the decisions for which it is not clear if they are acceptable or not. An illustrative example of iterative HM inspired decision support is given in Figure 6.7. This method of fitting a sequence of emulators (not to be confused with sequential design[1] like BayesOpt or ALM) is sometimes termed "iterative refocussing" as the NROY space focuses in on a smaller region after each wave of emulation/HM (Williamson *et al.*, 2017; Volodina & Challenor, 2021). Here we are trying to optimise a utility function $U(\mathbf{x}) = 2 - x_1^2 - x_2^2$ but the observations are noisy; $y(\mathbf{x}) \sim \mathcal{N}(U(\mathbf{x}), 0.1^2)$. We specified a model discrepancy variance of $\text{Var}\{\varepsilon_{\text{MD}}\} = 10^{-6}$.

Each wave of emulation allows us to concentrate simulator runs on regions of the input/decision space which typically produce larger values of $U(\mathbf{x})$. This allows for improved decision making because the expected utility values are usually increasing. Also the emulators become more accurate within the NROY region as it shrinks; this means that the emulator mean response at later waves typically becomes a more accurate representation of $U(\mathbf{x})$. Although the NROY volume shrinks, due to the reduction in $\text{Var}\{f(\mathbf{x})\}$, $\text{Var}\{\varepsilon_{\text{MD}}\}$ is a limiting factor in the reduction of the NROY space, and to some extent the uncertainty about the optimal decision, $\text{Var}\{U^*\}$ will also be a limiting factor (in principle, this will shrink to 0 as the number of runs grows, but runs are expensive thus limited). This means that, eventually, iterative refocussing will lead to diminishing returns. We have illustrated the diminishing returns in Figure 6.8; the volumes shown in these two sub-figures correspond to the NROY volumes in each sub-figure of Figure 6.7. We see that the size of the NROY volume decreases drastically in the first two waves of emulation, but the latter two waves lead to more modest reductions in the number of NROY decisions. This is explained by $\text{Var}\{f(\mathbf{x})\}$ reaching a small value. After each wave of emulation, the average value of $\text{Var}\{f(\mathbf{x})\}$ within the NROY regions $\mathcal{X}_2$ through to $\mathcal{X}_5$ is $(1.79, 0.35, 0.087, 0.14) \times 10^{-2}$. After each of the first three waves of emulation, the average variance typically decreases by a significant fraction. The final wave leads to a slight increase in the average variance. This is likely a random artefact of the sample, the final average variance is of similar size to the third, and the reduction in the NROY volume is small,

[1]It could be argued that iterative refocussing is a batch-sequential design where the size of the batches is much larger than the number of batches.

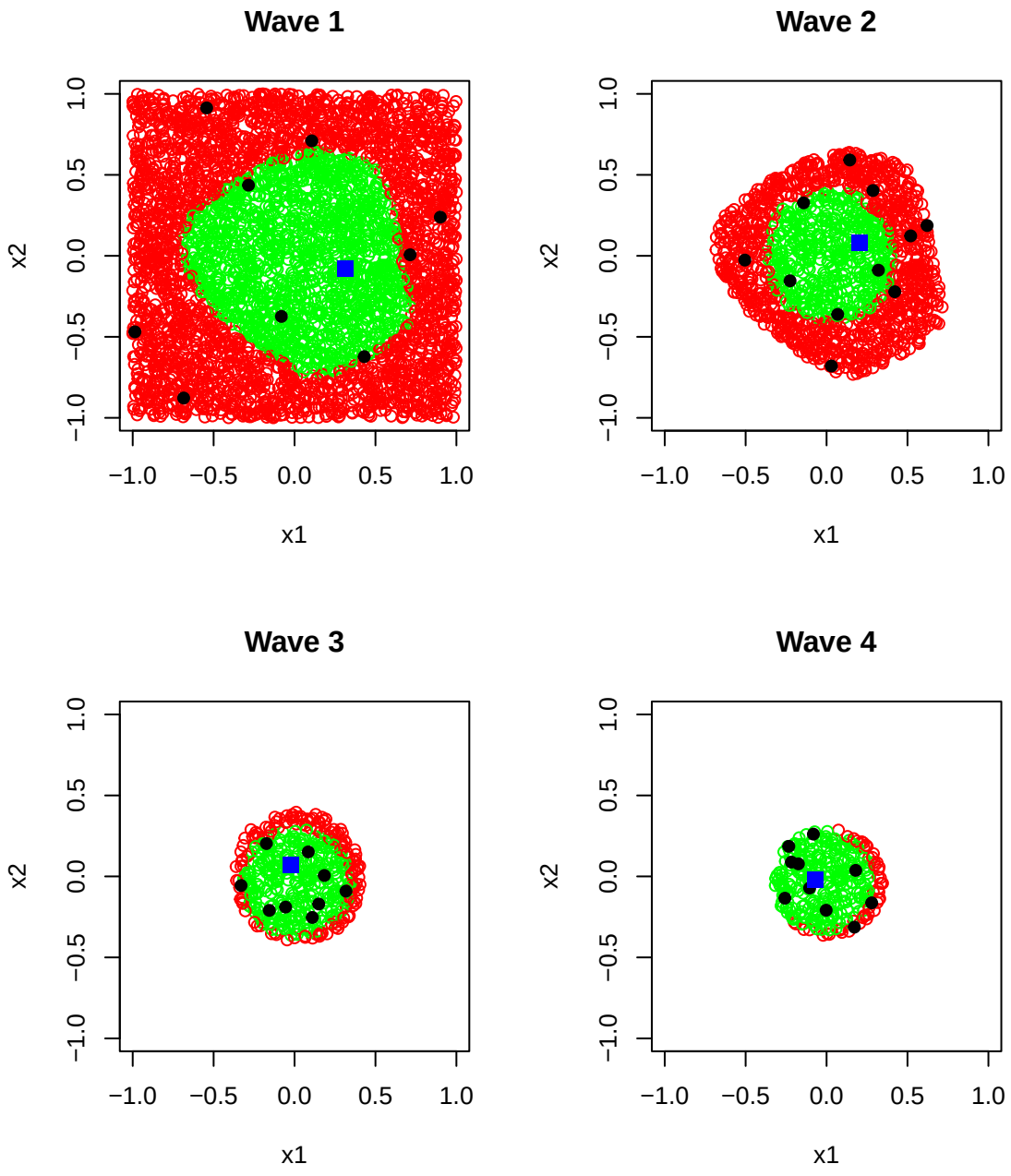

Figure 6.7: An illustrative example of using HM inspired techniques to construct a set of good decisions under uncertainty. The circles are a large set of candidate decisions; red circles denote decisions that have just been ruled out ($I^k(\mathbf{x}) \geq 3$) whereas green decisions are those which are NROY ($I^k(\mathbf{x}) < 3$). Black dots are design points for the wave k emulator with the blue square being the design point with largest expected utility at wave k.

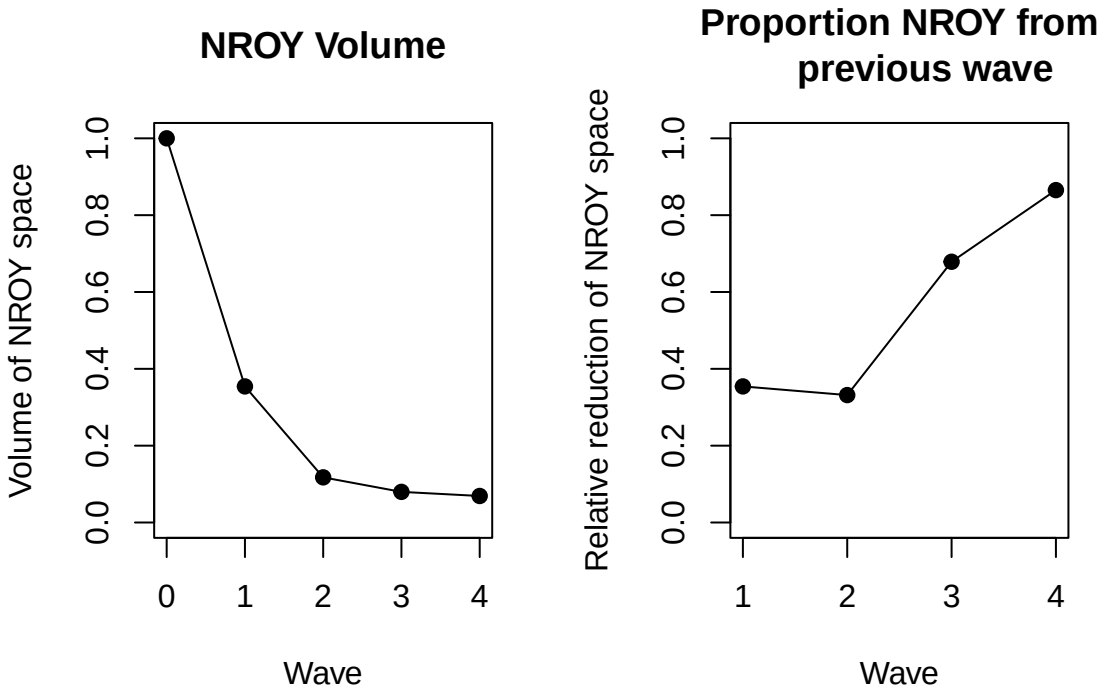

Figure 6.8: Graphs depicting how the volume of the NROY region shrinks as the number of waves increases. The left hand plot shows the ratio $|\mathcal{X}_{k+1}|/|\mathcal{X}|$ which is the proportion of NROY decisions remaining after k waves of emulation and HM; we see the first two waves lead to larger reductions in the NROY volume than the latter waves. The final wave of emulation leaves us with just 7% of the original space as NROY. The right hand plot shows $|\mathcal{X}_{k+1}|/|\mathcal{X}_k|$, that is, the proportion of decisions that are NROY after wave k that are also NROY at wave $k+1$. The right hand plot shows that the second wave was, in some sense, slightly more effective than the first wave. However the 4th wave of emulation only ruled out 13% of points in $\mathcal{X}_4$, which is small thus the HM procedure was terminated. The 0th 'wave' denotes all the initial volume (prior to emulation/HM).

which suggests we should terminate the HM procedure.

How does HM address exploration versus exploitation?

In BayesOpt, we can analyse the exploration-exploitation trade-off by studying acquisition functions. For HM inspired techniques we take a less technical approach to arguing that HM satisfies the trade-off.

During each wave of HM, we construct a design in $\mathcal{X}_{k+1}$ which is, in part, based

on an implausibility measure. The measure is of the form

$$I_{k+1}(\mathbf{x}) = \frac{y^* - m^*(\mathbf{x})}{\sqrt{v^*(\mathbf{x}) + k}} \tag{6.30}$$

where k represents other sources of uncertainty such as model discrepancy. Clearly, if $m^*(\mathbf{x})$ is large, then $I_{k+1}(\mathbf{x})$ will be small. Similarly, if $v^*(\mathbf{x})$ is large, $I_{k+1}(\mathbf{x})$ will also be small. We retain $\mathbf{x}$ for future study when $I_{k+1}(\mathbf{x})$ is small. Exploitation is achieved in two ways; firstly, by reversing the previous argument, when $m^*(\mathbf{x})$ and $v^*(\mathbf{x})$ are both small, we will not retain $\mathbf{x}$ at the next wave. This effect applies to previous waves also. We are only considering points which have previously been identified as promising or uncertain, thus we are exploiting knowledge from previous waves of emulation.

Some additional exploration, which is not present in BayesOpt, can be achieved via k. Large values of k, corresponding to larger model discrepancy and/or other sources of uncertainty, typically lead to larger NROY spaces. This means we consider a larger volume of decisions at later waves. Further, since the HM inspired approach reports uncertainty about the best decision seen so far, once we have terminated the HM procedure there is more exploration to be done. The decision maker themselves must consider decisions which are still NROY to settle on a final decision.

6.4.4 *Wave $k > 1$ designs*

A design is required for each wave of emulation. In wave 1, LH sampling is the usual approach due to the space filling properties of LHs. For waves 2 and beyond, the design space is characterised by the implausibility measure. The resulting NROY space has no obvious form when expressed as a function of $\mathbf{x}$, this is seen in Figure 6.7. Although not a rigorous term, 'blob shaped' would be a reasonable description. Ultimately the NROY space depends on the posterior mean and variance given by an emulator. Thus the only way to know if $\mathbf{x}$ is ruled out after k waves of emulation is to evaluate $I^k(\mathbf{x})$. This often leads to NROY regions with unusual shapes (Garbuno-Inigo *et al.*, 2020). As a result of the NROY space being defined in a black-box manner, design for waves $k > 1$ can be a challenging task.

The simplest approach to (uniform) design for a wave $k+1$ emulator is rejection

sampling. Usually, a large LH is constructed over the original input space and then the points which are NROY after k waves are retained for the wave $k+1$ emulator (Vernon *et al.*, 2010). This approach is reasonably efficient when the NROY space is a relatively large fraction of the original space (say, more than 1% of the original space). When the NROY space is a small fraction of the original space, the approach becomes very expensive. If, after k waves of emulation, $|\mathcal{X}_{k+1}| = p_k|\mathcal{X}|$ then if we want N NROY samples we would expect a LH of size N/p_k to give us N NROY samples. This problem is harder in higher dimensional spaces due to the curse of dimensionality and also more computationally expensive when we use non-random LHs (for example, maximin LHs). Another problem with this approach is that it returns a design of a random size. If we know we can perform no more than N runs of the simulator at each wave, we would usually want to perform exactly N runs.

There are promising alternatives to rejection sampling, based on other Monte Carlo schemes. The Monte Carlo schemes all take the same view on the problem. Sampling uniformly from $\mathcal{X}_{k+1}$ is mathematically equivalent to sampling from the following density

$$\pi_{k+1}(\mathbf{x}; I_0) \propto \begin{cases} 1, & I^k(\mathbf{x}) < I_0 \\ 0, & I^k(\mathbf{x}) \geq I_0 \end{cases}. \tag{6.31}$$

Williamson & Vernon (2013) use evolutionary Monte Carlo (EMC) and Drovandi *et al.* (2021) use sequential Monte Carlo (SMC) to construct NROY designs for which the the NROY space is a tiny subset of $\mathcal{X}$. Another promising approach, which is conceptually much simpler than the EMC and SMC approaches, is slice sampling. Within the HM context, this was first used by Andrianakis *et al.* (2017*a*) and the slice sampler for iterative refocussing is simpler than the standard slice sampling algorithm since the density is constant within $\mathcal{X}_{k+1}$ (Neal, 2003).

6.4.5 *Slice sampling for NROY regions*

We now provide some details of the slice sampling algorithm introduced by Neal (2003), which is given in Algorithm 2 and compare this to the sampler used by Andrianakis *et al.* (2017*a*), which we describe in Algorithm 3. We will aim to sample from a density $\pi(\mathbf{x})$ which is only known up to proportionality; $\pi(\mathbf{x}) = k\varphi(\mathbf{x})$ where $k^{-1} = \int \varphi(\mathbf{x})\, \mathrm{d}\mathbf{x}$ is an intractable normalisation constant. A version of the algorithm proposed by Neal (2003) is given in Algorithm 2, and a

corresponding illustration for a univariate example is given in Figure 6.9, where $\varphi(\mathbf{x})$ is a linear combination of three univariate Normal densities. The vertical blue line denotes our initial sample, x_0, and the horizontal blue line denotes $\varphi(x_0)$. The horizontal red line represents the horizontal slice $y_0 \sim \mathcal{U}(0, \varphi(x_0))$ and then the dashed red lines illustrate the sample space for the next sample, x_1. A uniform sample along the x axis where red dashed lines are present gives us our next draw from $\pi(x)$. Slice sampling is a powerful sampling technique because we only need to be able to (i) sample uniformly and (ii) solve equations (construct the horizontal slice). The equation solving aspect can be challenging. Our illustrative, one-dimensional example had four roots to find. In a complex high dimensional problem, we may not know how many roots exist.

The standard slice sampler (Algorithm 2) differs somewhat from the slice sampler for NROY regions (Algorithm 3). A similarity between the two is that they are both component wise; each element of $\mathbf{x}$ is updated one-by-one. Some steps in the standard slice sampler are not required for generating samples from the NROY region. The first difference is that in Algorithm 3 we do not calculate $\varphi(\mathbf{x}_0) = \mathbb{I}(\mathbf{x}_0)$ because we know it is 1. Secondly, we do not sample $y_0 \sim \mathcal{U}(0, \varphi(\mathbf{x}_0))$ because $\varphi(\mathbf{x})$ is either 0 or 1. Any NROY sample satisfies $\mathbb{I}(\mathbf{x} \in \mathcal{X}_{k+1}) = 1$, therefore $y_0 = 1$ in all cases. This bypasses the tricky root finding problem.

Slice sampling, as in Algorithm 3, allows us to update the ith input. The slice sampling can be made drastically more efficient by changing the limits of the uniform proposal. If it is known that $\mathcal{X}_{k+1} \subseteq [l_1, r_1] \times [l_2, r_2] \times \ldots [l_m, r_m]$ then we can replace the $\mathcal{U}(0, 1)$ random sample with a $\mathcal{U}(l_i, r_i)$ sample, where $0 \leq l_i < r_i \leq 1$. This efficiency gain may come at a cost: if an l_i or r_i are misspecified, we would neglect parts of the NROY space. For input i, suppose l_i is chosen to be too large, then any $\mathbf{x} \in \mathcal{X}_{k+1}$ such that $x_i < l_i$ would never be sampled. However, Algorithm 3 has an issue: in general, it is not *ergodic*. That is, it may not be possible to fully explore $\mathcal{X}_{k+1}$ with a single run of the algorithm. This can be amended by repeating the algorithm for many initial points $\mathbf{x}_0$ which are randomly chosen (for example, by sampling uniformly in $[0, 1]^m$ until we find an NROY point).

We formalise this in Algorithm 4.

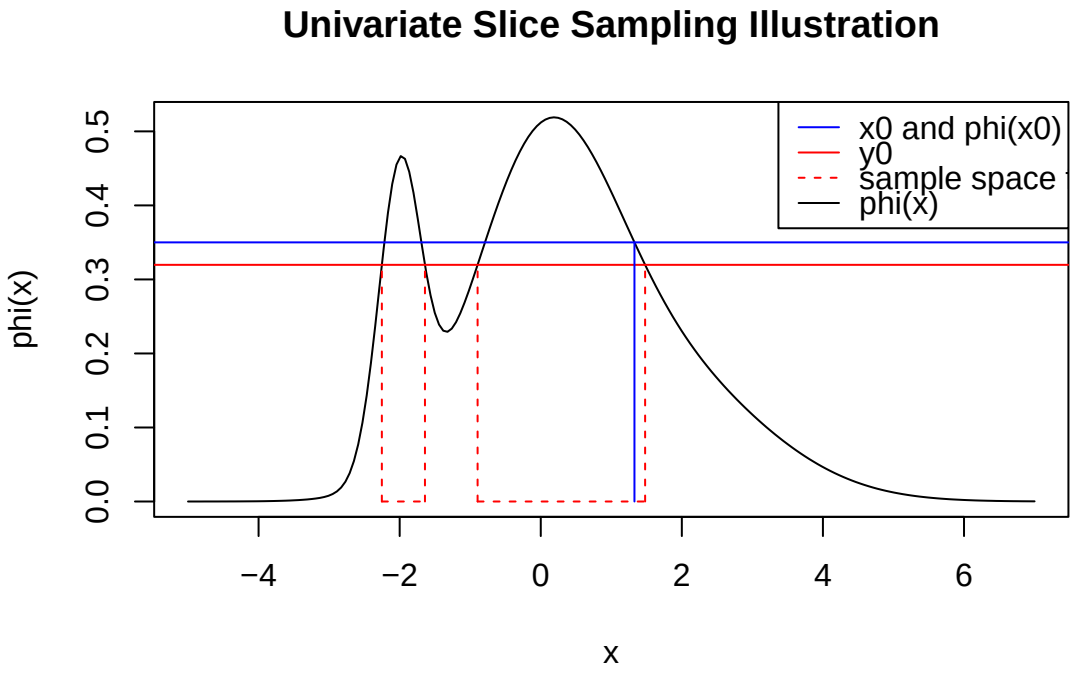

Figure 6.9: An illustration of how to sample $x_1 \sim \pi(x)$ given a draw $x_0 \sim \pi(x)$ when the only information we have about $\pi(x)$ is that $\pi(x) \propto \varphi(x)$. The black curve represents $\varphi(x)$, the vertical blue line represents x_0, the initial sample and the horizontal blue line represents $\varphi(x_0)$. The horizontal solid red line represents $y_0 \sim \mathcal{U}(0, \varphi(x_0))$. The dashed red lines denote the space in which x_1 is uniformly sampled from to obtain a new draw from $\pi(x)$.

6.5 *Discussion*

This chapter has reviewed methods for decision making under uncertainty to prepare us for the next chapter, in which we solve a decision problem using some of the illustrated techniques. We began by considering decision making as an optimisation problem. We discussed that although mathematically well defined, decision making is a difficult task. The difficulty of the problem is increased when we have limited understanding of $U(\mathbf{x})$ due to the stochastic, expensive and gradient-free nature of many simulators.

We discussed various optimisation methods, then settled on Bayesian Optimisation as being an efficient and effective approach to *decision making* with complex models. We discussed various acquisitions functions and presented decision theoretic justifications for each acquisition function, where relevant. A consequence

Algorithm 2 A single iteration of a generic, component-wise slice sampler, based upon the slice sampler presented by Neal (2003).

Require: A density known up to proportionality $\pi(\mathbf{x}) \propto \varphi(\mathbf{x})$, an initial sample $\mathbf{x}_0 \sim \pi(\mathbf{x})$ where $\mathbf{x}_0$ is an m dimensional parameter vector.
$\mathbf{x}_1 \leftarrow \mathbf{x}_0$
for $i = 1, 2, \ldots, m$ **do**
  $\varphi_0 \leftarrow \phi(\mathbf{x}_0)$ and sample $y_0 \sim \mathcal{U}(0, \phi_0)$
  Find the set $\tilde{\mathcal{X}} = \{x_i : \varphi(x_i, \mathbf{x}_{-i}) \geq y_0\}$
  Draw $\tilde{x}_i \sim \mathcal{U}(\tilde{\mathcal{X}})$ ▷ Sample uniformly from $x \in \tilde{\mathcal{X}}$
  Set $x_{1,i} \leftarrow \tilde{x}_i$ ▷ Update the ith component of x_0
end for
Return, $\mathbf{x}_1$, the new sample from $\pi(\mathbf{x})$.

Algorithm 3 A single iteration of a component-wise slice sampler used for sampling from NROY regions, based upon the slice sampler presented by Andrianakis *et al.* (2017*a*).

Require: An indicator function $\mathbb{I}(\mathbf{x} \in \mathcal{X}_{k+1})$ for identifying inputs which are NROY, an initial NROY point $\mathbf{x}_0 \in \mathcal{X}_{k+1} \subseteq [0, 1]^m$ where $\mathbf{x}_0$ is an m dimensional parameter vector.
$\mathbf{x}^* \leftarrow \mathbf{x}_0$
for $i = 1, 2, \ldots, m$ **do**
  $I^* \leftarrow 0$ ▷ Initialise an indicator variable
  while $I^* \neq 1$ **do**
    Draw $x_i \sim \mathcal{U}(0, 1)$ ▷ Change the ith input
    $x_i^* \leftarrow x_i$
    $I^* \leftarrow \mathbb{I}(\mathbf{x}^* \in \mathcal{X}_{k+1})$ ▷ Evaluate the implausibility measure at the new input
  end while
  $\mathbf{x}_1 \leftarrow \mathbf{x}^*$ ▷ Update the ith component of $\mathbf{x}_0$
end for
Return, x_1, the new sample from $\pi(\mathbf{x})$.

of using BayesOpt for decision making is that it makes decisions on behalf of the DM, which poses questions about *who* is making decisions. We argue that viewing complex, ill-defined problems as those possessing a single, optimal solution is myopic. An appropriate approach to addressing these concerns is to use *decision support* to simplify, but not solve, the problem for the decision maker. We discussed the idea of a Pareto front as a decision support tool, however, we are working within a regime where a utility function has been established but is expensive to evaluate. We therefore treat $U(\mathbf{x})$ as an unknown function which is optimised under multiple sources of uncertainty. These may include the uncer-

Algorithm 4 An ergodic variant of Algorithm 3.

Require: An indicator function $\mathbb{I}(\mathbf{x} \in \mathcal{X}_{k+1})$, a number of replications, R, a required number of samples, $n = n' \times R$, and a function `slice_sampler_one_step`($\mathbb{I}(\mathbf{x} \in \mathcal{X}_{k+1}), \mathbf{x}_0$) which takes an implausibility function, $\mathbb{I}(\mathbf{x} \in \mathcal{X}_{k+1})$ and an NROY point, $\mathbf{x}_0$ and returns a new NROY point.

```
for r = 1, 2, ..., R do
    I_r ← 0
    Initialise a matrix X_r with m columns and n' rows
    while I_r ≠ 1 do
        x_{1,r} ← U([0, 1]^m)
        I_r ← I(x1, r ∈ X_{k+1})
    end while
    for i = 2, 3, ..., n' do
        x_{i,r} ← slice_sampler_one_step(I(x ∈ X_{k+1}), x_{i−1,r})
    end for
end for
Collect all X_r into a single matrix X, with m columns and n rows.
Return X, the collection of NROY samples.
```

tainty induced by the expensive nature of the utility function, any stochasticity present in simulations, uncertainty about model parameters which are not decision parameters as well as the uncertainty about the elicited utility function.

Therefore, we settled on using HM inspired techniques to iteratively hone in on a set of feasible solutions, called the NROY space. These decisions are a set of solutions, which, given all relevant sources of uncertainty, are not worse than the best decision we have found so far. We briefly discussed some approaches to iterative construction of emulators, and in particular discussed some approaches to the wave $k > 1$ design problem. Monte Carlo methods, commonly used to perform Bayesian inference, offered a promising solution to constructing uniform designs across the NROY space. In particular, we advocate for slice sampling as it strikes a good balance of conceptual simplicity, computational simplicity and effective results. The only mechanism we need in addition to the implausibility function is a uniform sampling scheme, and all major scientific computing libraries allow for uniform sampling.

Empirical evidence tells us that these are typically much more efficient than rejection sampling when $|\mathcal{X}_k|$ is a tiny fraction of $|\mathcal{X}|$. However, we did not discuss how to get the optimum 'reference' value, which is central to the HM inspired approach. In the next chapter, we explore this further and employ a slice sampling

approach to construct a set of feasible decisions.

Chapter 7

A Decision Support Framework for Offshore Wind

Consider the problem of storing spare subassemblies, which will replace subassemblies in an offshore wind farm which have suffered a 'serious' failure. Spare parts may be stored in a warehouse near to the dock, at which spare parts will be loaded onto a boat so the spare can be shipped out to the turbine to be fitted. This problem has been considered previously in the literature. Tracht *et al.* (2013) perform a scenario analysis which investigates the effect of limited amounts of spare components and equipment on preventative maintenance. A review of various models used for analysing spare parts strategies is given in Tusar & Sarker (2022). The review stresses that not enough work is being done on constructing an optimal combination of spare parts, as once an order has been placed, the time to delivery for spare components can be sufficiently long to reduce availability. We aim to address this. The review suggests most analyses are interested in minimising downtime. Our analysis will consider availability (downtime is essentially the opposite of availability) but we will also consider the costs incurred by increasing availability.

This chapter proposes a decision support framework which is flexible enough to incorporate relevant sources of uncertainty into a decision support exercise. A key difference with our approach, compared to standard approaches, will be that we use ideas from the history matching (HM) and BayesOpt literature to construct a set of sensible decisions. BayesOpt is typically used to find a single decision which is (approximately) optimal, but the BayesOpt routine also provides us with an emulator across the entire decision space, which can be used to construct a set

of feasible decisions. Our approach is novel as we will employ ideas from HM to conduct a post-hoc analysis of the BayesOpt routine.

We will begin by outlining competing objectives a DM may consider in an offshore wind setting. We progress to formalising the decision problem and thus translating the decision problem to an optimisation problem. A novel application of BayesOpt and HM techniques then allows us to construct a set of sensible decisions; the DM will be presented with this set and uses it to make a final decision.

7.1 *The decision problem*

Recall that the Athena simulator models a turbine as being composed of 8 important subassemblies, and a ninth 'catch all' subassembly. They are as follows: 1. gearbox, 2. generator, 3. frequency converter, 4. transformer, 5. main shaft bearing, 6. blades, 7. tower, 8. foundations, and 9. catch all. The DM has two problems to solve simultaneously, whilst aiming to maintain high availability of the wind farm:

1. Which warehouse, from a set of candidate warehouses, should the spare parts be stored in?

2. What is the best way to manage the warehouse?

Good management of a sub-optimal warehouse may be much better than bad management of the optimal warehouse. Of course, we strive for optimal management of the optimal choice of warehouse. The management of the warehouse is broken down into the answers of the following two questions:

1. How many spare parts should we store for each subassembly?

2. At what point should we buy in more spare parts for the subassembly?

We therefore need to formulate a utility function over all relevant attributes and specify an appropriate prior distribution over uncertain quantities. We do not directly specify any particular prior distribution; we instead adopt the priors elicited by Zitrou *et al.* (2021). These priors were elicited from practising engineers who are able to give informed assessments of uncertain quantities. We will act as the DM as well as the analyst. In this exercise we will specify the utility

function. The utility function will express sensible preferences, but will not represent the preferences of a real DM. We stress that the emphasis of this chapter is the development of an appropriate optimisation/decision support framework, rather than precise details of utility functions or precise details of the problem itself.

7.2 *Mathematical formulation of the decision problem*

Here we will formulate the decision problem following the elicitation guidelines outlined in Chapter 2; more details can be found in Smith (2010) or Keeney & Raiffa (1976). Our problem is to choose a warehouse, k, of size W_k, and fill the warehouse with spare parts for each of the 9 subassemblies. We will allocate s_i spares for subassembly i. We also need to construct a policy for restoring the numbers of spare parts as they deplete, to ensure smooth running of the warehouse.

We begin by identifying an appropriate set of attributes. We then specify preferences through marginal utility functions, which will be combined into a single multi-attribute utility function. The first stage of elicitation of a utility function is to define all attributes relevant to the problem.

Relevant attributes

There are several attributes relevant to this problem. The key attributes are:

- The availability time series (performance) of the wind farm.
- The choice and cost of the warehouse in which spare parts are stored.
- The number of spares of each type of subassembly.
- The policy which dictates when to buy in more spare parts.

There are many more attributes a DM could consider, but this is sufficient in our case to demonstrate the complexity of the problem and the application of methodology. We would next need to decide what the consequences, c_i, of any possible decision would be. The consequences of any decision are given in Table 7.2 alongside information about the consequences. For example, the availability time series output by Athena over a 5 year period is internally processed by Athena and a length 240 vector is output. We know that this time series is stochastic. The

Label	Description	Parameter Space	Stochastic?
c_1	Availability time series	$c_1 \in (0,1)^{240}$	Yes
c_2	Choice of the restoration policy	$c_2 \in (1,99)^9$	No
c_3	Warehouse size	$c_3 \in \{50,75,100\}$	No

Table 7.1: Summary of the consequences of a future decision to be made. These variables cause changes in the stochastic and deterministic consequences and it is the decisions themselves that will form the inputs of a future emulator. The stochastic column describes how we model the consequence.

Label	Description	Parameter Space
x_1–x_9	Number of spare parts	$x_{1:9} \in \mathcal{S}^8_{x_{19}}$
x_{10}–x_{18}	Critical Percentages	$x_{10:18} \in (1,99)^9$
x_{19}	Warehouse size (number of parts)	$x_{19} \in \{50,75,100\}$

Table 7.2: Summary of the decision variables. These are the variables for which individual utility functions will be elicited, to be combined into an overall utility function. See Equation (7.1) for the definition of $\mathcal{S}^k_n$.

restoration policy is both a decision and a consequence here; the consequence is the set of p_i, the thresholds that trigger an ordering for more spare parts of type i as in Section 2.1.1, we would want to leave this as late as possible. The cost of the warehouse could be argued to be stochastic (for example, if the unit is rented, the owner may increase the rent to some uncertain value at some uncertain future time point). We assume, for simplicity, that the cost is fixed and related to its capacity. In Table 7.2 we the use notation $\mathcal{S}^k_n$. This represents the k dimensional discrete simplex which we will define as

$$\mathcal{S}^k_n = \left\{ \mathbf{x} \in \mathbb{N}^{k+1} \;\middle|\; \sum_{i=1}^{k+1} x_i = n, x_i \in \{1,2,\ldots,n-k+1\} \right\}. \tag{7.1}$$

We have chosen to define $\mathcal{S}^k_n$ in this way as it reflects aspects of our application. A warehouse of size n can have a minimum of 1 spare part of type i and can have at most $n-k+1$ spare parts of type i. This is because the other $k-1$ types of component all need at least 1 spare part. This essentially assumes that all spare subassemblies are of similar shape and size, this is clearly an over simplification which could be addressed in future work.

Our approach to eliciting the utility function will follow the advice of Smith

(2010) and Keeney & Raiffa (1976). We assume that consequences are mutually utility independent, elicit a marginal utility function for each consequence using probability or certainty equivalents and then combine the marginal utility functions into an overall utility function via a multiplicative or additive form, depending on the preferences of the DM.

7.3 *The marginal utility functions*

Utility functions are unique up to positive affine transformations. That is, $au(x)+b$ expresses exactly the same preferences as $u(x)$ for any $a > 0$ and $b \in \mathbb{R}$. For this reason, we will construct all utility functions on the unit interval. This also allows for easier elicitation, as utility and probability will be on the same scale.

Utility of availability

The availability is a time series. At time t, denote the (random) availability by $A(t)$, this will form our first consequence. In particular, the Athena simulator outputs a length L time series, $(A(t_1), A(t_2), \ldots, A(t_L))$ at equally spaced intervals. In our analysis, $L = 240$ due to various settings within Athena. We therefore define $c_1(t) = A(t)$ and let $c_1 = (c_1(t_1), c_1(t_2), \ldots, c_1(t_L))$. The utility of a time series may be time dependent, however, in this problem we will work with the assumption that a given availability has utility which does not depend on time. In other words, if we let $\tilde{u}_1(c_1(t), t)$ be the utility for $c_1(t)$ at time t then $\tilde{u}_1(c_1(t), t) = \tilde{u}_1(c_1(t))$. We would then take the utility of the entire time series to be

$$u_1(c_1) = \frac{1}{L} \sum_{j=1}^{L} \tilde{u}_1(c_1(t_j)). \tag{7.2}$$

This is just point-wise utility averaged over time. This average could be weighted if high availability is more important at some time points than others. See Ríos Insua *et al.* (2003) for examples of time-dependent utility elicitation.

We believe a utility function which allows for a varying attitude to risk would be appropriate for $\tilde{u}_1(c_1(t))$. The reason for this is that, for regions of high availability, a DM would be risk-averse, or risk-neutral, in trying to increase availability. However, whenever the availability is very low, we would be risk-seeking in order

to ensure the wind farm is viable. An appropriate functional form is:

$$\tilde{u}_1(c) = \beta^{-1}\left\{\frac{(c-a)}{\sqrt{1+b(c-a)^2}} - \alpha\right\} \tag{7.3}$$

where $a \in (0,1)$ and $b > 0$ are parameters to be elicited and $\alpha, \beta \in \mathbb{R}$ are scaling parameters which force $0 \leq \tilde{u}_1(c) \leq 1$ for any $c \in [0,1]$. α and β have the following algebraic forms:

$$\alpha = -\frac{a}{\sqrt{1+a^2b}} \tag{7.4}$$

$$\beta = \frac{1-a}{\sqrt{1+b(1-a)^2}} - \frac{a}{\sqrt{1+a^2b}}. \tag{7.5}$$

Now, since we enforce the constraints $\tilde{u}_1(0) = 0$ and $\tilde{u}_1(1) = 1$, we only need to elicit two certainty equivalents to construct a fitted form for $u_1(c)$. To do this, we will use a bisection method. First we specify the availability, $c^{(1)}$, such that $[0, 1; 0.5] \sim c^{(1)}$ then we specify $c^{(2)}$ such that $[c^{(1)}, 1; 0.5] \sim c^{(2)}$. These certainty equivalences imply $\tilde{u}_1(c^{(1)}) = 0.5$ and $\tilde{u}_1(c^{(2)}) = 0.75$. We therefore have two equations in two unknowns which can be solved for (a, b) (and thus α and β are also found). The chosen values were $u_1(0.9) = 0.5$ and $u_1(0.95) = 0.75$ which implies $(a, b) = (0.95, 50)$. To validate the utility function, we could ask for $c^{(3)}$ such that $\tilde{u}_1(c^{(3)}) = 0.25$. We would then compare the value of c which gives $\tilde{u}_1(c) = 0.25$ to the elicited $c^{(3)}$. An illustration of the fitted $\tilde{u}_1(c_1)$ is given in Figure 7.1. We see that when $c_1 < 0.9$, we have risk-seeking behaviour, for $c_1 \geq 0.9$, the behaviour is approximately risk-neutral.

Utility of warehouse size

We assume that the smallest warehouse possible is preferred, as this will likely be cheaper. The sizes of the warehouses are $\{50, 75, 100\}$, thus we need $u_2(100) = 0$ and $u_2(50) = 1$. Since the warehouses are discrete decisions, and there are only three warehouses in our problem, it may be better to simply elicit a value for $u_2(75)$ than specify a utility function. However, this would not allow for feedback or analysis of risk aversion. We propose using a risk averse utility function for

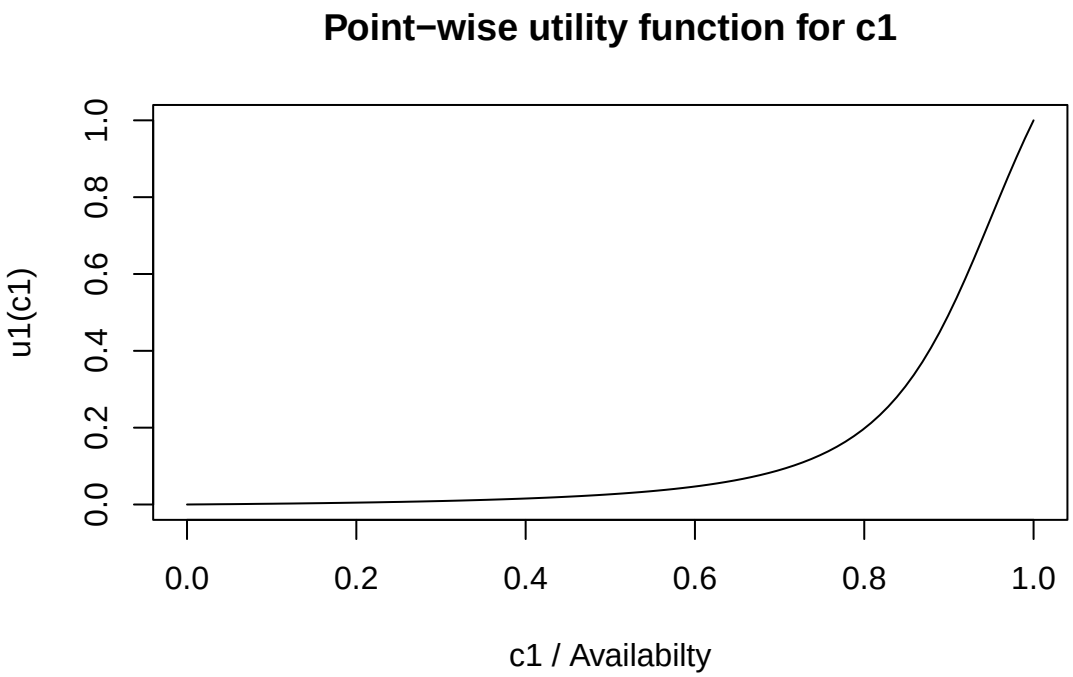

Figure 7.1: Plot of the fitted utility function for point-wise availability; $\tilde{u}_1(c_1)$.

this part of the problem. A simple, risk averse, form is

$$u_2(c_2) = 1 - \left(\frac{c_2 - a}{b - a}\right)^2 \tag{7.6}$$

where (a, b) are constants to ensure $u(50) = 1$ and $u(100) = 0$. For this problem we have $a = 50$ and $b = 100$. We could then ask the DM what their utility for $c_2 = 75$ is and compare to the value $u_2(75) = 0.75$. We could also try other functional forms. However, since this is an extended illustrative example, we settle on

$$u_2(c_2) = 1 - \left(\frac{c_2 - 50}{50}\right)^2 \tag{7.7}$$

as it reflects realistic beliefs about the problem whilst retaining simplicity.

Utility of spares restoration

When spares are restored, we aim to fill up the initial allocation of spares for that particular type of component. We adopt a policy that reflects that spares should

be restored as late as possible, without adversely affecting availability. Our utility function should then reflect that restoring the numbers of spares happens as late as possible. Let the number of remaining spare parts for subassembly i be $s_i(t)$ (which from hereon in will be implicitly a function of time). One way to specify when spares are restored is to specify a value s_i' for the ith subassembly such that, at the instant when $s_i < s_i'$ is first satisfied, an order for more spare parts is placed. We would like to avoid having a utility function which depends on another decision, since $s_i' \leq s_i$. We can instead express this decision as an independent attribute by finding the proportion of spares, p_i such that when $\frac{s_i}{s_i'} < p_i$ more spares must be ordered, where $p_i \in (1, 99)\%$ is the 'critical percentage' for ordering more spares. This phrasing of the problem in terms of proportions has two advantages. It makes specifying the utility function easier, as the proportion and the number of spares are independent attributes. Additionally, maximising $U(\mathbf{x})$ when there is a complex dependence structure between elements of $\mathbf{x}$ will be difficult.

Therefore, the third set of decision variables will be $x_{10:18} = (p_1, p_2, \ldots, p_9)$. The following utility function is an appropriate choice for the restoration policy for the jth spare component

$$u_{3,j}(c_{3,j}) = 1 - \left(\frac{1 - c_{3,j}}{99 - 1}\right)^2 \tag{7.8}$$

which is a risk-averse utility function which decreases as $c_{3,j} = x_{j+9}$ increases. To combine this into an overall utility function for the spares restoration policy, we could construct a weighted average

$$u_3(c_3) = \sum_{j=1}^{9} k_j u_{3,j}(c_3). \tag{7.9}$$

To elicit the weights, k_j, we can apply standard multi-attribute elicitation techniques, such as those in González-Ortega *et al.* (2018). We however have no preference for one type of spare over another, hence we take $k_j = \frac{1}{9}$ for all j. This results in the utility function for c_3 being

$$u_3(c_3) = 1 - \frac{1}{9} \sum_{j=1}^{9} \left(\frac{1 - c_{3,j}}{98}\right)^2. \tag{7.10}$$

Combining all attributes

To combine the attributes we will use a probability equivalence approach (González-Ortega *et al.*, 2018). Recall that c^* is the vector of the best possible set of consequences (those which maximise the marginal utility functions). Also recall that c^0 is the vector of the worst possible consequences, so, minimum availability, restoring spares as soon as one spare part has been used and opting for the largest warehouse. Recall that $u(c^*) = 1$ and $u(c^0) = 0$. Now let c_j^0 be the consequence which has all consequences at their worst value, except the jth consequence which takes its best value. We can also extend j to be a collection of consequences (so everything contained in the set j is at its best consequence). We propose using an additive utility function

$$u(c) = \sum_{j=1}^{3} w_j u_j(c_j) \tag{7.11}$$

where w_j are non-negative weights which sum to 1.

To specify the weights, we elicit p_1 such that $[c^0, c^*; p_1] \sim c_j^*$. In our case, we settled on the value $p_1 = 0.2$, thus $w_1 = 0.8$. We next elicit p_2 such that $[c^0, c^*; p_2] \sim c_{(1,2)}^*$. In this case, we choose $p_2 = 0.1$, hence $w_2 = 0.1$. The sum to unity constraint implies $w_3 = 0.1$. We could ask a feedback question to verify that the weights are reasonable, but we stop here for simplicity. Therefore our final expected utility function, as a function of $\mathbf{x}$, is

$$U(\mathbf{x}) = 0.8U_1(\mathbf{x}) + 0.1U_2(\mathbf{x}) + 0.1U_3(\mathbf{x}) \tag{7.12}$$

where $U_i(\mathbf{x}) = \mathrm{E}_{\boldsymbol{\theta}}\{u_i(c(\mathbf{x}, \boldsymbol{\theta}))\}$ is the expected utility function for the ith decision and averaging is performed over all possible consequences induced by decision $\mathbf{x}$ when the state of knowledge is quantified by $\boldsymbol{\theta} \sim \pi(\boldsymbol{\theta})$.

7.4 *Decision support strategy*

Our problem is to maximise $U(\mathbf{x})$. This is challenging for multiple reasons:

(1) $U(\mathbf{x})$ is expensive, stochastic and black-box in nature.

(2) We want to communicate uncertainty in the solution to the problem, that is, construct a set of 'sensible' decisions.

(3) $\mathbf{x}$ has a complex form:

(i) The warehouse aspect of $\mathbf{x}$ means $U(\mathbf{x})$ is a combination of 3 different, possibly related, functions.

(ii) Since $x_{1:9} \in \mathcal{S}_n^8 \subset \mathbb{N}^9$, with $n \in \mathbb{N}$, the use of gradient-based optimisers is difficult.

To overcome challenges (1) and (2) we will use a hybrid, novel union of BayesOpt and HM inspired techniques. Our approach will be to first perform a run of BayesOpt to find the (approximate) optimal decision, $\hat{\mathbf{x}}$, then subsequently use rounds of iterative refocussing to rule out clearly sub-optimal decisions. For the refocussing, we will use implausibility measures of the form:

$$I(\mathbf{x}) = \frac{\mathrm{E}\{U(\hat{\mathbf{x}})\} - \mathrm{E}\{U(\mathbf{x})\}}{\sqrt{\mathrm{Var}\{U(\hat{\mathbf{x}}) - U(\mathbf{x})\}}} \tag{7.13}$$

where $\hat{\mathbf{x}} = \arg\max_{\mathbf{x} \in X} m^*(\mathbf{x})$ where X is the set of inputs at which $U(\mathbf{x})$ has been evaluated, and $m^*(\mathbf{x})$ is given by Equation (3.19). That is, the largest expected value of $U(\mathbf{x})$. First note that $I(\mathbf{x})$ can be negative. $I(\mathbf{x}) < 0$ indicates that $\mathbf{x}$ is predicted to be a better decision than $\hat{\mathbf{x}}$. The next novel feature is that, the denominator of Equation (7.13), when expanded, will contain a covariance term since $\mathrm{Var}\{X - Y\} = \mathrm{Var}\{X\} + \mathrm{Var}\{Y\} - 2\mathrm{Cov}\{X, Y\}$. The covariance term will vanish if $\mathbf{x}$ and $\hat{\mathbf{x}}$ correspond to different warehouses. This choice of implausibility measure is a consequence of our novel approach. It is much more common to assume independence between the 'reference' value (data) and the model, since data will typically be generated by a physical process which would be independent of any models. In the case of optimisation, the reference value may have come directly from the simulator, rather than the emulator, as in Owen *et al.* (2020), thus we can assume independence. Alternatively, we may have some knowledge of what the maximiser already is, as in Nguyen & Osborne (2020). We however will be comparing one emulator prediction to many others and so correlation is present and must be accounted for. We will use a cut off rule $I(\mathbf{x}) > 3$, motivated by Pukelsheim's 3σ rule (Pukelsheim, 1994), to rule out decisions which are thought to be worse that $\hat{\mathbf{x}}$.

We approach (3i) by performing separate BayesOpt routines/constructing independent emulators for each warehouse. It may be possible to specify a suitable covariance structure between warehouses, however our approach has some

advantages. The first is simplicity; constructing independent emulators is computationally, programmatically and conceptually simpler than a joint emulator for all warehouses. The fact that the choice of warehouse changes the permitted values of $\mathbf{x}$ could make construction of a joint emulator very challenging. Our approach allows us to make use of large, parallel computing facilities in a simple way. We will perform a BayesOpt run for each warehouse independently of the other warehouses. Using the Rocket HPC facility at Newcastle University allows us to perform the runs for separate warehouses simultaneously. This approach also offers more computationally efficient emulators. When, for each warehouse, we have observed simulator runs at n locations, independent emulator prediction will have cost $O(n^2)$ and the cost of inference will be $O(3n^3)$. Note that the cost of prediction is *not* multiplied by 3 since the inputs tell us which emulator to use for prediction. Joint emulation will have prediction cost $O((3n)^2)$ and inference cost $O((3n)^3)$. If another problem requires more warehouses (or in a more general problem, the function to be emulated is composed of more cases), the cost of the joint case will be much greater. If iterative refocussing allows us to rule out a warehouse, then the independent emulation strategy allows us to act as if that warehouse was never part of the problem, providing further computational savings.

Now, (3ii) is dealt with by first relaxing the maximisation of the acquisition function to be over a continuous domain. Let the relaxed maximiser be $\tilde{\mathbf{x}}$. This may give us an invalid point at which to next run the model (the numbers of spares *must* be integers). To satisfy the integer constraint, we try all valid solutions within a given domain of $\tilde{\mathbf{x}}$. We construct a set of candidate solutions by rounding each element of $\tilde{\mathbf{x}}_{1:9}$ either up to the nearest integer or down to the nearest integer. This returns $2^9 = 512$ candidate solutions, not all of which are *valid* candidate solutions. Candidate solutions which do not satisfy $\sum_{i=1}^{9} x_i = x_{19}$ are discarded as they are not members of the decision space. We then evaluate the acquisition function at each of the remaining candidate inputs and then run the simulator at the input from the set of candidate decisions which returns the largest value of the acquisition function. This may not be the true maximiser of the acquisition function, but should be a 'good' location at which to query $U(\mathbf{x})$.

Note that the iterative refocussing cannot continue *ad infinitum*. We must terminate the procedure at some point. When any of the following conditions hold, we should stop the procedure (Vernon *et al.*, 2010):

(1) Emulator variance is much smaller than other sources of uncertainty.

(2) Computational resources have been exhausted.

(3) The entire parameter space has been ruled out as implausible.

(4) The reduction in the size of the NROY volume is negligible.

Note that in our analysis, condition (3) will never happen as the maximum value cannot be ruled out. In our analysis, we will not consider other sources of uncertainty such as model discrepancy, but they can be incorporated and used to terminate HM. See Ling *et al.* (2014) and Brynjarsdóttir & O'Hagan (2014) for guidance on eliciting model discrepancy. Our stopping mechanism will be exhausting our computational resources and/or observing a negligible change in the NROY region.

We have described all relevant details of our problem and motivated appropriate methodology for decision support. A succinct description of our approach is as follows:

1. Run the simulator at collection of inputs to construct a Wave 1 emulator. We will use BayesOpt to choose the design points.
2. Find the (approximately) optimal decision $\hat{\mathbf{x}}$ and run a HM procedure to construct the NROY set.
3. Within the NROY set, construct a space-filling design and run the simulator at these design points.
4. Fit an emulator to the new training data and run a HM procedure to construct the new NROY set.
5. If none of the stopping criteria are met, go to step 2. Otherwise, terminate the procedure and report the NROY set to the DM.

7.5 *Wave* 1: *Bayesian optimisation*

7.5.1 *Volume of the initial decision space*

Prior to performing any intensive calculations we will first calculate the volume of the non-implausible region. Typically this would be 1 as, usually, all inputs are

continuous and can be rescaled to occupy $[0,1]^p$ for some $p \in \mathbb{N}$. In our problem, we have discrete inputs. We assign each warehouse-spares combination a volume of 1. This is the volume implied by the spares restoration policy. Now the initial decision space, $\mathcal{X}_1$, is given as

$$\mathcal{X}_1 = \bigcup_{n \in \{50,75,100\}} (\mathcal{S}_n^8 \times (1,99)^9). \tag{7.14}$$

Now we will count the numbers of ways to fill each warehouse with spares. For warehouse i let this number be V_i. The total volume is $|\mathcal{X}_1| = V = \sum_{i=1}^{3} V_i$. The number of members of $\mathcal{S}_n^k$ is $\binom{n-1}{k}$. This is because, using the 'standard' definition of the discrete simplex, denoted $\mathcal{T}_t^k$ where $x_i \in \{0,1,\ldots,t\}$ and $\sum_{i=1}^{k+1} x_i = t$ has size $|\mathcal{T}_t^k| = \binom{t+k-1}{k}$ (Costello, 1971). Substituting $t = n-k$ gives $|\mathcal{S}_n^k| = \binom{n-1}{k}$.

Now, if each discrete decision has volume 1, then the total volume of the decision space is

$$\begin{aligned} V &= \binom{49}{8} + \binom{74}{8} + \binom{99}{8} \\ &= 450978066 + 15071474661 + 171200862756 \\ &= 186723315483 \approx 2 \times 10^{11} \end{aligned}$$

which is also the number of spares-warehouse combinations.

7.5.2 *BayesOpt implementation details*

The first emulator will be a homoscedastic Gaussian process emulator constructed by employing BayesOpt techniques. As noted before, emulators for different warehouses are independent of each other. The remaining decision variables, x_1 – x_{18} can then be fed into the emulator. These variables are transformed to a $[0,1]$ scale. From hereon, when we refer to (elements of) $\mathbf{x}$, we will mean the version scaled to $[0,1]$, unless stated otherwise. Now since the first 9 inputs sit on a simplex, one of them is redundant, given the rest. Therefore, our emulators only use the decision variables $x_2 - x_{18}$. We can also show that inclusion of x_1 alongside $x_2 - x_9$ induces a non-stationary covariance function. This is because $(x_1 - x_1')^2 = \left[(1 - \sum_{i=2}^{9} x_i) - (1 - \sum_{i=2}^{9} x_i')\right]^2$. This may also cause identifiability issues with any lengthscale parameters, as the distance between individual elements of $\mathbf{x}$ and $\mathbf{x}'$

essentially appears twice.

In the BayesOpt stage, we use a simple constant-mean GP with all parameters estimated via a MAP estimate. Recall we are trying to optimise $U(\mathbf{x})$, which depends on the expensive, stochastic Athena simulator. Since we view our approach as a unification of decision making and decision support, we will use an acquisition function that can be employed in the Bayes linear or fully Bayesian frameworks. Our acquisition function will be a UCB acquisition function with $\nu_n = 3$:

$$\alpha_n(\mathbf{x}) = \mu_n(\mathbf{x}) + 3\sigma_n(\mathbf{x}). \tag{7.15}$$

We have chosen $\nu_n = 3$ to tie in with the future rounds of iterative refocussing — this choice of acquisition function is related to many common implausibility measures. Thus, if $U(\mathbf{x}) = \mu_n(\mathbf{x}) + 3\sigma_n(\mathbf{x})$ it would have small implausibility. The choice of $\nu_n = 3$ is a fairly large value for ν_n; we are encouraging more exploration. One argument against using a GP in this example is that the utility function is constrained to $(0, 1)$ but the GP predictions occupy the full real line. We are simply using the GP as a computationally convenient modelling choice. Note that in earlier stages of the BayesOpt routine, it is likely that for some values of $\mathbf{x}$ that $\mu_n(\mathbf{x}) + 3\sigma_n(\mathbf{x}) > 1$. This will be because $\sigma_n(\mathbf{x})$ is large. However if $\mu_n(\mathbf{x}) + 3\sigma_n(\mathbf{x}) > 1$ it is likely that we will be query $U(\cdot)$ at a point close to $\mathbf{x}$ and the uncertainty will be resolved. Therefore, by the end of the BayesOpt routine, predictions should lie in $(0, 1)$ with high probability.

We run a BayesOpt scheme for each of the three warehouses. The idea is to find the best possible inputs for each warehouse to give it the best possible chance of being retained as NROY. The data in this instance will be realisations of $u(\mathbf{x})$, with individual realisations denoted by $u(\mathbf{x})_i$. To fit and train emulators, a single 'observation' will be the mean of 30 i.i.d. realisations of $u(\mathbf{x})$, that is

$$y(\mathbf{x}) = \frac{1}{30}\sum_{i=1}^{30} u(\mathbf{x})_i \tag{7.16}$$

$$= U(\mathbf{x}) + \varepsilon \tag{7.17}$$

where $\varepsilon \overset{\text{iid}}{\sim} \mathcal{N}(0, \lambda^2)$ is assumed random error. The choice of 30 replicates is a trade-off between computing time and accuracy. A small number of replicates will be inexpensive, but give a noisy perspective of $U(\mathbf{x})$. A larger number of replicates

will give a clearer perspective of $U(\mathbf{x})$, and, by the Central limit theorem (CLT), a large enough number of replicates means $y(\mathbf{x})$ should be approximately Normal.

Each scheme will be initialised with 40 $(\mathbf{x}, y)$, pairs with the initial design chosen by simple random sampling. The GP hyperparameters are estimated based on these 40 runs. Since this is a small sample size, we update the GP hyperparameters after every 40 acquisitions. We perform acquisitions until we have observed 1000 y values, or the pre-defined wall-clock simulation time has elapsed. Usually, BayesOpt terminates when an optimisation-based stopping rule is satisfied. However, since $U(\mathbf{x})$ is not observed perfectly, but with noise, there will be uncertainty about $U(\mathbf{x})$ for finite numbers of replications and observations. Performing additional acquisitions *after* conventional stopping rules have been satisfied, allows us to reduce uncertainty about $U(\cdot)$. This allows us to reduce uncertainty about the optimal decision $U(\hat{\mathbf{x}})$ which, in turn, may lead to smaller NROY spaces after iterative refocussing or HM. This should further simplify the decision that the DM has to make, as the final NROY volume should be smaller than if we terminated BayesOpt when a stopping rule has been met.

7.5.3 *Wave* 1 *results & analysis*

Once the BayesOpt schemes had been run, we checked the quality of the fitted emulators via a leave-one-out (LOO) procedure. We used a standard LOO procedure; re-fitting the emulators without $(\mathbf{x}_i, y_i)$ and then use this reduced data set to predict y_i. Let the LOO prediction for y_i have mean m_i and variance v_i. We then define standardised LOO prediction errors by $\text{SLPE}_i = (y_i - m_i)/\sqrt{v_i}$. These showed, that for each warehouse, there was an issue with x_9. The plots within Figure 7.2 show a 'tail' of residuals for small values of x_9 (the catch all subassembly). For larger values of x_9, the cloud of residuals looks uncorrelated. Plots of SLPEs against each input (for every warehouse) are given in Appendix A.1. To improve the emulator, we introduced a non-constant mean function: $\mu(\mathbf{x}) = \beta_0 + \beta_1 \log(x_9 + 10^{-8})$. The addition of 10^{-8} avoids the undefined value $\log(0)$. The value 10^{-8} could be replaced by an unknown constant to be inferred, but we found that this initial guess value to satisfactory emulators. For the revised emulators, we integrated out the β coefficients by adopting the prior specification $\beta_i \overset{\text{iid}}{\sim} \mathcal{N}(0, 0.5^2)$ for $i \in \{0, 1\}$. This led to emulators with improved LOO diagnostics, which are illustrated in Figure 7.3. These residuals suggest that the revised emulator offers a better es-

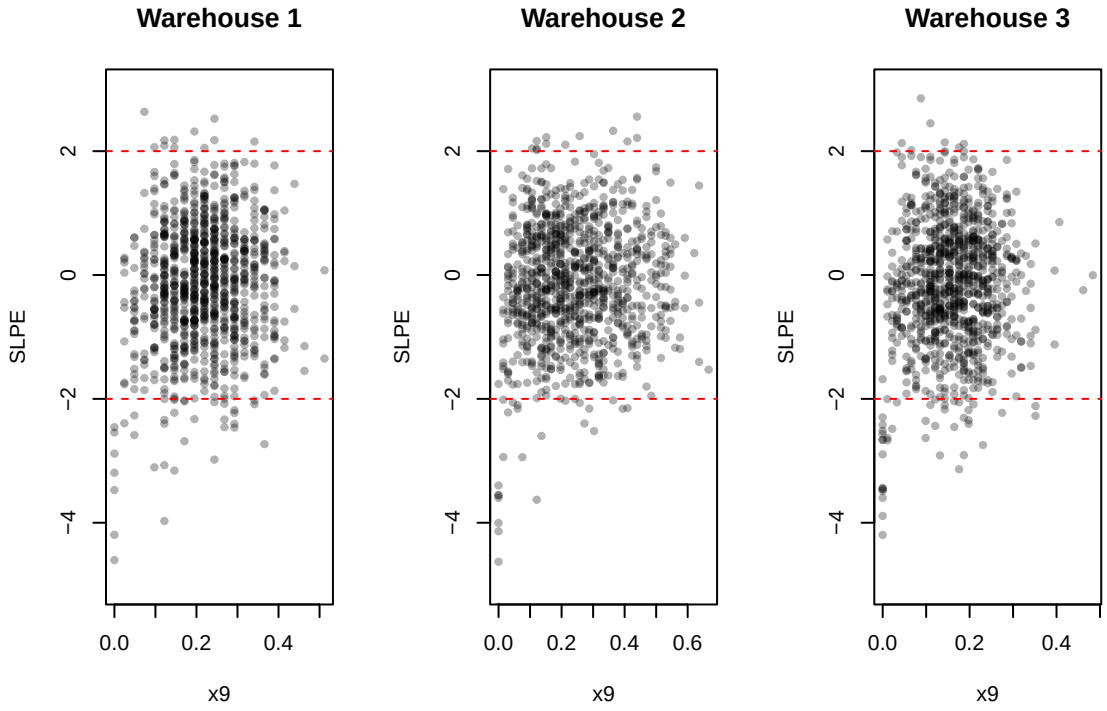

Figure 7.2: SLPEs plotted against x_9; the dashed red lines denote ± 2 and the x axis is on the standardised $[0, 1]$ scale. We see, in each case, that there is a 'tail' of residuals for small values of x_9. Each emulator has at least one SLPE larger than 4 in modulus, suggesting a serious discrepancy between the emulator and $U(\mathbf{x})$. The vertical bands of residuals are present since x_9 takes values from a finite set.

timate of $U(\mathbf{x})$ than the first emulator. In this section we have only shown the SLPEs plotted against x_9; plots of SLPEs against inputs $x_1 - x_{18}$ are given in Appendix A.2. A suite of diagnostic plots in Figure 7.3 suggest that the emulator is an improved and appropriate surrogate for $U(\mathbf{x})$; the LOO residuals now appear to be uncorrelated with x_9, the histograms of the LOO errors are close to that of a standard Normal distribution (the distribution of SLPEs for Warehouse 1 has a slightly longer left tail than one might expect) and the QQ plots further indicate the LOO errors are well described by a standard Normal distribution. We also verify that the cut off of 3 is appropriate by checking quantiles of the Normal distribution. Table 7.3 shows that over 95% of the LOO errors lie in $(-3, 3)$, thus there is agreement with Pukelsheim's 3σ rule and we are now in a position to rule out inappropriate decisions.

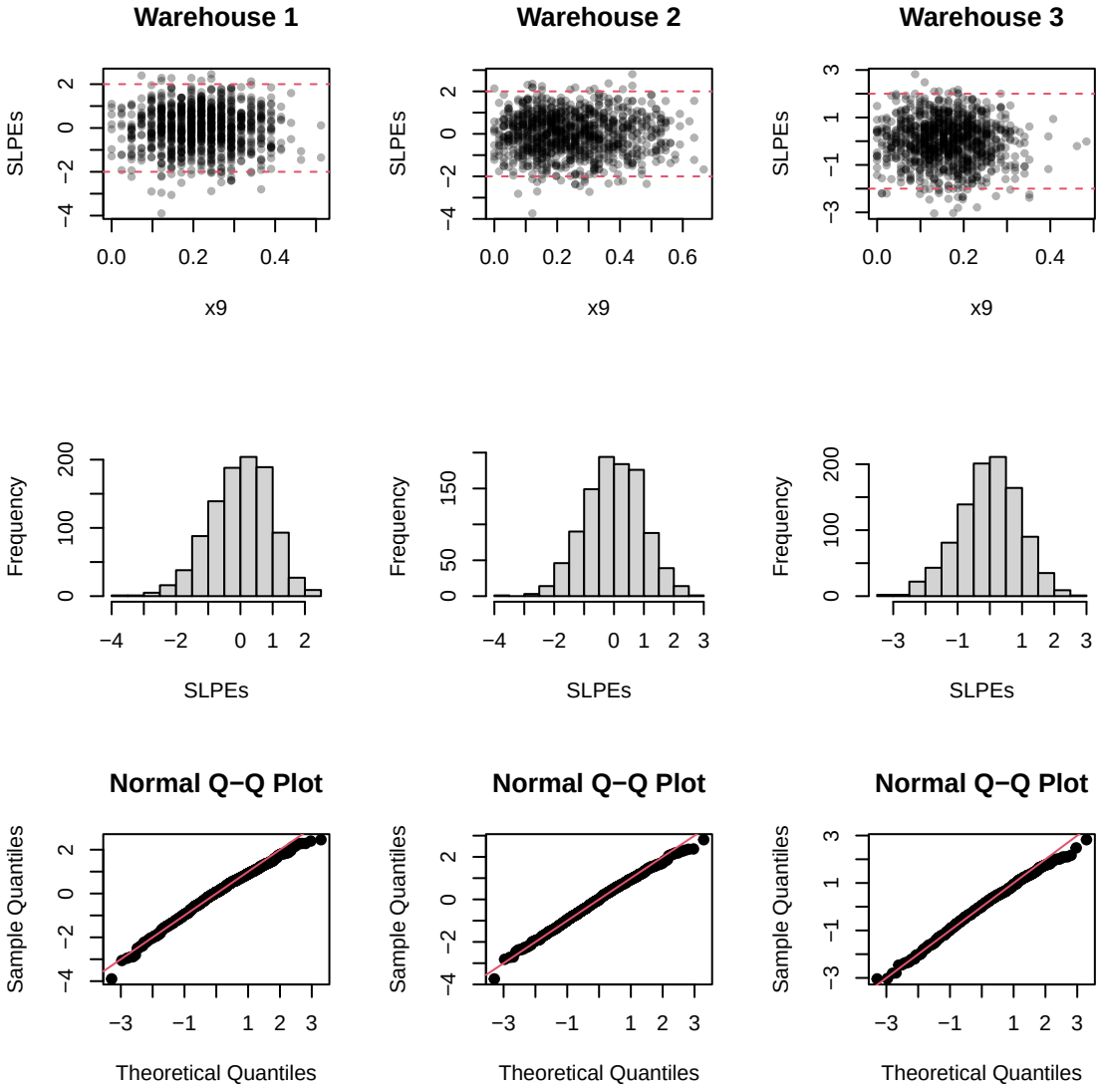

Figure 7.3: A suite of diagnostics based on SLPEs, organised in columns. Left column corresponds to Warehouse 1, the middle column to Warehouse 2 and the right column to Warehouse 3. The top three plots show the SLPEs plotted against x_9 after incorporating the prior mean function $\mu(\mathbf{x}) = \beta_0 + \beta_1 \log(x_9 + 10^{-8})$. Dashed red lines are at ± 2. The second row of plots are histograms of the SLPEs and the final row are Normal QQ plots, where the red line is the unit diagonal.

Warehouse	SLPE Interval	Expected Proportion	Observed Proportion
1	$(-2, 2)$	0.95	0.968
	$(-3, 3)$	0.997	0.998
2	$(-2, 2)$	0.95	0.967
	$(-3, 3)$	0.997	0.999
3	$(-2, 2)$	0.95	0.964
	$(-3, 3)$	0.997	0.998

Table 7.3: Comparing the expected proportion to the observed proportion of SLPEs to lie within given intervals, assuming Normality of the SLPEs. The observed proportions are those from the improved emulator.

i	1	2	3	4	5	6	7	8	9
$\hat{x}_i$	1	4	9	7	2	4	9	1	13
$\hat{x}_{i+9}$	57.03	39.96	42.07	28.25	2.29	40.25	46.62	1.32	57.5

Table 7.4: The estimated optimal decision reported on the decision scale rather than the $[0, 1]$ scale. The warehouse is absent from the table and this decision corresponds to Warehouse 1. The top row of variables corresponds to the number of spares of type i, the bottom row corresponds to the critical percentage for spares of type i.

7.5.4 *Ruling out decisions*

Using the improved emulator, we can find the design point, $\hat{\mathbf{x}}$, which maximises $\mathrm{E}\{U(\mathbf{x})\}$. In this case, $\hat{x}_{19} = 50$, so the decision belongs to Warehouse 1. The other elements of $\hat{\mathbf{x}}$ are given in Table 7.4. Our uncertainty about $U(\hat{\mathbf{x}})$ is characterised by

$$U(\hat{\mathbf{x}}) \sim \mathcal{N}(0.8314, 8.749 \times 10^{-5}) \tag{7.18}$$

Now, Warehouse 1 is in the NROY space as $I(\hat{\mathbf{x}}) = 0$. We constructed a random, uniform sample of decisions ($N = 10^5$) within each warehouse and found that 90.87% of decisions corresponding to Warehouse 1 satisfied $I(\mathbf{x}) < 3$ (95% CI: $(90.81, 90.93)$%) whereas only 1.4×10^{-3}% of decisions corresponding to Warehouse 2 are NROY (95% CI: $(6.52, 21.5) \times 10^{-3}$%). Credible intervals were found using a Binomial model for the number of observed NROY decisions within Warehouse i, M_i, from a sample of N_i random and uniformly distributed decisions. If the proportion of NROY decisions is α_i then *a priori* $\alpha_i \sim Beta(a_i, b_i)$ and the sampling model for the number of NROY decisions observed from a random sample is $M_i \mid \alpha_i \sim Binomial(N_i = 10^5, \alpha_i)$. It is well known that the distribution of

$\alpha_i \mid M_i$ is given by

$$\alpha_i \mid M_i \sim Beta(A_i, B_i) \tag{7.19}$$

$$A_i = a_i + \sum_{j=1}^{N_i} \mathbb{I}(I(\mathbf{x}_{i,j}) \geq 3) \tag{7.20}$$

$$B_i = b_i + \sum_{j=1}^{N_i} \mathbb{I}(I(\mathbf{x}_{i,j}) < 3) \tag{7.21}$$

where $\mathbf{x}_{i,j}$ is the jth candidate decision for Warehouse i. For each warehouse, we took a flat prior for α ($a_i = b_i = 1$) and then the credible intervals are found as the 95% highest posterior density intervals for $\alpha_i \mid M_i$.

Our random sample returned 0 decisions in Warehouse 3 that were NROY. To see if we could conclusively rule out Warehouse 3 as a sub-optimal decision, we minimised the implausibility function for Warehouse 3. To simplify the minimisation, we treated the discrete inputs as continuous. After running the optimiser 20 times, our minimum implausibility was $\min I(\mathbf{x}) = 7.36 > 3$. We are therefore satisfied that all decisions corresponding to Warehouse 3 will be ruled out.

The nature of the NROY space for Warehouse 1 is shown in Figure 7.4. We see that most decisions are green (NROY). The red, ruled-out decisions all seem to be those with very small values of x_9. For warehouse 2, only 14 out of 10^6 points from the large random sample were retained as NROY. We have therefore presented only the NROY samples in Figure 7.5. For Warehouse 2 we see that every $x_{10:18} < 0.5$ on the $(0, 1)$ scale. This suggests a larger warehouse size affords us more time to buy in spares.

We have greatly simplified the analysis for the DM by applying a post-hoc history matching inspired idea to the BayesOpt routine to find decisions which are consistent with the approximate maximiser and find those which are clearly suboptimal. If desired, this would be a sensible place to stop the analysis and report our findings to the DM. However, the HM literature has shown that further probing can be advantageous (Jackson *et al.*, 2018; Vernon *et al.*, 2022*a*).

7.5.5 *Constructing a wave* 2 *design*

We follow the convention of further investigation within the NROY space to refine the set of sensible decisions. To do so, we need a design for a wave 2 emulator. For

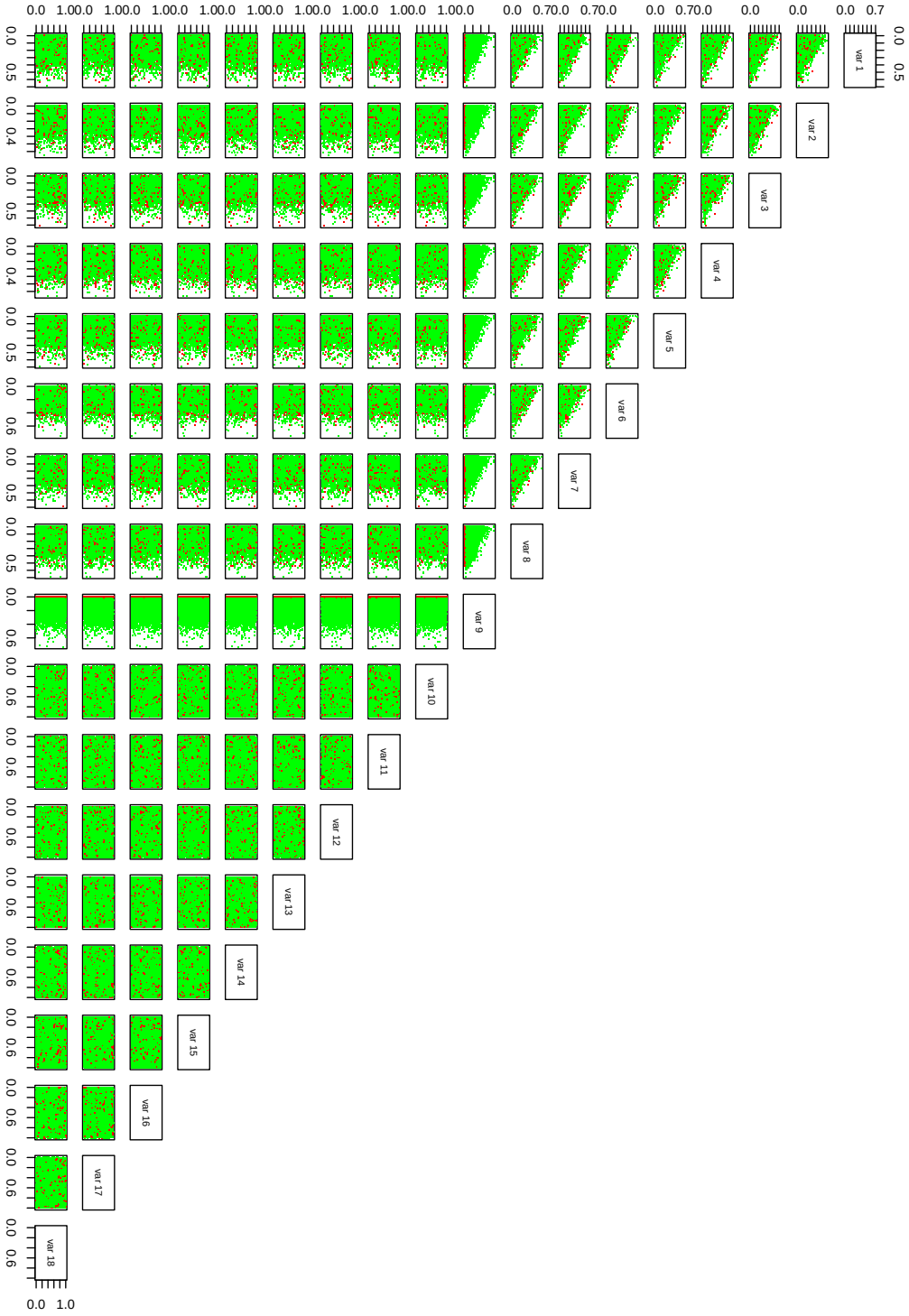

Figure 7.4: A large uniform sample of decisions corresponding to Warehouse 1. Green decisions are those that are NROY and red have been ruled out. Although x_1 was not explicitly included in the emulator, we have included it in this figure. Further note that the first 9 inputs have been jittered with $\mathcal{N}(0, 9 \times 10^{-6})$ noise to aid visualisation.

Figure 7.5: A collection of NROY decisions corresponding to Warehouse 2. Although x_1 was not explicitly included in the emulator, we have included it in this figure.

Warehouse 1, since the remaining NROY volume is quite large, simple rejection sampling will lead to a uniform design without much trouble. For Warehouse 2, the NROY volume is a small fraction of the original volume, consequently, rejection sampling is inefficient.

To overcome the inefficiency of rejection sampling, we employ an adapted version of the slice sampler presented by Andrianakis *et al.* (2017*a*). The adaptation addresses the discrete simplex aspect of the decision space and is presented in Algorithm 5. The sum to unity constraint on the simplex means one decision variable is determined, given the rest. This algorithm does not include any continuous-valued components of $\mathbf{x}$, however, since the algorithm is component-wise, we can first update the set of discrete components and then update the set of continuous components using Algorithm 3.

Algorithm 5 A single sweep of a component-wise slice sampler for generating NROY samples within the discrete simplex.

Require: An indicator function $\mathbb{I}(\mathbf{x} \in \mathcal{X}_{k+1})$ for identifying inputs which are NROY, an initial NROY point $\mathbf{x}_0 \in \mathcal{X}_{k+1}$ where $\mathbf{x}_0$ is an m dimensional parameter vector. A set of $q+1$ unique, equally spaced, values $\mathcal{S} = \{d_0, d_1, d_2, \ldots, d_{q-1}, d_q\}$ where $0 = d_0 < d_1 < \ldots < d_{q-1} < d_q = 1$.

$\mathbf{x}^* \leftarrow \mathbf{x}_0$
for $i = 1, 2, \ldots, m$ **do**
  $I^* \leftarrow 0$ ▷ Initialise an indicator variable
  while $I^* \neq 1$ **do**
    Compute $d_p = 1 - \sum_{j \neq i} x_j^*$
    Draw $x_i \sim \mathcal{U}\{d_0, d_1, d_2, \ldots, d_p\}$ ▷ Change the ith input
    $x_i^* \leftarrow x_i$
    $x_1^* \leftarrow 1 - \sum_{j=2}^{m} x_j^*$ ▷ Force sum to unity constraint
    $I^* \leftarrow \mathbb{I}(\mathbf{x}^* \in \mathcal{X}_{k+1})$ ▷ Evaluate the implausibility measure at the new input
  end while
end for
$\mathbf{x}_1 \leftarrow \mathbf{x}^*$
Return $\mathbf{x}_1$, the new NROY sample.

As with Algorithm 3, Algorithm 5 is not ergodic. However, as with Algorithm 3, we can adapt Algorithm 5 to become ergodic in a similar way to Algorithm 4: we essentially re-run the algorithm from a collection of randomly chosen starting points. We write this out explicitly in Algorithm 6.

For low-dimensional, continuous-valued simplices, uniform sampling typically gives reasonable coverage of the margins. However, as the dimension in-

Algorithm 6 An ergodic algorithm for simulating uniformly on of $\mathcal{S}_q^n \times [0,1]^p$.

Require: An indicator function $\mathbb{I}(\mathbf{x} \in \mathcal{X}_{k+1})$, a set of $q+1$ unique, equally spaced, values $\mathcal{S} = \{d_0, d_1, d_2, \ldots, d_{q-1}, d_q\}$ where $0 = d_0 < d_1 < \ldots < d_{q-1} < d_q = 1$, the number of continuous inputs, p, a number of replications, R, a required number of samples, $n = n' \times R$, and a function `slice_sampler_one_step`$(\mathbb{I}(\mathbf{x} \in \mathcal{X}_{k+1}), \mathbf{x}_0)$ which takes an NROY point $\mathbf{x}_0 \in \mathcal{X}_{k+1} \subseteq \mathcal{S}_q^n \times [0,1]^p$, and returns a new NROY point.

for $r = 1, 2, \ldots, R$ **do**
  $I_r \leftarrow 0$
  Initialise a matrix X_r with m columns and n' rows
  while $I_r \neq 1$ **do**
    $\mathbf{x}_{1,r} \leftarrow \mathcal{U}(\mathcal{S}_q^n \times [0,1]^p)$ ▷ Simulate uniformly from the cross product of $\mathcal{S}_q^n$ and $[0,1]^p$
    $I_r \leftarrow \mathbb{I}(\mathbf{x}1, r \in \mathcal{X}_{k+1})$
  end while
  for $i = 2, 3, \ldots, n'$ **do**
    $\mathbf{x}_{i,r} \leftarrow$ `slice_sampler_one_step`$(\mathbb{I}(\mathbf{x} \in \mathcal{X}_{k+1}), \mathbf{x}_{i-1,r})$
  end for
end for
Collect all X_r into a single matrix X, with m columns and n rows.
Return X, the collection of uniformly distributed NROY samples.

creases, we encounter a problem. It becomes increasingly difficult for large values of x_i to be sampled. Although we work with a discrete simplex, a continuous-valued simplex is easier to analyse and provides us with a good intuition for the marginal behaviour of a discrete simplex.

A uniform sample on the k dimensional, continuous simplex is equivalent to sampling from a $Dirichlet(1_{k+1})$ distribution, where 1_{k+1} is a length $(k+1)$ vector of 1s. A well known result states that if $\boldsymbol{X} \sim Dirichlet(\boldsymbol{\alpha})$ then the marginal distributions are given by $X_i \sim Beta(\alpha_i, \alpha_0 - \alpha_i)$ where $\alpha_0 = \sum_{j=1}^{k+1} \alpha_j$. Applying this result to the simplex means that a uniform sample on the k dimensional simplex has $Beta(1,k)$ margins. Now for any $b \in (0,1)$, $\Pr(X_i > b) \to 0$ as $k \to \infty$. Although we have stated an asymptotic result, the effects are present even for moderate values of k. In Figure 7.6 we see plots of $\Pr(X_i > b)$ for various choices of b over the range $k = 1, 2, \ldots, 13$. Note that we have $k = 8$ and thus $\Pr(X_i > 0.2) = 0.17$, $\Pr(X_i > 0.4) = 0.017$, $\Pr(X_i > 0.6) = 6.6 \times 10^{-4}$ and $\Pr(X_i > 0.8) = 2.6 \times 10^{-6}$. With emulators, we often have quite small samples sizes. Typically, designs have at most 1000 points. From a decision support

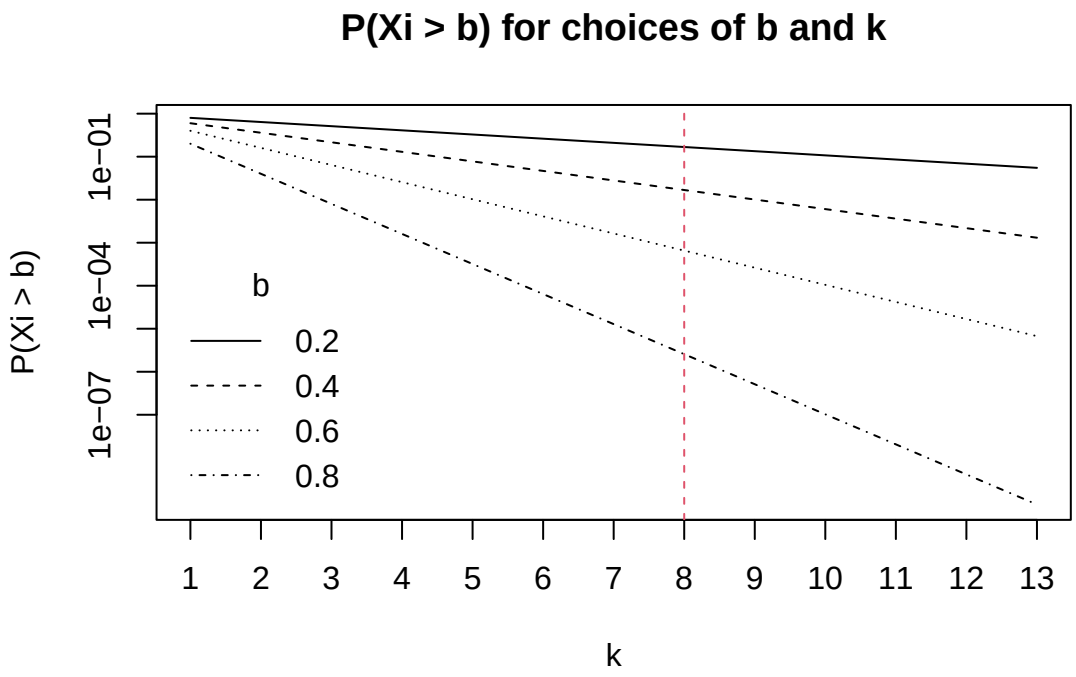

Figure 7.6: Lines showing how $\Pr(X_i > b)$ decreases with k, the dimension of the simplex. The vertical, red dashed line corresponds to $k = 8$, the dimension of the simplex in our wind farm example. The black lines of various types correspond to different values of b. Note that the y axis is on the logarithmic scale.

perspective, this is worrying. We want to consider a diverse set of decisions with a relatively small sample size. Therefore, we aim to construct a design which fills the space or, at least, targets areas of the decision space which are uncertain. This is especially important for Warehouse 1 as the current NROY space is almost all of the Warehouse 1 space.

One way to alleviate this problem is to force the design points to be far apart. This is a standard procedure. For example, maximin LHs achieve this goal by forcing the closest points to be as far away as possible, that is, maximising the minimum distance between design points. Our approach to 'filling the space' will be to construct an approximate ALM (Active Learning McKay, Equation (3.28)) design of size n. This exploits the fact that, in certain circumstances, posterior GP covariance matrices depend only on the locations of the design points and *not* the response $y = f(\mathbf{x})$. The posterior design matrix will usually rely on parameters estimated from the data therefore there is some dependence on y.

However, we can use estimated parameters from a previous experiment or specify the parameters. For a single warehouse, we proceed in the following way: first construct a large, uniform design within the NROY space of size $N >> n$. Call this design $\tilde{\mathcal{X}}$. Next, using assumed GP hyperparameters (from the previous wave, for example), find the point $\mathbf{x}^{(1)} \in \tilde{\mathcal{X}}$ which maximises $\text{Var}\{U(\mathbf{x})\}$. If a constant mean function is used, all points will have equal variance and so a random sample from $\tilde{\mathcal{X}}$ can initialise the procedure. We then remove $\mathbf{x}^{(1)}$ from $\tilde{\mathcal{X}}$ and find the next point with largest variance, call this point $\mathbf{x}^{(2)}$. We keep repeating this procedure until we have obtained $\mathbf{x}^{(n)}$. Then the set $X = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(n)}\}$ will be an approximate ALM design. Increasing N will likely lead to a better approximation to an exact ALM design. Finding $\arg\max_{\mathbf{x}\in\tilde{\mathcal{X}}} \text{Var}\{U(\mathbf{x})\}$ is achieved by computing $\text{Var}\{U(\mathbf{x})\}$ for all $\mathbf{x} \in \tilde{\mathcal{X}}$. This approach is useful in two ways. Firstly it allows us to construct a space-filling design, but it also allows us to perform post-processing of the slice sampler. Since the slice sampler is an MCMC method, the design is prone to auto-correlation. This post processing via ALM allows us to cope with a potentially autocorrelated MCMC chain. Rather than thinning the chain by a fixed factor (keeping every t-th iteration), we automatically choose points which are not close to each other.

We used this approach for both the remaining warehouses. The slice sampler was used to generate $N = 10^5$ candidate points for ALM designs of $n = 10^3$. For an ALM design, we first tried two different approaches. The first approach utilised our knowledge of $U(\mathbf{x})$ and used the GP hyperparameters of the emulator with mean function $h(\mathbf{x}) = \beta_0 + \beta_1 \log(x_9 + 10^{-8})$. Since the emulator was fitted using an empirical Bayes approach, the posterior distribution of each (β_0, β_1) pair is a direct consequence of the multivariate Normal equations. Therefore, we use the wave 1 posterior for the β parameters as the prior for the wave 2 emulators. This means that our ALM acquisition function, for warehouse i, is

$$\alpha_i(\mathbf{x}) = \sigma_i^2 + h(\mathbf{x})B_i h(\mathbf{x})^T - K_i(\mathbf{x}, X')\left\{K_i(X', X') + \lambda_i^2\right\}^{-1} K_i(X', \mathbf{x}) \tag{7.22}$$

where $i \in \{1, 2\}$ denotes the choice of warehouse, $K_i(\mathbf{x}, \mathbf{x}') = h(\mathbf{x})B_i h(\mathbf{x})^T + C_i(\mathbf{x}, \mathbf{x}')$ is the GP covariance function for warehouse i where $B_i = \text{Var}\{\beta^{(i)} \mid \mathcal{D}^{(i)}\}$, $\mathcal{D}^{(i)}$ is the set training data for warehouse i. $C_i(\mathbf{x}, \mathbf{x}')$ is the covariance function for the residuals from the mean function of warehouse i; we assume a squared exponential covariance function. As in Section 7.5.5, X' represents all locations

	Warehouse	
Design	1	2
ALMb	0.72	0.17
Uniform	0.35	0.15

Table 7.5: Values of $\min_{\mathbf{x},\mathbf{x}' \in X} \rho(\mathbf{x}, \mathbf{x}')$, where X is either the ALMb or the uniform (slice sampling) design.

already in the design. That is, points used to initialise the design as well as those that have been acquired. The second approach relied less on our knowledge of $U(\mathbf{x})$. The second design did not use knowledge of the mean function and thus the covariance function was simply a squared exponential. This is achieved by setting $B_i = 0$. We will call the first type of ALM design (with mean) ALMa, and the second (without mean) will be called ALMb. The hyperparameters of each squared exponential covariance function were taken from the wave 1 emulators.

The ALM post-processing of each MCMC chain took approximately 4.5 hours. This gave designs which led to x_i, for $i \in \{1, 2, \ldots, 9\}$, having wider margins than a random, uniform design of the same size. For this particular problem, the ALMa approach led to margins which were wider than uniform, as did the ALMb design.

The differences between the ALM designs and the uniform design for Warehouse 2 are more subtle. We suspect this less dramatic difference is due to the fact that the NROY volume within the Warehouse 2 space is a small subset of the original Warehouse 2 space. Figure 7.10 shows that, again, the range of each margin for the ALM designs are typically wider than the uniform design. Some of the uniform margins are wider then the corresponding ALM margins, but those that are wider are only wider by a small amount. In both cases, the difference between the ALMa and uniform design for x_9 is very large. This is because we used a non-constant mean function which depends on x_9. In particular, the functional form $\beta_0 + \beta_1 \log(x_9 + 10^{-8})$ has large in magnitude values for small x_9. Since the variance is essentially the square of this, $\text{Var}\{\beta_0 + \beta_1 \log(x_9 + 10^{-8})\}$ is maximised when $x_9 = 0$.

This led to the marginal density of x_9 having unsatisfactory properties. A comparison of the three designs is given in Figure 7.7. This figure shows the marginal density of x_9 under three design schemes. ALMa, the ALM design

which utilised knowledge of the mean function, is clearly not space-filling. For example, 985 out of the 1000 points are all exactly the same value, and there is a huge void from 0.2–0.8. The uniform design gives a more diverse set of candidate decisions for us to query, but we see that the largest value of x_9 is 0.585. ALMb provides a wide range of decisions (slightly narrower than ALMa) and lacks holes in the design. Since ALMb has a large range, but does not have any holes, we will use this design for our wave 2 experiment. Maximising minimum distance is a common design heuristic. We computed the minimum distances, $\min_{\mathbf{x},\mathbf{x}' \in X} \rho(\mathbf{x}, \mathbf{x}')$, where $\rho(\cdot, \cdot)$ is Euclidean distance, for the ALMb and uniform designs, which are given in Table 7.5. We see that the minimum distances for the ALMb designs for warehouses 1 and 2 are larger than the minimum distances for the corresponding uniform designs, thus the ALMb designs improves a standard design heuristic. We also want to quantify how much larger, on average, the range of the ALMb design is than the uniform design. An appropriate metric here will the geometric mean of the relative widths of the margins of one design to the other:

$$\delta_g(i) = \left(\prod_{j=2}^{18} \frac{\delta_{\text{ALMb},i,j}}{\delta_{\text{Uniform},i,j}} \right)^{\frac{1}{17}} \tag{7.23}$$

where $\delta_{D,i,j}$ denotes the range of margin j within warehouse i for design type D. Our design has 17 degrees of freedom and so we have removed the index $j = 1$ from the product. Now we have $\delta_g(1) = 1.13$, which indicates, in a multiplicative sense, that the ALMb design has margins which are, on average 13% wider than uniform. We also have $\delta_g(2) = 1.14$, so again, the ALMb design offers an improvement.

A similar effect is seen for x_9 within warehouse 2. We see (Figure 7.8) that, to a lesser extent than warehouse 1, that the ALMa design has a large clump of points at the lower end of the range. The uniform and ALMb designs appear to be similar (apart from a small shift in location), perhaps this is because the NROY volume for warehouse 2 is a fairly small fraction of the original space and thus, many extreme values may have been ruled out. Since we will be using the ALMb design, from hereon in we will refer to it as the ALM design.

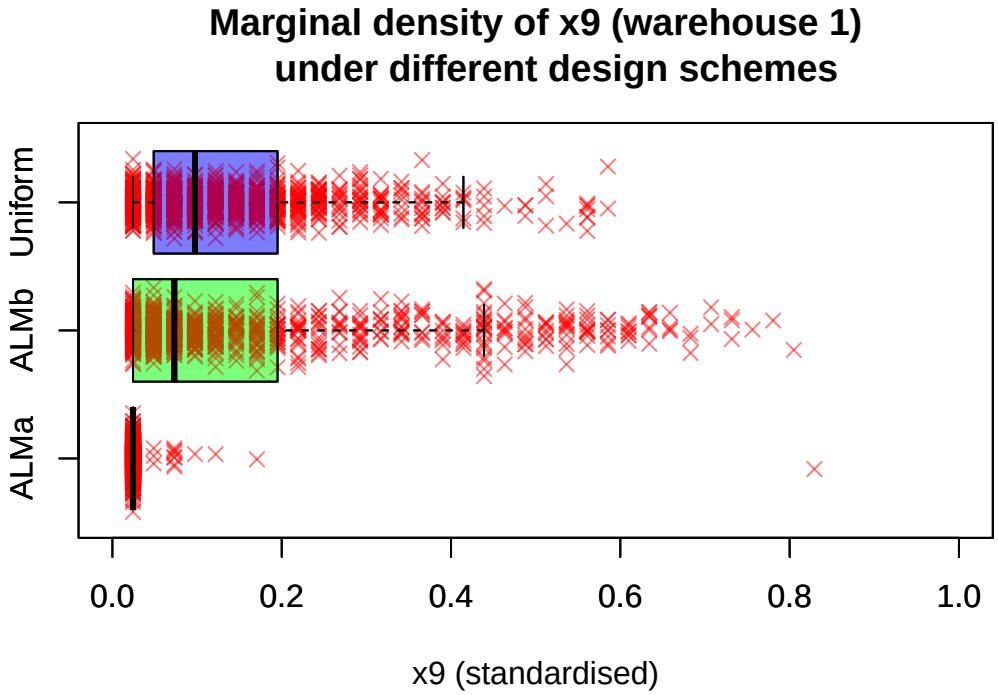

Figure 7.7: Boxplots depicting the marginal density of x_9, for warehouse 1, under three different design schemes. ALMa is the ALM design with the mean included, ALMb is the ALM design with no mean and Uniform is a uniform design generated by slice sampling. The red crosses denotes the values of x_9 from each design; these points are jittered vertically to aid visualisation.

7.5.6 *Wave* 2 *emulators*

To construct the wave 2 emulators we used the ALM designs discussed above. As with the wave 1 emulators the training data is generated by averaging over 30 realisations of $u(\mathbf{x})$ for each $\mathbf{x}$.

For the GP hyperparameters, we adopted the same priors as the wave 1 emulators, as these are relatively weak, but chosen to omit unrealistic values of the hyperparameters. We adopted the prior mean function $\mu(\mathbf{x}) = \beta_0^{(i)} + \beta_1^{(i)} \log(x_9 + 10^{-8})$. The prior for each $\boldsymbol{\beta}^{(i)}$ was chosen as $\boldsymbol{\beta}^{(i)} \sim \pi(\boldsymbol{\beta} \mid \mathcal{D}^{(i)}, \Theta_i)$. That is, the posterior distributions from wave 1 are adopted as wave 2 prior distributions. We also tried including a more detailed mean function (and thus adjusted the prior on each $\beta^{(i)}$), however, we found that leave-one-out RMSE (Equation (3.40)) and Score (Equation (3.41)) degraded when using a more detailed mean function. We therefore retain the functional form $\mu(\mathbf{x}) = \beta_0^{(i)} + \beta_1^{(i)} \log(x_9 + 10^{-8})$. The emulators

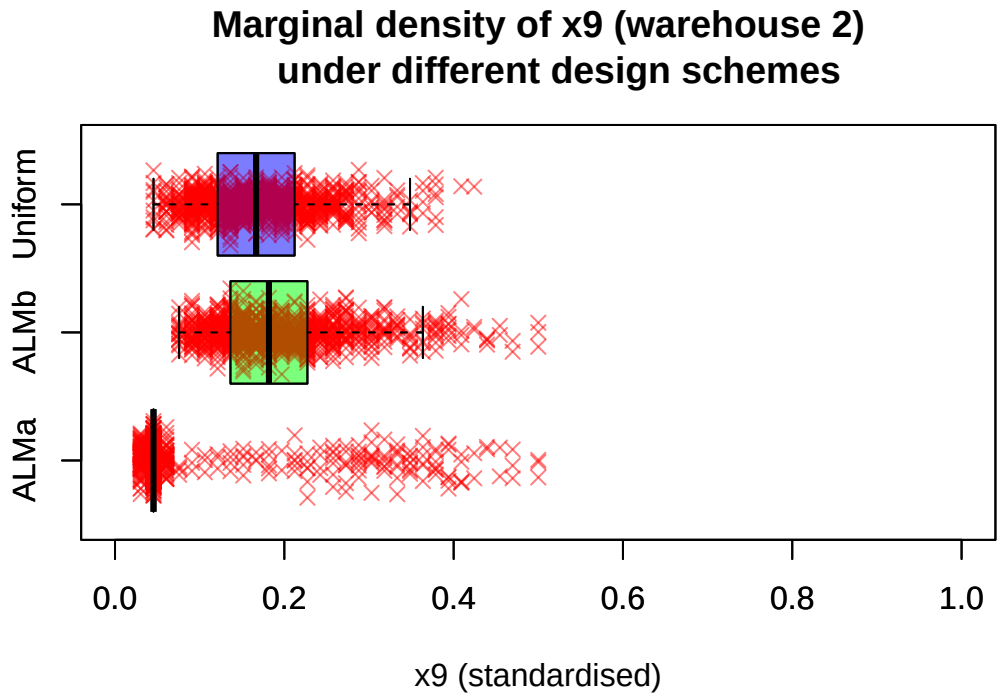

Figure 7.8: Boxplots depicting the marginal density of x_9, for warehouse 2, under three different design schemes. ALMa is the ALM design with the mean included, ALMb is the ALM design with no mean and Uniform is a uniform design generated by slice sampling. The red crosses denotes the values of x_9 from each design; these points are jittered to aid visualisation.

were fitted using our usual EB approach; assume a Normal likelihood, integrate out $\beta^{(i)}$ and then finding a MAP estimate of the GP covariance structure. Global diagnostics are given in Figure 7.11. We see there is some deviation from the Normality assumption. Namely, the left hand tail appears to be longer than the right hand tail. The excessive number of large, negative residuals implies there is some systematic under-prediction. This means the proportion of decisions ruled out may be larger than the proportion that would have been ruled out under a 'perfect' emulator. When considering the problem of optimisation under uncertainty, we would argue this is preferable to the opposite scenario of many under-predictions with large error, since systematic under-prediction would lead to the NROY space being smaller than it should be. That is, we would be ruling out decisions that should be retained. Since this approach is conservative and aims to address uncertainty, we would rather retain some sub-optimal decisions

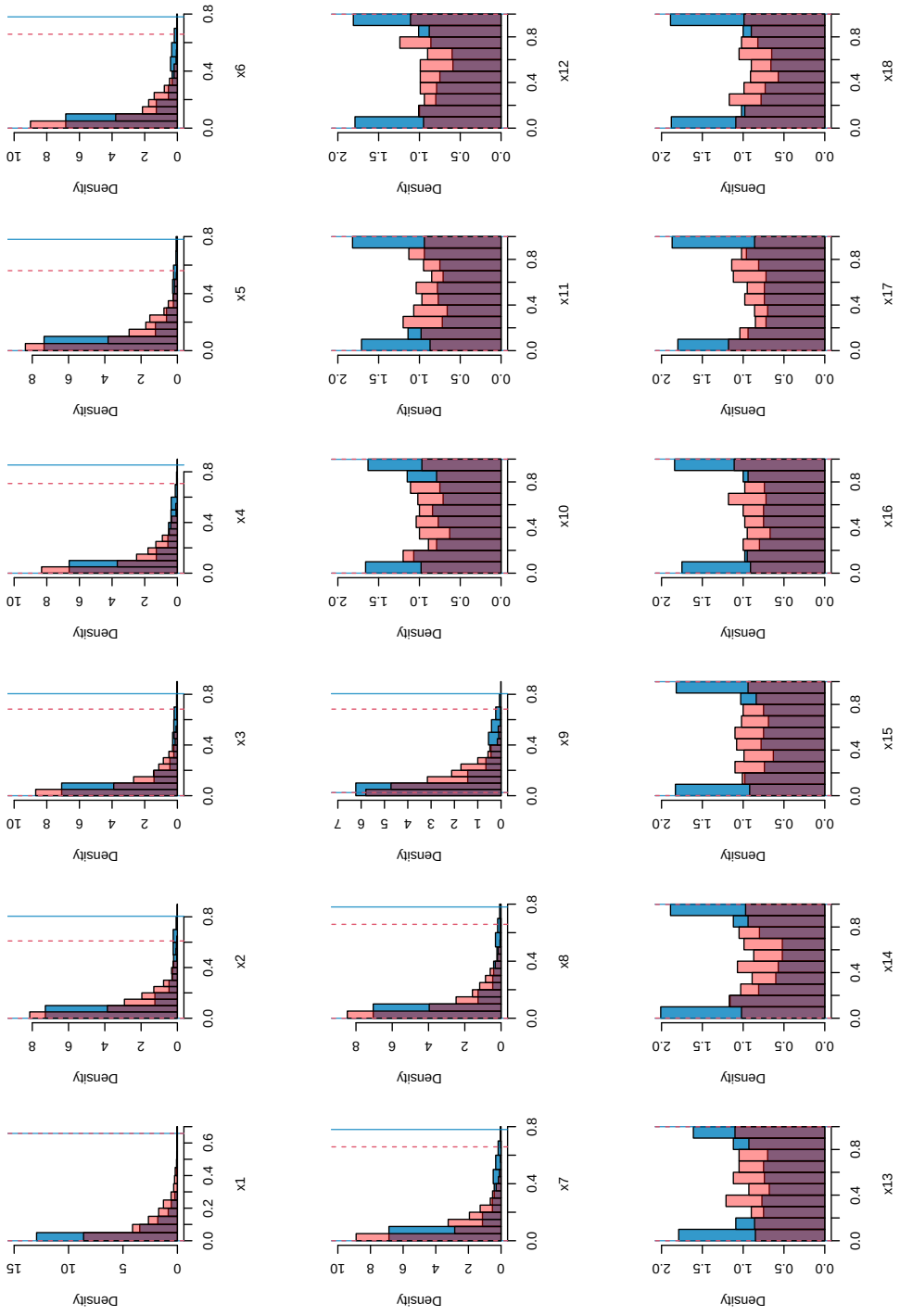

Figure 7.9: Comparing the marginal behaviour of the ALM and the uniform designs for a sample size $n = 1000$ for Warehouse 1. The ALM design is given as the blue histograms, whereas uniform are red. The limits of the ALM design are the solid blue lines, the dashed red lines are the limits of the uniform design.

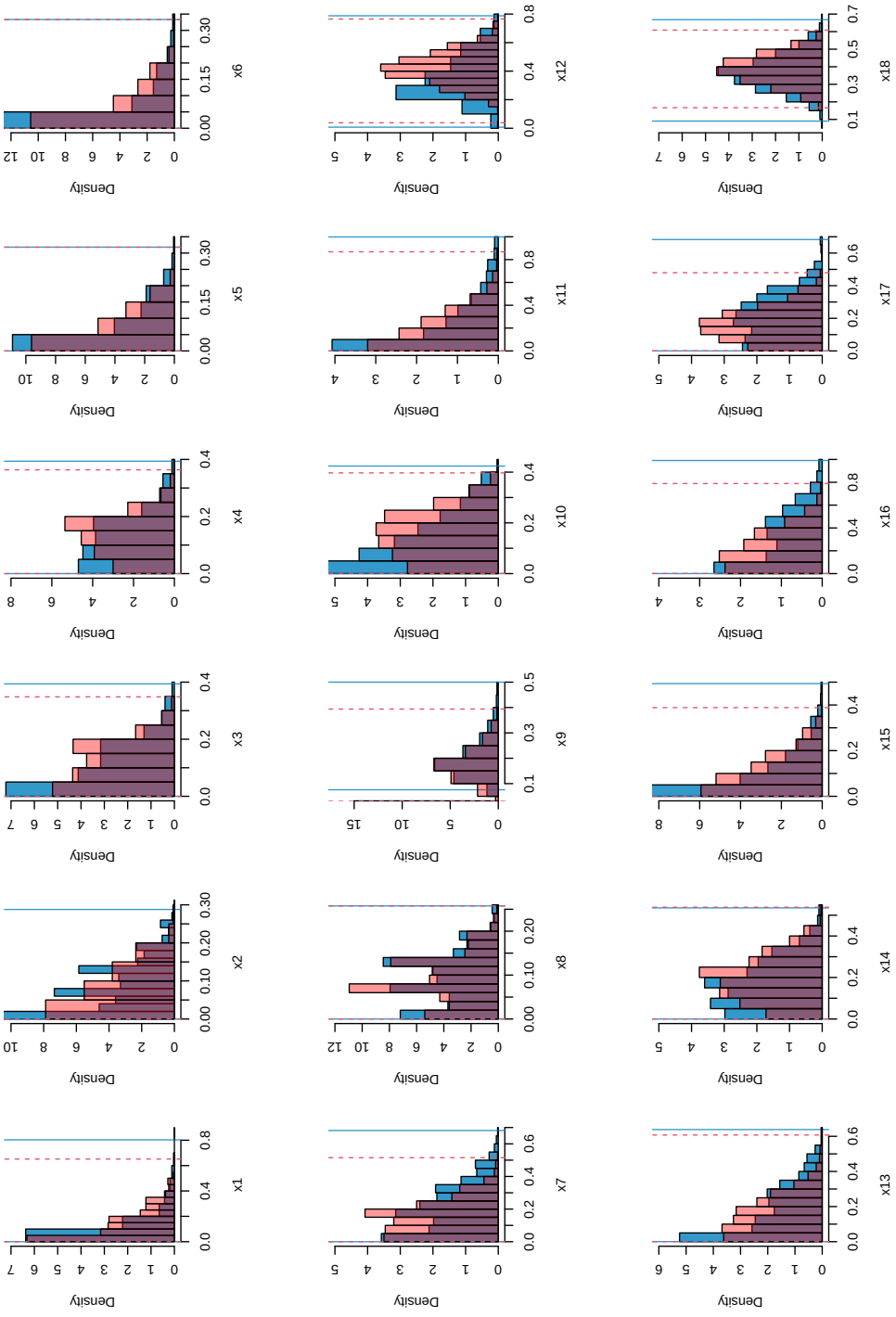

Figure 7.10: Comparing the marginal behaviour of the ALM and the uniform designs for a sample size $n = 1000$ for Warehouse 2. The ALM design is given as the blue histograms, whereas uniform are red. The limits of the ALM design are the solid blue lines, the dashed red lines are the limits of the uniform design.

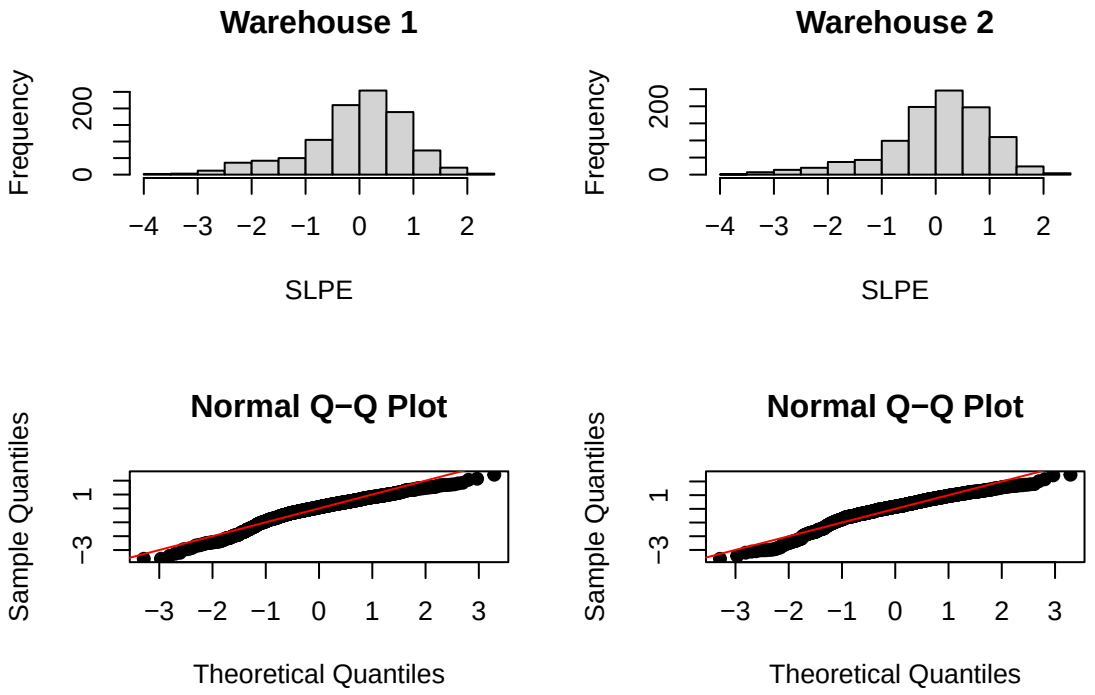

Figure 7.11: Global diagnostics for the wave 2 emulators. The left hand plots are for warehouse 1 and the right hand for warehouse 2. The top plots show histograms of LOO errors and the bottom plots are Normal QQ plots for the LOO errors.

(be inefficient) than rule out potentially good decisions (be ruthless). Plots of SLPEs against each inputs, for both warehouses, are given in Appendix A.3.

However, the mean SLPE for warehouse 1 is -0.0452 and the mean SLPE for warehouse 2 is 0.00757 which suggests that the SLPEs are not biased away from 0. Moreover, we see that the observed proportions of SLPEs within ± 2 and ± 3 are close to their theoretical proportions, assuming Normality (Table 7.6). The observed proportions are slightly under the theoretical proportions, however, there is agreement with Pukelsheim's 3σ rule so we can perform another round of iterative refocussing with confidence in the calibration of the approach.

7.5.7 *What is the maximiser*?

Although our philosophy towards decision making views the problem as having a set of sensible solutions, rather than one optimal solution, the HM inspired approach still needs a 'best' value to rule out decisions. If our simulator was

Warehouse	SLPE Interval	Expected Proportion	Observed Proportion
1	$(-2, 2)$	0.95	0.944
	$(-3, 3)$	0.997	0.995
2	$(-2, 2)$	0.95	0.954
	$(-3, 3)$	0.997	0.992

Table 7.6: Comparing the expected proportion to the observed proportion of SLPEs to lie within given intervals, assuming Normality of the SLPEs for the wave 2 emulators.

deterministic, we would simply take the largest value seen so far. However, we are working with a stochastic simulator. The uncertainty about the largest expected utility is an additional complication.

After the first wave, taking the largest expected utility, amongst the design points, is a natural choice for 'best'. However, the wave 2 emulators offer a new perspective of $U(\mathbf{x})$. The purpose of the wave 2 emulator is to explore a diverse set of decisions (via an ALM design) rather than to exploit knowledge of $U(\mathbf{x})$ to optimise it. Since the wave 2 emulator was not designed to optimise $U(\mathbf{x})$, should we trust its maximiser?

If we plug $\hat{\mathbf{x}}_1$ into the wave 2 emulator, we obtain $U(\hat{\mathbf{x}}_1) \sim \mathcal{N}(0.790, 2.54 \times 10^{-4})$ which is rather different from the wave 1 prediction. This is troublesome as it is not clear how to define our implausibility measure, which is reliant on the distribution of $U(\mathbf{x})$, and how it is correlated with $U(\mathbf{x}')$. However, $\frac{|0.790-0.831|}{\sqrt{2.54\times10^{-4}+8.75\times10^{-5}}} = 2.22 < 3$ so these two characterisations of uncertainty about $U(\hat{\mathbf{x}}_1)$ are in some sense compatible with each other. If we search the wave 2 design, the largest expected utility belongs to warehouse 1 and is characterised by

$$U(\hat{\mathbf{x}}_2) \sim \mathcal{N}(0.818, 2.47 \times 10^{-4}). \tag{7.24}$$

Now, $\mathrm{E}\{U(\hat{\mathbf{x}}_2)\}$ is much closer to $\mathrm{E}\{U(\hat{\mathbf{x}}_1)\}$; note that $|\frac{(0.831-0.818)}{\sqrt{8.75\times10^{-5}+2.47\times10^{-4}}}| = 0.725$, thus relative to their uncertainties, these two characterisations of the optimum value are fairly close. One solution to this problem is to perform sensitivity analysis: perform the analysis for both potential maximisers and see how different the results are. To perform the sensitivity analysis we will use $\hat{\mathbf{x}}_1$ and take our uncertainty about $\hat{\mathbf{x}}_1$ as characterised by the wave 1 emulator. Our uncertainty about $\hat{\mathbf{x}}_2$ will be characterised by the wave 2 emulator.

This means there will be two possible implausibility measures. They are, for

Wave	Best value	Warehouse	
Wave 2 only		1	2
	$\hat{\mathbf{x}}_1$	50.5%	99%
	$\hat{\mathbf{x}}_2$	90.8%	100%
Both waves		1	2
	$\hat{\mathbf{x}}_1$	45.89%	1.39×10^{-3}%
	$\hat{\mathbf{x}}_2$	82.51%	1.40×10^{-3}%

Table 7.7: Estimated proportion of the decision space remaining for each warehouse. The top half of the table is the reduction in space due to the wave 2 emulators, having already reduced the space by wave 1 emulators; the bottom half is the total reduction in NROY space from both waves of emulation.

$\mathbf{x} \in \mathcal{X}_2$

$$I_{2,1}(\mathbf{x}) = \frac{\mathrm{E}^{(1)}\{U(\hat{\mathbf{x}}_1)\} - \mathrm{E}^{(2)}\{U(\mathbf{x})\}}{\sqrt{\mathrm{Var}^{(1)}\{U(\hat{\mathbf{x}})\} + \mathrm{Var}^{(2)}\{U(\mathbf{x})\}}} \tag{7.25}$$

$$I_{2,2}(\mathbf{x}) = \frac{\mathrm{E}^{(2)}\{U(\hat{\mathbf{x}}_2)\} - \mathrm{E}^{(2)}\{U(\mathbf{x})\}}{\sqrt{\mathrm{Var}^{(2)}\{U(\hat{\mathbf{x}}) - U(\mathbf{x})\}}} \tag{7.26}$$

where the (j) superscript denotes the expectation/variance at the jth wave of emulation. The i, j subscript denotes that this is the implausibility at wave i and we are using $\hat{\mathbf{x}}_j$ as the 'reference' value for HM.

7.5.8 *Second round of refocussing*

To assess the reduction in the NROY space we first use $I_{2,1}(\mathbf{x})$ as an implausibility measure. We will then use $I_{2,2}(\mathbf{x})$. Both measures will use the cut off of 3 to determine which points have been ruled out. To estimate the reduction in the NROY space (compared to the previous wave) we will use a random, uniform sample of 1000 points which were deemed not implausible after the first wave of emulation. The reduction in NROY space for each warehouse, using the different implausibility measures, is given in Table 7.7. Using $\hat{\mathbf{x}}_2$ as the maximiser leaves the NROY space essentially unchanged; it offers no reduction whatsoever in the space of warehouse 2 and removes under 10% of the space for warehouse 1. When using $\hat{\mathbf{x}}_1$ as the maximiser, we obtain a modest reduction in NROY space for warehouse 1 and a small reduction in the space for warehouse 2.

We will use $\hat{\mathbf{x}}_2$ as the maximiser since it represents our current understanding

of $U(\mathbf{x})$. This does not reduce the NROY space, which is somewhat frustrating. However, we should not to rule out decisions solely for the sake of ruling out decisions. This poses a question; since $\mathrm{E}\{U(\hat{\mathbf{x}})\}$ has decreased, and $\mathrm{Var}\{U(\hat{\mathbf{x}})\}$ has increased, should we go back a wave and re-introduce decisions that were ruled out? We will not do this, but, Baker (2021) considers the notion of a flexible NROY space and shows that allowing the NROY region to shrink as well as grow protects against flaws in emulators from early waves. Baker (2021) shows, for a small set of simple examples, that the flexible approach can be advantageous in finding the 'correct' NROY space.

7.5.9 *Termination of iterative refocussing*

After performing these two waves of analysis, we have used, approximately, a total of 22 cores $\times$ 5 training rounds $\times$ 6 days $\approx$ 1.8 years of CPU time on generating training data for the emulators. This is based on the assumption that running Athena at 1000 inputs, each replicated 30 times, will take 6 days (over 22 cores). Training runs of Athena for this analysis took in the region of 5-7 days, therefore 6 days is a reasonable approximation. There is also the issue of queuing on HPC facilities which can be of the order of several days during busy periods. In terms of wall-clock time, we used a total of around 14 days computing time. Another round of emulation would take approximately another 6 days in training time. None of our calculations have included the human time required to perform analyses, the computational cost of fitting and validating of emulators or the computational cost of a design. For this reason, combined with the very small reduction in the NROY space we now terminate our iterative procedure.

Since we have terminated our procedure, we can calculate what proportion of decisions are NROY compared to the original set. This quantity is given by

$$\begin{aligned} |\mathcal{X}_{\mathrm{NROY}}| &= \frac{0.8251 \times \binom{49}{8} + 1.39 \times 10^{-6} \times \binom{74}{8} + 0 \times \binom{99}{8}}{\binom{49}{8} + \binom{74}{8} + \binom{99}{8}} \\ &= \frac{3.72 \times 10^{8}}{1.87 \times 10^{11}} \\ &= 1.99 \times 10^{-3} \end{aligned}$$

which is a reasonable reduction in the size of the NROY volume; around 0.2% of the original decisions have been deemed NROY. Note that we have completely

ruled out warehouse 3, and the number of decisions remaining from warehouse 2 is tiny. Most of our remaining decisions correspond to warehouse 1. By ruling out many decisions, we have greatly simplified the problem for the DM.

7.6 *Incorporating the DM: evaluating the consequences of decisions*

The final aspect of our analysis contains two parts. First we will provide the DM with a small set of NROY decisions. Displaying the consequences of a large set may overwhelm the DM and thus they may not be able to adequately assess the relative merits of each decision. These are decisions which are not clearly worse than $\hat{\mathbf{x}}_2$. For each decision put forward to the DM, we will present the decision alongside the consequences of that decision. Recall, the consequences are the choice of warehouse, the availability time series and the restoration policy. We will present many replicates of the availability times series to the DM to communicate that this is an uncertain consequence.

The DM then has two tasks. The first task is to provide feedback on the decision space. If the DM finds believes that the given decisions are not consistent with their preferences, we must go back to the start of our analysis and perform the entire task again; re-elicit attributes, consequences, the functional form and parameters of the utility function (Smith, 2010). However, this is very computationally expensive so, if the DM does reject all of the decisions shown, we could first present a new sample of decisions. Note that this step is computationally expensive because one consequence relies on running the Athena simulator. However around 30 replicates for a small handful of decisions can be computed in about an hour on the Rocket HPC facility, or overnight on a modern desktop computer. If the DM is not satisfied, it may be worth trying to optimise a single consequence. For example, if the DM was not happy with the availability trajectories we have produced from the NROY space, it would be worth investigating if there are *any* availability trajectories within the initial decision space that the DM is happy with. We should also remind ourselves that decision analysis allows us to make the best decision, however, it is entirely possible that there are no 'good' decisions, but rather, we must find the least worst option. For example, suppose you must make a long journey. You can either take the train, which takes 12 hours and

involves multiple connections, or you can take a 1 hour flight. Further suppose you are environmentally conscious, and are aware that taking a flight is much more harmful to the environment than the train. Both decisions have obvious disadvantages and you might be unhappy with both prospects, but there will still be an optimal decision.

If the DM finds one or more solutions satisfactory, they must then decide which decision to make. This decision should solely be that of the DM.

7.6.1 *Presenting NROY decisions to the DM*

Since no real DM is involved in the analysis, we chose a set of 12 decisions to present to our fictitious DM; 6 from each warehouse. One was chosen as $\hat{\mathbf{x}}_2$, another was chosen as the decision which maximised the wave 2, warehouse 2 emulator and a further 5 random decisions from each warehouse within the NROY set. The values of $x_1 - x_{18}$ for each of these decisions is given in Table 7.8. The first 6 decisions corresponds to warehouse 1 decisions, the final 6 to warehouse 2. The 6th decision is $\hat{\mathbf{x}}_2$ and the 12th decision is the decision which maximised the wave 2, warehouse 2 emulator. If a DM were to be involved, they may request the analyst to further investigate particular decisions, provided they are NROY.

We ran the Athena simulator 300 times at each of these input configurations to give the DM a detailed picture of the availability trajectories one could see if we made any of these 12 decisions. Allowing the DM to choose one of these decisions for themselves has the advantage of allowing the DM to bring in any additional information that was not given to the analyst. This ensures that the DM's preferences are honoured if the elicited utility function is a misrepresentation of their beliefs. In Figure 7.12 we see that the distribution of availability trajectories is very similar for the 12 examples. By eye, it is essentially impossible to tell them apart. Each set of trajectories seems to have two modes. The 'main' mode has the availability trajectories typically sitting in the region of $[0.9, 1]$ for all Time $\in [0, 5]$ years. The 'minor' mode has a prominent dip in availability in for when Time is in the region of $[2, 3]$ years; the availability drops to a minimum which is typically in the range $[0.6, 0.7]$.

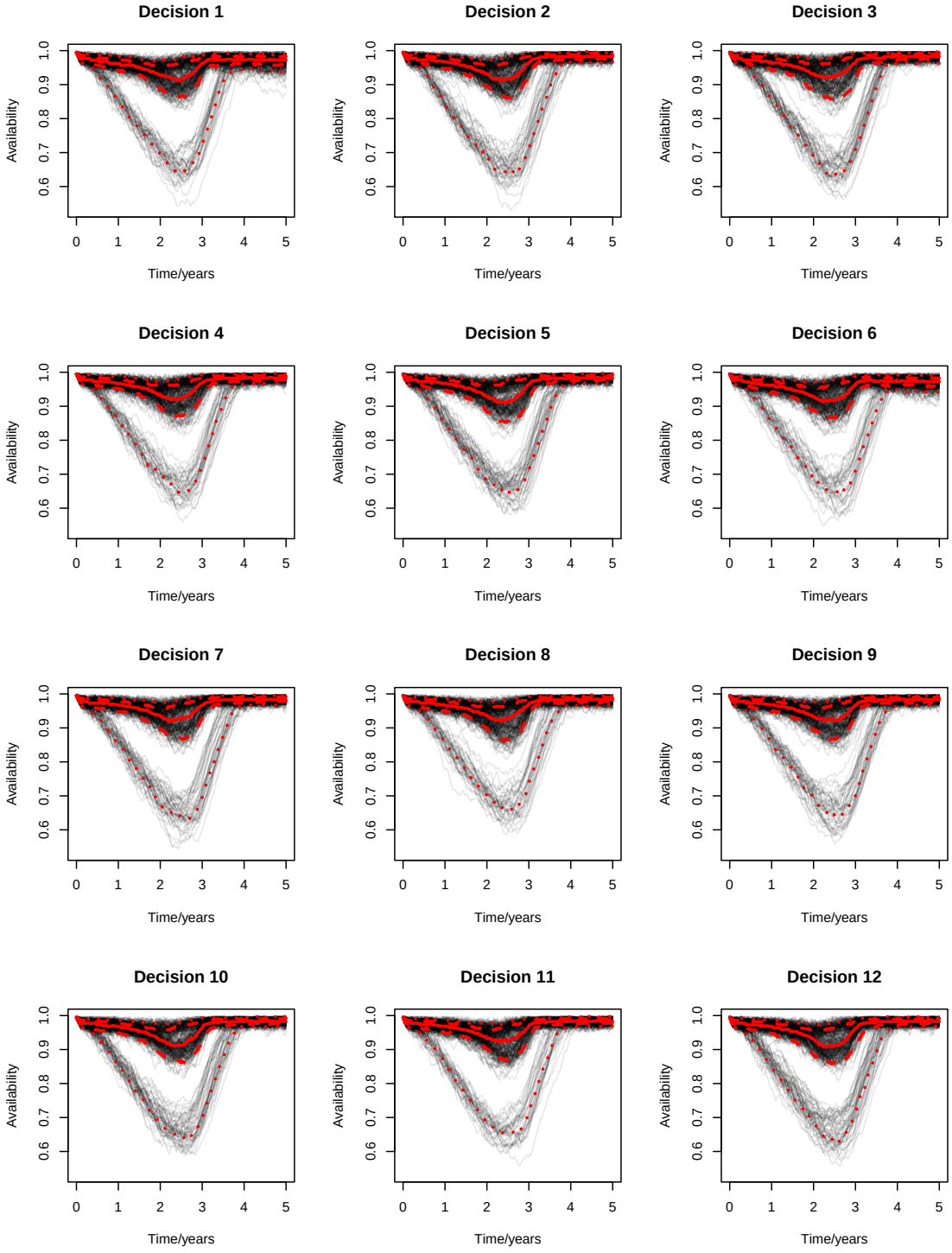

Figure 7.12: Each sub-figure shows 300 availability trajectories (grey lines) for each of the decisions given in Table 7.2. The solid red line represents the median availability trajectory for a decision, the dashed red lines correspond to 20% and 80% quantiles for the availability and the dotted red lines represent 5% and 95% quantiles for the availability. Quantiles are all point-wise.

	Decision variable (x_i)																	
Decisions	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	x_{16}	x_{17}	x_{18}
1	5	5	7	2	14	2	11	2	2	31.89	80.33	33.42	23.32	40.41	26.62	63.68	24.71	4.69
2	3	6	2	2	2	7	12	2	14	98.36	43.13	38.82	54.03	91.45	42.74	22.9	29.08	78.51
3	2	1	3	12	8	3	9	2	10	14.62	19.98	50.37	31.71	31.12	63.65	12.91	68.23	22.01
4	9	2	6	4	7	14	1	2	5	89.01	66.72	83.35	15.17	59.15	17.19	46.21	25.14	67.15
5	5	5	16	1	4	7	3	2	7	62.27	5.71	22.87	45.19	21.53	79.31	57.25	84.51	46.01
6	2	5	3	15	6	1	7	9	2	28.93	3.76	1.87	22.3	8.44	1.16	53.27	43.18	20.63
7	28	2	4	1	7	1	14	7	11	14.72	13.15	21.49	20.59	9.64	19.73	85.27	29.34	50.52
8	13	4	20	9	2	11	8	3	5	17.9	20.98	60.65	12.2	30.72	3.19	10.14	8.88	43.05
9	4	2	1	8	4	3	32	6	15	1.85	18.93	22.12	3.48	29.25	6.82	23.69	30.68	39.73
10	6	5	4	13	6	2	21	7	11	29.26	20.3	29.9	4.56	28.15	27	2.77	10.96	32.88
11	8	10	8	7	2	4	9	10	17	19.06	30.82	18.7	6.85	11.36	4.63	8.44	27.75	28.81
12	5	7	10	10	9	5	14	1	14	13.04	6.13	39.9	28.37	12.88	20.06	23.68	17.87	33.22

Table 7.8: The values of x_1–x_{18} for each of the 12 decisions presented to the DM. Note that the first 6 decisions correspond to warehouse 1 ($x_{19} = 50$) and the final 6 corresponds to warehouse 2 ($x_{19} = 75$).

7.6.2 *Retrospective validation of assumptions*

We now wish to (retrospectively) verify three aspects of our analysis. Our maximin design was constructed from, essentially, one long run of **??**, rather than the ergodic variant — we need to check that the Markov chain is irreducible (that is, not ergodic). Next, we relied on Pukelsheim's 3σ rule to determine which decisions are (not) consistent with the (estimated) optimal decision. We can only use Pukelsheim's 3σ rule when $U(\mathbf{x})$ has a continuous, unimodal distribution; we should verify this assumption. Finally, we should check that decisions from the final NROY set are indeed consistent with our findings.

Checking the Markov Chain is irreducible

To verify that the Markov chain used to generate a large candidate design is indeed irreducible, we implement Algorithm 6 with $R = 30$ repeated runs, $n' = 1000$ within-chain runs for a total sample size of $N = 30000$. This is performed for each of warehouses 1 and 2.

To investigate as to whether or not the chain is reducible, we first performed principal components analysis to obtain a low-dimensional visualisation of the NROY samples. The first two principal components are shown for the two samples of $N = 30000$ NROY points generated from Algorithm 6 are shown in Figure 7.13.

In each of the subfigures of Figure 7.13, there is no evidence to suggest that the NROY region is a union of disconnected regions. The red triangles — the points from just one of the parallel runs — have a similar joint distribution to all other samples. Further, the samples all appear to come from one connected region, rather than two or more disconnected sub-regions. Note that these samples are uniform across the NROY region and not processed to form (e.g.) a maximin design.

Pukelsheim's 3σ rule

Our history matching procedure has relied on Pukelsheim's 3σ rule which states that $P(|U(\mathbf{x}) - \mu| > 3\sigma) < 0.05$ under two assumptions. Firstly, $U(\mathbf{x})$ must be continuous. This is satisfied as $u(\mathbf{x})$ is a smooth function of $\mathbf{x}$ and availability is a continuous-valued random quantity. The second assumption is that $U(\mathbf{x})$ must be unimodal. The sub-figures within Figure 7.12 all show that the availability distri-

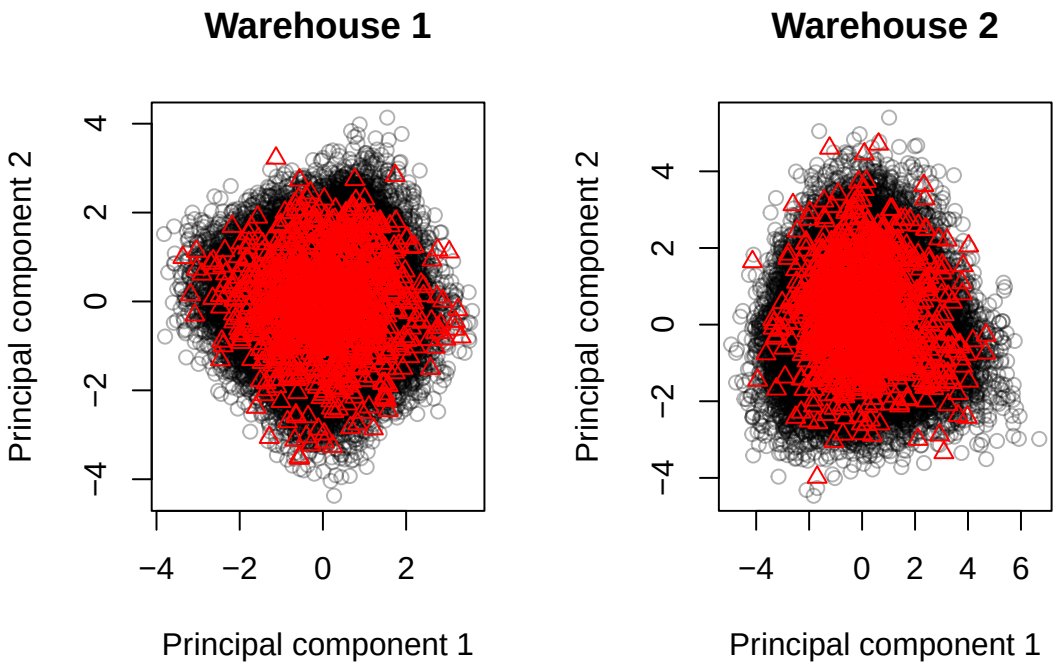

Figure 7.13: For each warehouse, the figures show the first two principal components of a large collection of uniform NROY samples. The black circles are points from all chains, whereas the red triangles are a all the points from just one of the parallel runs. We see in both cases, that there is no evidence of a disconnected NROY region.

bution has at least two modes. By the CLT, we would expect our training outputs, $y(\mathbf{x})$, to be approximately Normal (and thus unimodal) as they are constructed via sample means. However, it is important to remember that the CLT is only an asymptotic result and offers no general guarantee for samples sizes $n < \infty$. To verify that the training data are unimodal quantities, we perform a simple bootstrap procedure to estimate the sampling distribution of $y(\mathbf{x}) = (1/30)\sum_{i=1}^{30} u(\mathbf{x})_i$. The bootstrap distributions shown in Figure 7.14 all support the assumption that $y(\mathbf{x})$ is unimodal, even though $u(\mathbf{x}_i)$ has a multimodal distribution. Note that, if $y(\mathbf{x})$ was still multimodal, the methodology we used would not work. To remedy this, we can increase the sample size which $y(\mathbf{x})$ are based on. This will mean that training data are more expensive to obtain; the benefit is that we can avoid many awkward probability distributions and thus utilise relatively simple and robust methods to quantity uncertainty about $U(\mathbf{x})$.

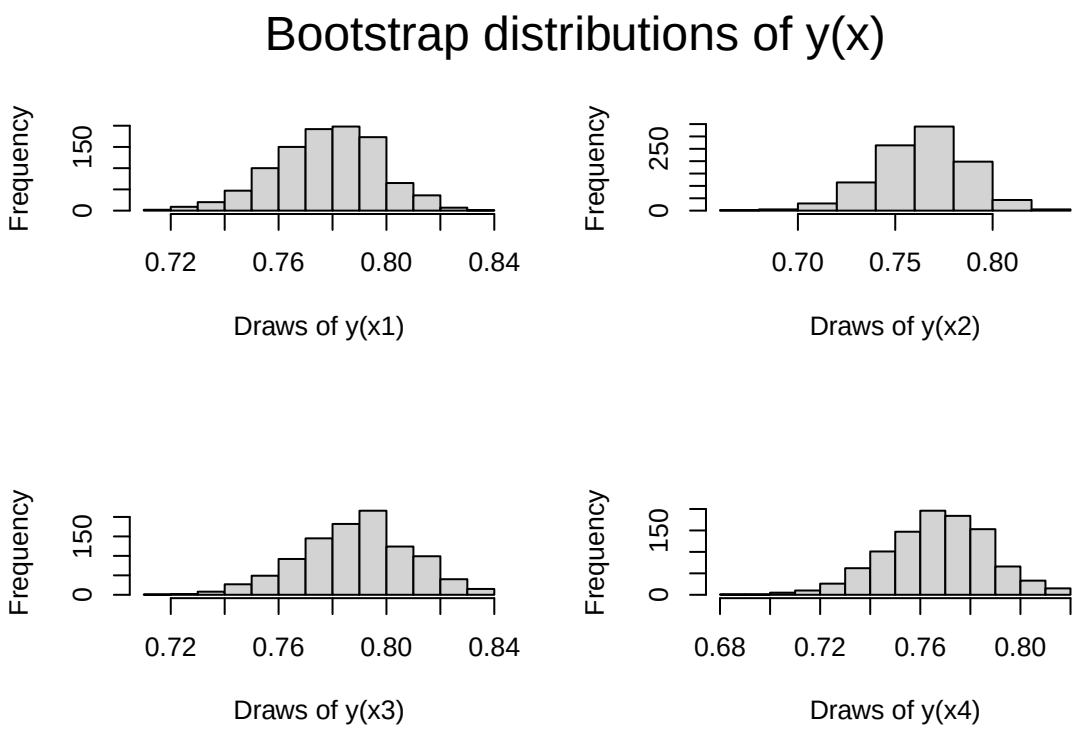

Figure 7.14: Bootstrap distributions for 4 randomly chosen inputs, $\mathbf{x}_i$, $i = 1, 2, 3, 4$, from the wave 2 design. The bootstrap distribution for $y(\mathbf{x}_i)$ is based on 1000 values of $y(\mathbf{x}_i)$.

Consistency of final NROY volume

To check that the final NROY volume is indeed consistent with our uncertainty about the optimal decision, we can re-use the large samples presented in Figure 7.12. In particular, the 300 samples shown in each sub-figure are random draws which are independent of any emulator training data and the emulators themselves. The means and standard errors implied by these samples offer an independent perspective of $U(\mathbf{x})$ for each $\mathbf{x}$. Comparing these to the emulator's characterisation of $U(\hat{\mathbf{x}}_2)$ allows us to assess if the apparently NROY decisions should have been ruled out. Table 7.9 shows the mean and standard error for each of the 6 decisions given in Figure 7.12. Our implausibility measure for this final aspect of the analysis will be

$$I^*(\mathbf{x}) = \frac{\mathrm{E}\left\{U(\hat{\mathbf{x}}_1)\right\} - \hat{U}(\mathbf{x}_i)}{\sqrt{\mathrm{Var}\left\{U(\hat{\mathbf{x}}_1)\right\} + \widehat{\mathrm{s.e.}}\{\hat{U}(\mathbf{x}_i)\}^2}} \tag{7.27}$$

Decision	$\widehat{U}(\mathbf{x})$	$\widehat{\text{s.e.}}(\mathbf{x})$	$I^*(\mathbf{x})$
1	0.788	5.592×10^{-3}	1.801
2	0.786	6.138×10^{-3}	1.897
3	0.806	6.254×10^{-3}	0.739
4	0.797	5.832×10^{-3}	1.281
5	0.793	6.326×10^{-3}	1.484
6	0.800	5.414×10^{-3}	1.071
7	0.789	6.244×10^{-3}	1.728
8	0.801	5.657×10^{-3}	1.053
9	0.799	5.978×10^{-3}	1.139
10	0.791	6.406×10^{-3}	1.614
11	0.807	5.708×10^{-3}	0.693
12	0.788	6.317×10^{-3}	1.797

Table 7.9: Means ($\widehat{U}$) and standard errors ($\widehat{\text{s.e.}}$) for the expected utility of the 12 decisions corresponding to Figure 7.12 and Table 7.8, as well as an implausibility measure (I^*) for each decision.

where $\widehat{\text{s.e.}}\{\hat{U}(\mathbf{x}_i)\}$ is the estimated standard error of $\hat{U}(\mathbf{x}_i)$. Recall that $\mathbf{x}_6 = \hat{\mathbf{x}}_2$. In this case we see that 0 decisions are ruled out under I^*. Under Pukelsheim's 3σ rule we would expect to retain *at least* 90% of decisions after two well-calibrated waves. Assuming Normality (and a cut off of 3) we would expect to retain around 98% of decisions after two well-calibrated waves. This sample size is small but 100% is close enough to 98%, and clearly 100% > 90%, we therefore judge that our ruling out procedure is well-calibrated.

7.7 Discussion

We have presented a history matching inspired procedure for decision support with the stochastic Athena simulator. Our approach drew on established ideas, such as the UCB BayesOpt heuristic (Srinivas *et al.*, 2009) and history-matching inspired decision support for deterministic simulators (Owen *et al.*, 2020) but relied on novel applications of these ideas to address the challenges within our applied problem of decision support under uncertainty for the Athena simulator.

We first provided an illustrative elicitation of a utility function. We drew on key attributes to the problem, but we stress the elicitation will always be subjective; every DM has their own utility function.

We then proposed unifying two established approaches. We began by performing three independent rounds of BayesOpt to optimise the choices of x_1–x_{18} then used techniques from the HM literature to communicate uncertainty about the optimal decision. Our HM inspired approach required a novel implausibility measure which accounts for dependence across the decision space.

We then considered the design for our decision support exercise. Since a subset of our inputs were a discrete simplex, which is non-standard in the computer experiments literature, this posed some interesting challenges. BayesOpt allowed us to side-step the problem of design for a wave 1 emulator since it automatically chooses where to run the simulator. The design for a wave 2 emulator was more challenging as we had to adhere to the constraints of the decision task a whilst constructing a design within the NROY region that provided a diverse set of decisions. Sequential design offered a promising approach again. Our approximate ALM design allowed us to construct a design which was more space-filling than a uniform design. We believe sequential designs are a promising option for non-standard input spaces.

We then constructed a new pair of emulators to further explore the NROY space. We performed a sensitivity analysis based on the choice of possible maxima. One maximum led to a negligible reduction in the NROY space and the other led to a small reduction in the NROY space. In general, we would advise using the results of the most recent emulator to choose the maximiser (when $y(\mathbf{x})$ is stochastic). This would protect against $U(\hat{\mathbf{x}})$ being an over-estimate, which may rule out too many decisions. In the deterministic case, it would be natural to choose the largest observed value across all waves.

Next we presented the DM with a set of sensible decisions that could be taken. We presented decisions from both warehouses and from diverse decisions within each warehouse.

Finally, we performed some post-hoc checks to verify some assumptions, as well as checking whether the proportion of incorrectly ruled out decisions is consistent with Pukelsheim's 3σ rule. Our small sample of decisions suggested that we did not rule out too many decisions, although a much larger sample would be needed to make a conclusive statement.

If we were to perform the analysis again, with the assistance of a DM, we could prevent ruling out too many decisions by incorporating model discrepancy. It would be possible to estimate model discrepancy via the elicitation procedure.

When fitting the utility function to the DM's beliefs, we could use the 'over-fitting' technique in which we elicit more preferences than necessary. We can then fit a functional form to the elicited quantities via least squares. The sum of squares will be positive and can be interpreted as a variance. This sum of squares can be used as a *minimum* model discrepancy. Nonetheless, even if we did rule out too many decisions, commonly used approaches would have only provided one decision. Thus, we have been successful in *supporting* decision making, rather than making the decision on behalf of the DM.

Another avenue for further investigation would be simulation experiments to learn about various aspects of our approach. Firstly, we are not aware of any simulation experiments which address how often a history matching based approach will rule out the 'true' maximiser. In part, this gap in the literature will be due to the complexities of HM. Describing an NROY region is only practically possible by testing whether points are in the NROY region or not. We do not consider this to be a problem for this work as HM delivers what we need: a greatly reduced decision space from which a DM can choose a good — not necessarily optimal — decision.

Another aspect which would benefit from further investigation would be the choice of design. We pragmatically chose to use an ALM design based on (a) the range of the NROY volume covered by the design and (b) the theory behind ALM designs which suggests points should be spread apart, so we are considering 'diverse' decisions. It would be interesting to see if there is any *numerical* evidence to suggest that this approach is better, or indeed, worse than other methods. We would also like to investigate whether integrating the BayesOpt routine improved the history matching inspired approach. This would be in contrast to Owen *et al.* (2020) who used one shot designs for every wave of emulation, including the first, which may have allowed for greater exploration.

In terms of the decision problem, we considered a static policy in the sense that the decision variables do not change with time. However, many approaches within Tusar & Sarker (2022) were dynamic policies. We could consider two forms of dynamic policy: those which change with time, or those which change as data arrives (for example, updating prior beliefs with data, then maximising the DM's utility function with respect to a posterior distribution). A dynamic policy is where $\mathbf{x}$ is replaced with $\mathbf{x}(t)$, a vector-valued function of time. If $\mathbf{x}$ is replaced by $\mathbf{x}(t)$, the decision problem could be of a much higher dimension than our problem

which has 17 degrees of freedom. HM is an effective approach even in very high dimensional settings, for example White (2018) uses HM to find plausible values for around 2500 inputs. We imagine the methodology used in this chapter and used by Lawson *et al.* (2016) and Owen *et al.* (2020) could be successfully applied in other high dimensional settings. If the policy is dynamic in the sense that beliefs are being updated constantly or periodically, we would use exactly the same approach as above, with the addition that the procedure is run according to a pre-defined schedule. This would allow the DM to revise their utility function if aspects of the problem had changed. For example, the Athena simulator could be revised to offer an improved reflection of reality, or the DM may wish to update their utility function to incorporate new concerns. A dynamic policy however may not be suited to warehouse choice, but rather updating how we run a given warehouse.

Chapter 8

Conclusion

The goal of this thesis was to contribute to the toolkit of the subjective Bayesian; in particular we wished to reduce the computational burden of problems a subjective Bayesian may encounter when working with complex, stochastic simulators.

The contributions of this thesis were application-driven; we considered two problems within an offshore wind farm setting. The first problem was performing a meaningful sensitivity analysis so that a facilitator could prioritise the most important parameters for a SHELF elicitation workshop. This relied on constructing a data-hungry HetGP emulator. To improve the predictive properties of the HetGP emulators in a low-data regime, we developed and constructed the SML emulator. The next problem we considered was performing decision support so that a DM could devise a plan for managing a warehouse containing spare parts to be used when subassemblies suffer serious failures. The computational cost of each problem was greatly reduced by replacing the Athena simulator by suitably constructed emulators. Our emulators allowed us to perform decision support for stochastic simulators, we also considered sampling from NROY regions when the decision space is a discrete simplex.

8.1 *Thesis summary*

We began by introducing the Athena simulator and explaining some of its key components. The standard version of Athena overlooks the fact that spare components for key subassemblies may well be in limited supply. We therefore developed, and incorporated, some additional mechanisms which take into account that spare parts are difficult to obtain. We established that in situations where

relevant data are difficult — or even impossible — to obtain, probability elicitation offers a practical solution to the problem of quantifying parameter uncertainty. We therefore discussed the SHELF; a framework for eliciting probability distributions which allows for, and encourages, the use of many experts for parameter elicitation. In a similar vein, when it is not clear what decision is optimal, we can elicit a utility function from a DM. This motivated our discussion of utility elicitation within the wider context of Bayesian decision analysis.

We then formally introduced the Gaussian Process, a stochastic process which naturally lends itself as a prior distribution for functions. The GP leads us to the notion of an emulator; a fast statistical surrogate model which produces a central prediction for a complex simulator with an appropriate quantification of uncertainty attached. We discussed a handful of common covariance functions and showed that (Bayesian) linear regression is linked to GP regression via linear covariance functions — this fact can be used to improve the interpretability of emulators as the sum of two or more independent GPs is also a GP. We also discussed two other approaches to emulation. Linear regression offers a highly efficient (but perhaps less accurate) approach to emulation. Bayes linear approaches are also well suited to emulation; Bayes linear emulators are mathematically very similar to GPs, but offer a more robust analysis when we believe that a GP approximation of the simulator is too restrictive. These emulators are useful as they do not tie us to any probabilistic beliefs about how we expect an unknown function to behave. The catch is that, within a Bayes linear framework, we are limited in the types of statements we can make about unknowns.

We then explored methods for simultaneous mean and variance emulation. Simultaneous approaches, like HetGP, are more satisfying than the two-emulator approach (Henderson *et al.*, 2009; Andrianakis *et al.*, 2017*b*) as we only need one surrogate for a single quantity of interest. A shortcoming of both of these types of models is that they require a lot of information to produce good quality emulators. This was avoided by the development of the SML emulator which exploited properties of Athena (the ability to run Athena at varying levels of complexity) to construct an improved emulator. We verified that the SML emulator offered an improvement by quantitative measures (RMSE and an appropriate scoring rule), as well as graphical diagnostics suggesting a reduced emulator-simulator discrepancy under SML.

We then used our novel emulator to gain a deeper understanding of how a

subset of the Athena simulator's parameters impact the output of the Athena simulator. We performed a probabalistic sensitivity analysis to see how changing the mean time to degradation impacted the distribution of the mean availability. We found that two parameters were responsible for the majority of the variation in the mean probit availability; the mean time to degradation for the turbine blades and the generator. The parameters were all of roughly equal importance for the variance component of Athena.

We then returned to decision analysis and reviewed various optimisation methods. Since the Athena simulator is computationally expensive, BayesOpt offered a promising solution to maximising a function of the outputs of the Athena simulator. In particular, BayesOpt allows us to maximise an expected utility function which depends on Athena. We also discussed that, because utility functions, elicited beliefs and models will always be imperfect representations of the objects they aim to represent, that allowing the analyst to make the decision, via performing mathematical optimisation, is myopic. The analyst in the problem should only *support* the decision making process; it is up to the DM to make the decision. One way to do this is to employ history matching inspired techniques to construct a set of decisions which are, given all relevant and specified uncertainties, consistent with the maximiser.

We then applied a novel combination of BayesOpt and history matching inspired methods to construct a set of decisions to aid a logistics problem. The Athena simulator featured in the problem by incorporating the availability time series into the elicited utility function. In our particular problem, we managed to greatly simplify the DM's decision by reducing the 'big' warehouse decision down from a selection of 3 warehouses to 2. We also found decisions within each warehouse that were consistent with the approximate maximiser. Our volume of the final set of decisions was around 0.2% of the size of the original decision space. The final set of decisions all lead to reasonably similar availability trajectories from the Athena simulator, which tells us that the DM can choose a warehouse and management policy from the final set of decisions that suits their preferences, without adversely affecting wind farm performance.

8.2 *Future research avenues*

8.2.1 *HetGP, SML and sensitivity analysis*

We showed that our novel SML emulator provided an improved emulator, over the benchmark HetGP, for our motivating Athena example at a fixed training budget. This allowed us to perform a sensitivity analysis to determine the relative importance of a subset of parameters to the Athena simulator.

The design for this example was chosen to be space filling, but beyond this, had no special properties. A promising avenue for future research would be developing designs which leverage our knowledge about the multiple levels of code. Plainly, we should explore sequential designs. Although one 'expensive' run may be computationally equivalent to t 'cheap' runs, it is not clear how informative cheap runs really are for expensive runs. Minimising some criterion, such as integrated mean squared prediction error, would allow the emulator to choose which of (i) a single expensive run or (ii) multiple cheap runs would be most useful for predicting the output of Athena (or some other simulator). This would also allow us to incorporate replication within the design. Replication is encouraged when emulating stochastic simulators with GPs since the cost and inference of GP modelling depends on the number of unique design *locations*, n, rather than the total number of simulator runs, N. If $n << N$ then large computational savings are available due to the cubic/quadratic nature of GP calculations. This allows us to maximise the amount learned about $y(\cdot)$ whilst minimising the cost of implementing our emulators. Another idea that could be explored is incorporating more levels. The autoregressive function structure offers a natural way to incorporate many levels of code (Kennedy & O'Hagan, 2000); we could also use this method to see which level of code is most time effective. Recall that an autoregressive model for functions takes the form

$$f_{t+1}(\cdot) = \rho_t f_t(\cdot) + \delta_t(\cdot) \tag{8.1}$$

where t is a code level, ρ_t is a regression parameter and $\delta_t(\cdot)$ is a discrepancy term. If $\delta_t(\cdot)$ is close to zero, and $\rho_t \approx 1$, then this suggests that $f_t(\mathbf{x}) \approx f_{t+1}(\mathbf{x})$ and thus there is little advantage in using $f_{t+1}(\cdot)$ over $f_t(\cdot)$. The stochastic case should also consider if the output stochasticity is similar in the various code levels.

We would also like to consider ways to assess the joint impact of inputs on the simulator distribution. We mentioned in the discussion of Chapter 5 that EVPI based methods would naturally allow for this when we have access to an appropriate utility function. Off the shelf utility functions, such as those based on the Kullback-Leibler divergence, may offer a good 'default' when we are either unsure of what the utility function should be, or are interested in using the model for tasks such as parameter inference. Borrowing commonly used utility functions from the Bayesian design of experiments literature offers a promising start.

8.2.2 *Decision support*: *multiple stakeholders*

Using history matching techniques for decision support could be expanded to the case of multiple decision makers (typically termed "stakeholders") with different perspectives. Current approaches are somewhat limiting. For example, Keller *et al.* (2009) review several decision problems with multiple stakeholders. Typical solutions to decision problems with multiple stakeholders involve each stakeholder assigning a score to each possible decision by scoring the possible consequences, which are typically discrete in nature. Notably, such methods do not allow for any quantification of uncertainty about which decision is optimal, or the ability to clearly see whether the stakeholders can come to a compromise or not. Application of the single-stakeholder (a single DM) approach as in Chapter 7 would generalise in a natural way.

Applying a history matching inspired approach, with E stakeholders, we could construct E sets of decisions that are consistent with the eth stakeholder's maximiser. Let these sets be $\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_E$. The intersection of all these sets, $\mathcal{X}^* = \bigcap_{e=1}^{E} \mathcal{X}_e$, is the set of mutually agreeable decisions; those that all stakeholders would be happy to take. Note that the DMs would not have to have the same set of attributes. If a DM, Alice, wanted to include x_i in their analysis, but if a DM, Bob, did not want to include x_i, then we can just ensure that Bob's utility function is flat with respect to x_i. In an additive form, set $u_i(x_i) = 0$, and for a multiplicative form set $u_i(x_i) = 1$.

Although this is conceptually no more difficult than applying the approach to a single DM, there are computational and design issues that need to be considered. For example, when constructing wave $k > 1$ designs, we should investigate whether to construct a design amongst the intersection of each decision maker's

NROY set, and evaluate each DM's utility function *only* at the intersection, or whether to construct a design amongst each DM's NROY set and evaluate each DM's utility function at their own set of NROY decisions. The advantage of the first approach is efficiency; if a decision is ruled out as NROY by a single DM, then it would not typically be considered any further. The second approach may be useful when the $U_e(\mathbf{x})$, the utility function of each DM, are replaced by emulators. As knowledge about the $U_e(\mathbf{x})$ evolves over different waves of emulation, the intersection of each $\mathcal{X}_e$ may shrink, but may also grow. An additional consideration that could be built into this framework would be a willingness to compromise. In such a case, the eth DM's implausibility function would be of the form

$$I_e(\mathbf{x}) = \frac{\mathrm{E}\left\{U(\hat{\mathbf{x}}_e)\right\} - \mathrm{E}\left\{U(\mathbf{x})\right\}}{\sqrt{V + \sigma^2_{e,c}}} \tag{8.2}$$

where V represents all the usual sources of uncertainty (for example, emulator variance or model discrepancy), $\hat{\mathbf{x}}_e$ is the (approximate) maximiser of $U_e(\mathbf{x})$ and $\sigma^2_{e,c}$ is a variance-like term which quantifies how willing the eth DM is to compromise. It is not clear how to elicit a compromise parameter; this should be given consideration in future research.

Appendix A

Additional residual plots for the fitted emulators in Chapter 7

A.1 *Plots of SLPEs for the wave* 1 *emulator, with no mean function*

We see in each of the following three groups of residual plots, that apart from x_9, the residuals appear randomly scattered. The variance of the residuals is too large, but we suspect that this is due to the structure present in the plots of SLPEs against x_9. Each warehouse has a pair of plots; one for the first 9 inputs (the numbers of spares, scaled to $[0, 1]$) and then the last 9 inputs (critical percentages, scaled to $[0, 1]$). Although the emulators did not directly take x_1 as an input, we have included it in the residual plots to verify that its exclusion is reasonable.

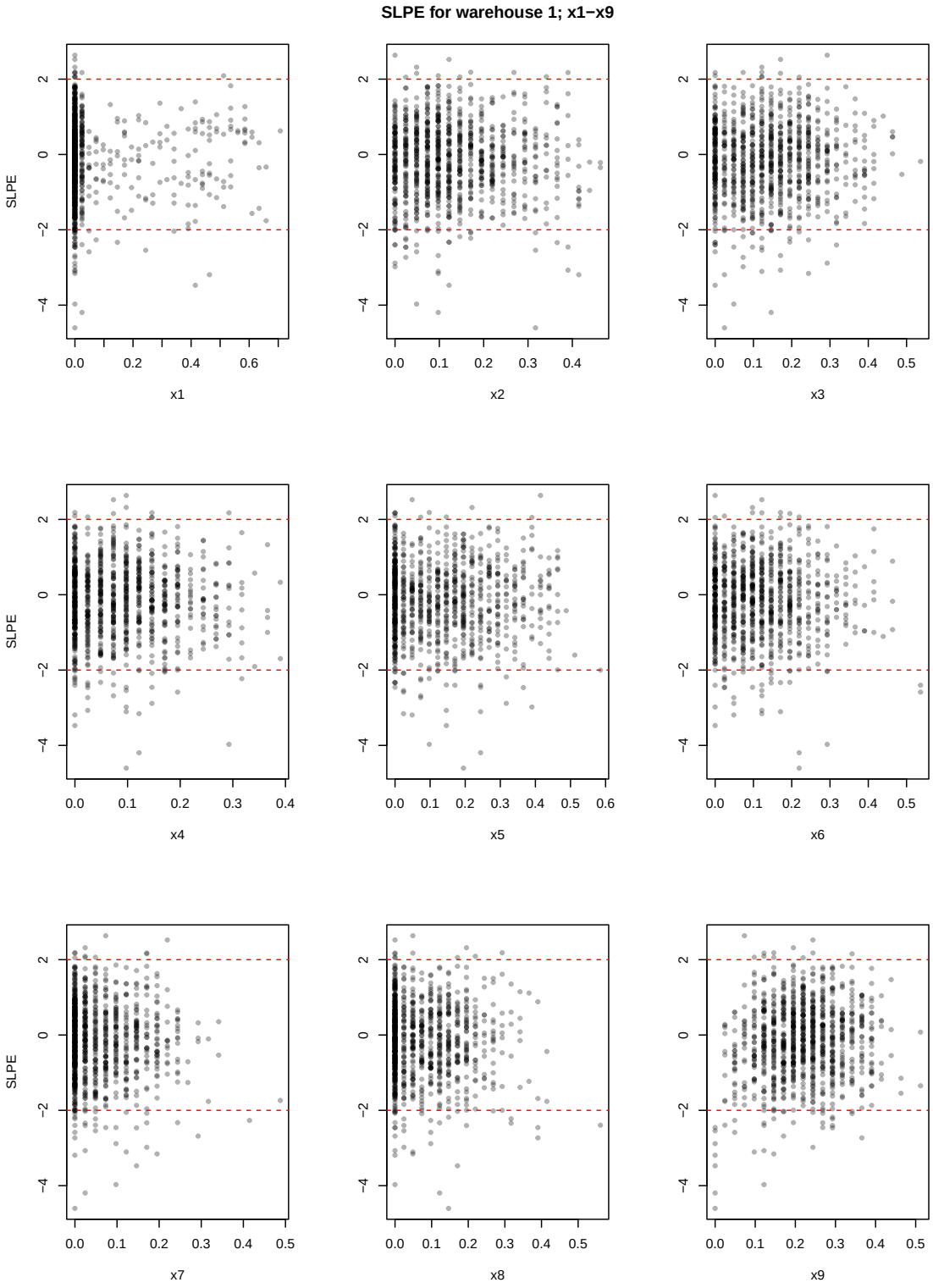

Figure A.1: SLPEs for the first 9 inputs of the wave 1 emulator, for warehouse 1 and $h(\mathbf{x}) = 1$.

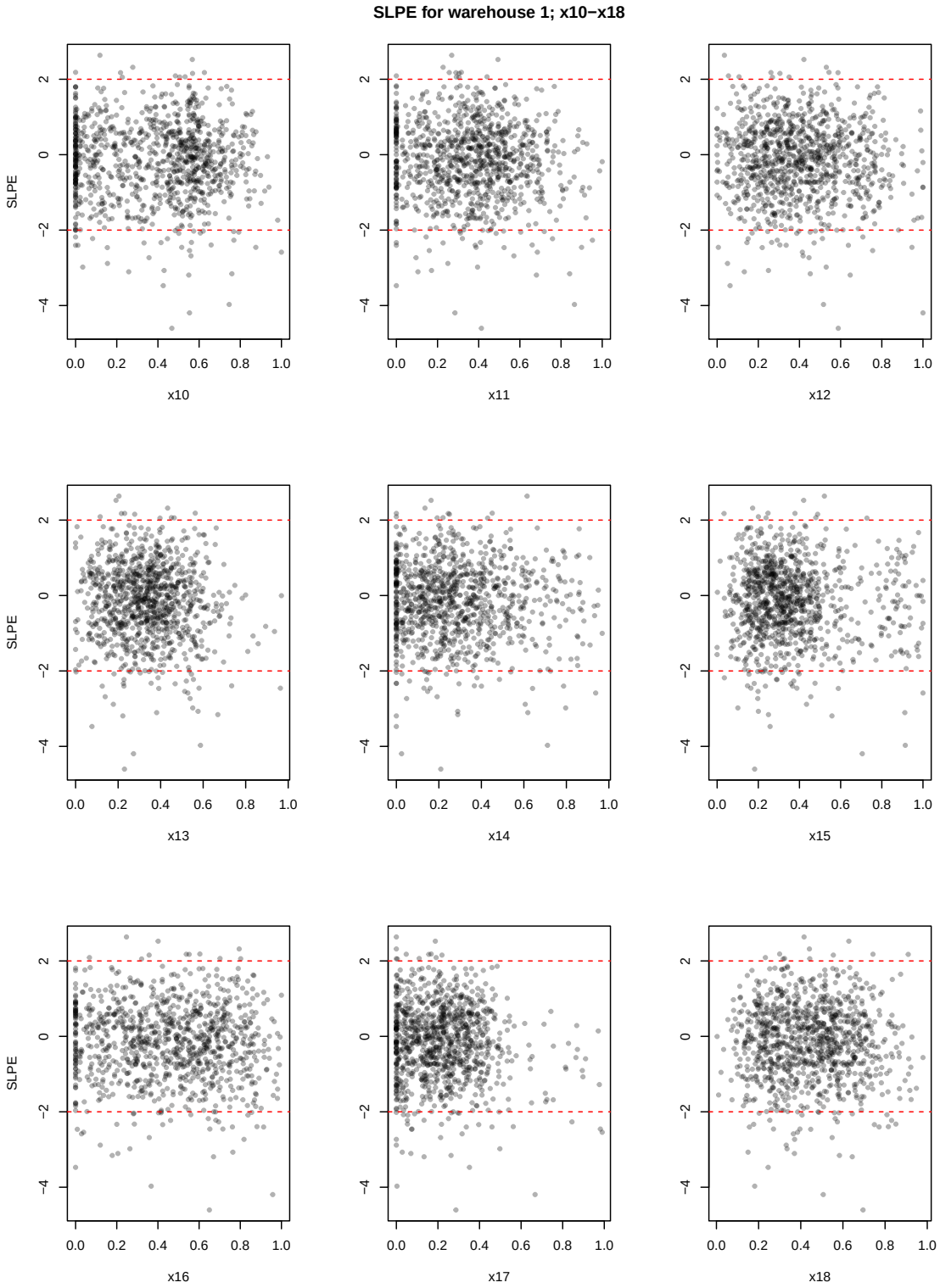

Figure A.2: SLPEs for the last 9 inputs of the wave 1 emulator, for warehouse 1 and $h(\mathbf{x}) = 1$.

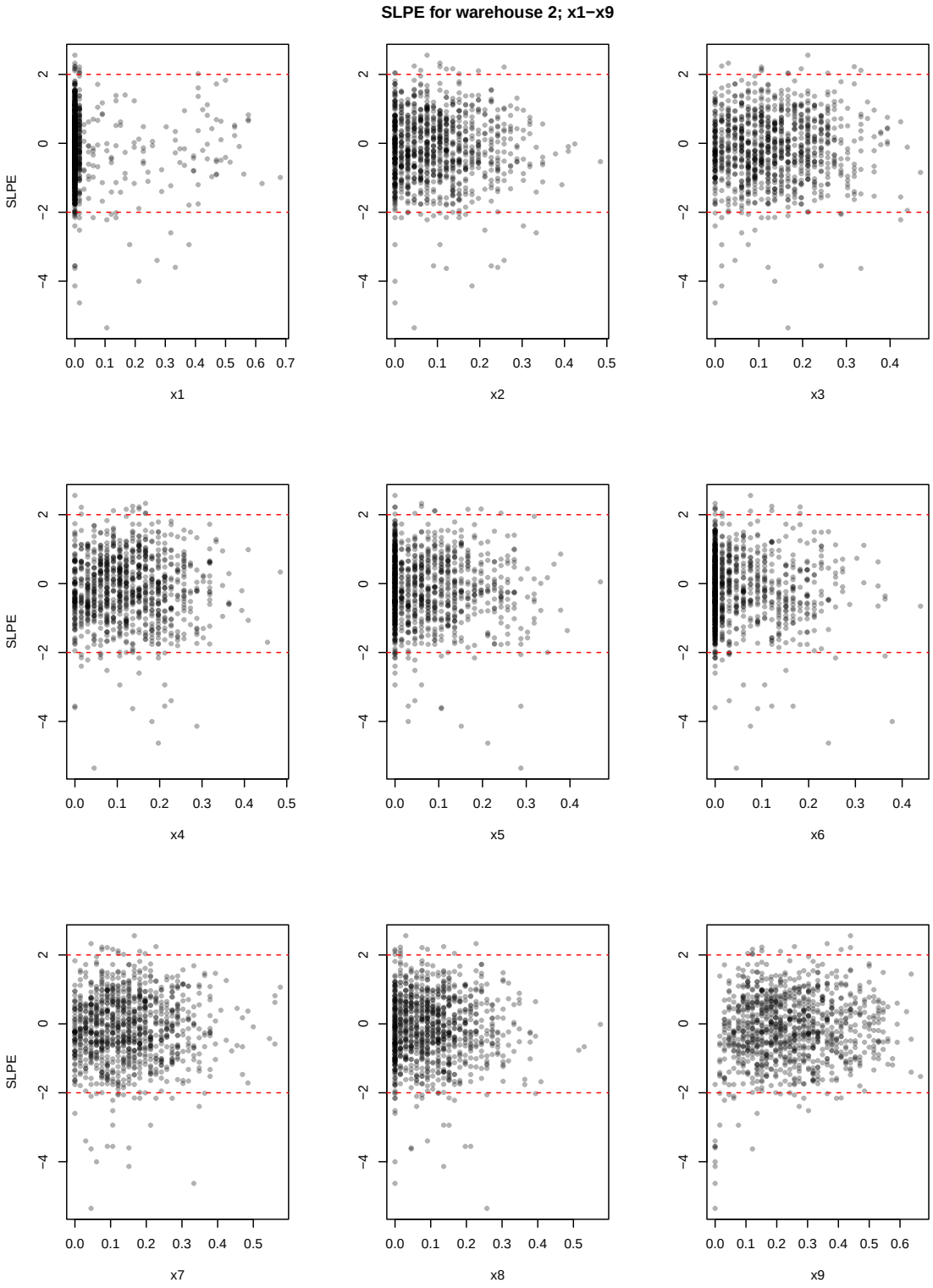

Figure A.3: SLPEs for the first 9 inputs of the wave 1 emulator, for warehouse 2 and $h(\mathbf{x}) = 1$.

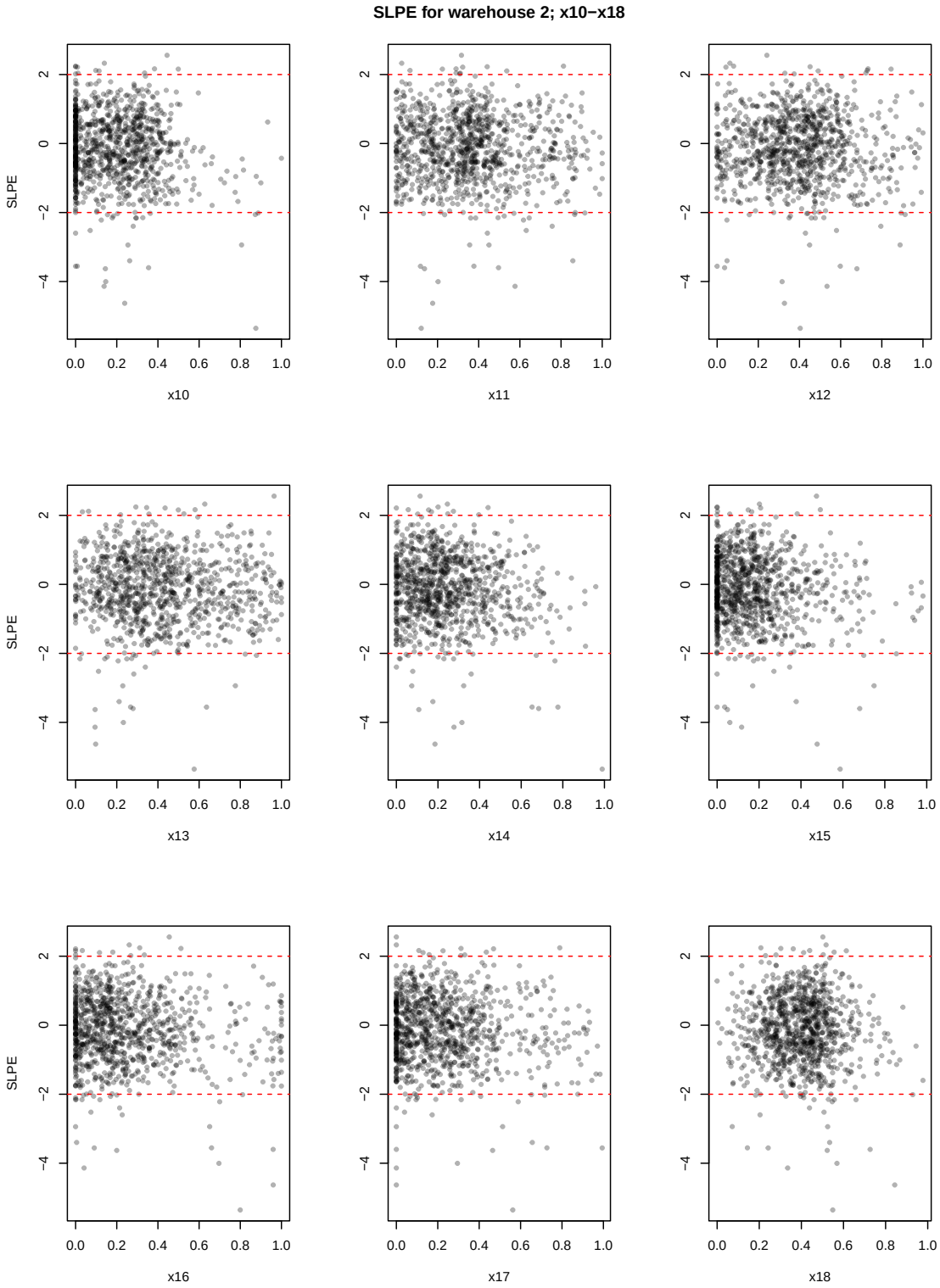

Figure A.4: SLPEs for the last 9 inputs of the wave 1 emulator, for warehouse 2 and $h(\mathbf{x}) = 1$.

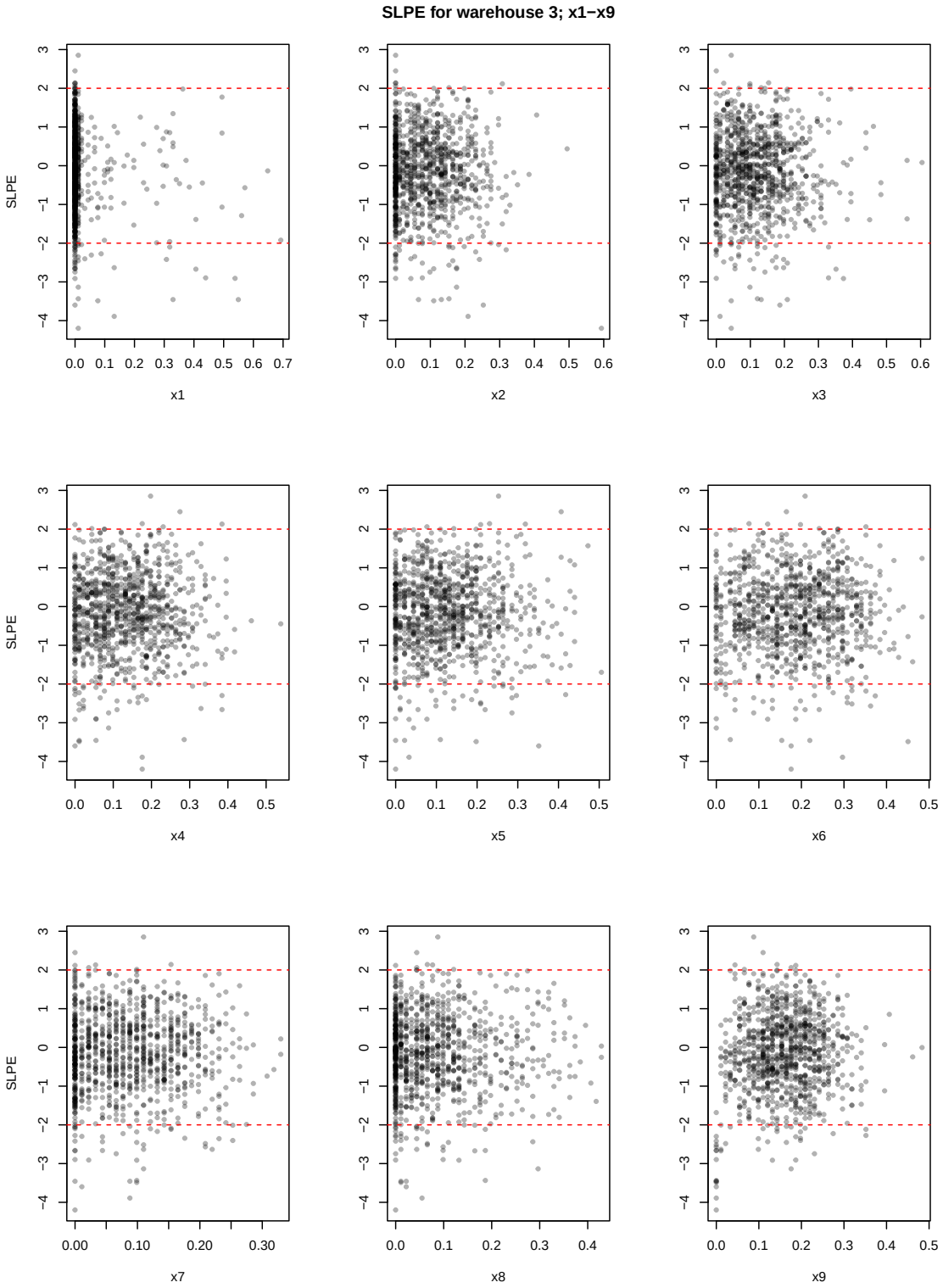

Figure A.5: SLPEs for the first 9 inputs of the wave 1 emulator, for warehouse 3 and $h(\mathbf{x}) = 1$.

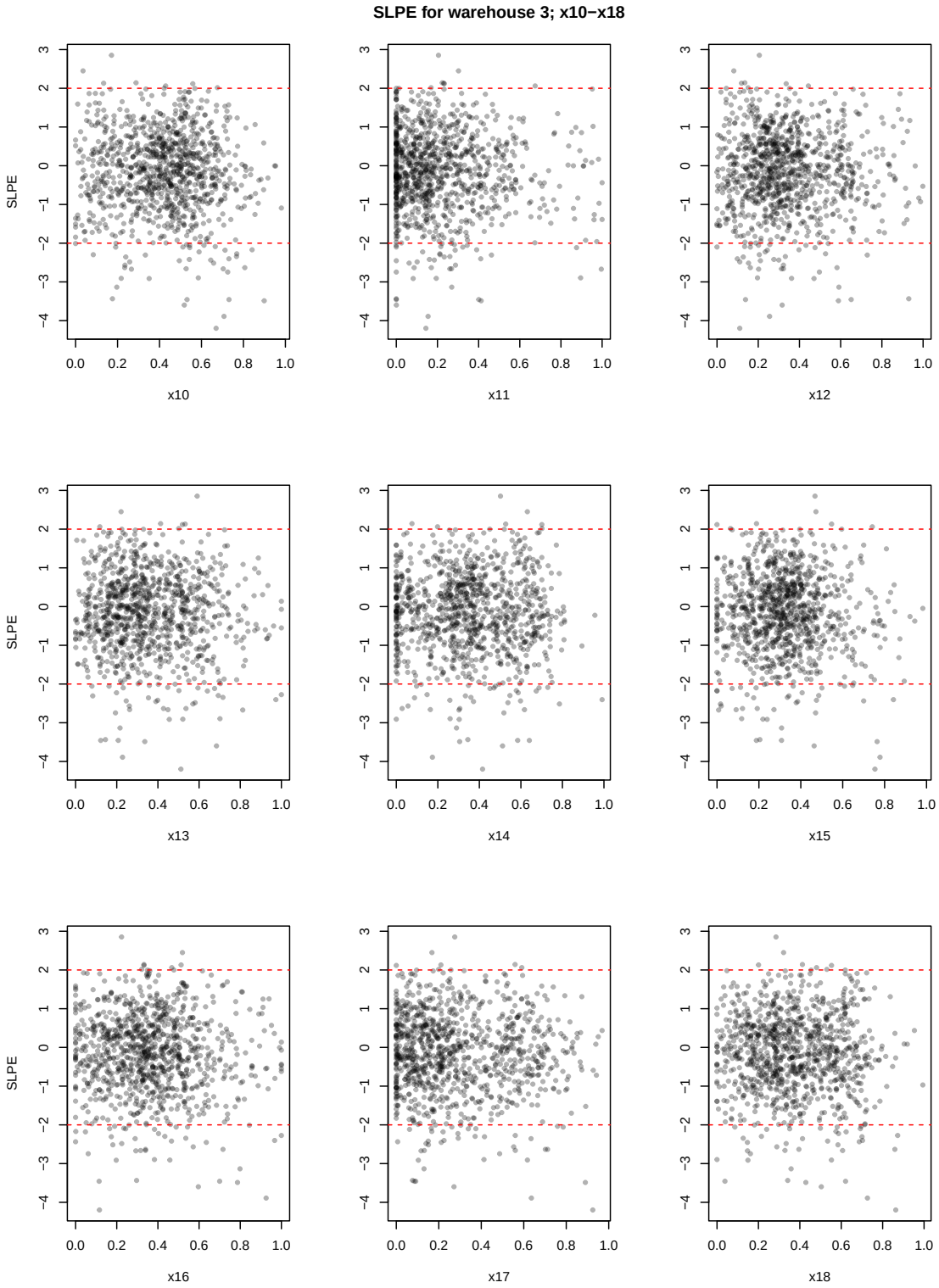

Figure A.6: SLPEs for the last 9 inputs of the wave 1 emulator, for warehouse 3 and $h(\mathbf{x}) = 1$.

A.2 *Plots of SLPEs for the wave* 1 *emulator, with mean function*

The following 6 plots correspond directly to the previous 6. In particular, the values on the x axis are exactly the same in the corresponding plots. The change here is that the residuals are from a different emulator; in particular, the mean function is $h(\mathbf{x}) = (1, \log(x_9 + 10^{-8})$ and the prior $\beta_i \overset{\text{iid}}{\sim} \mathcal{N}\{0, 0.5^2\}$ is assigned to the regression parameters (but integrated out to allow for efficient computation).

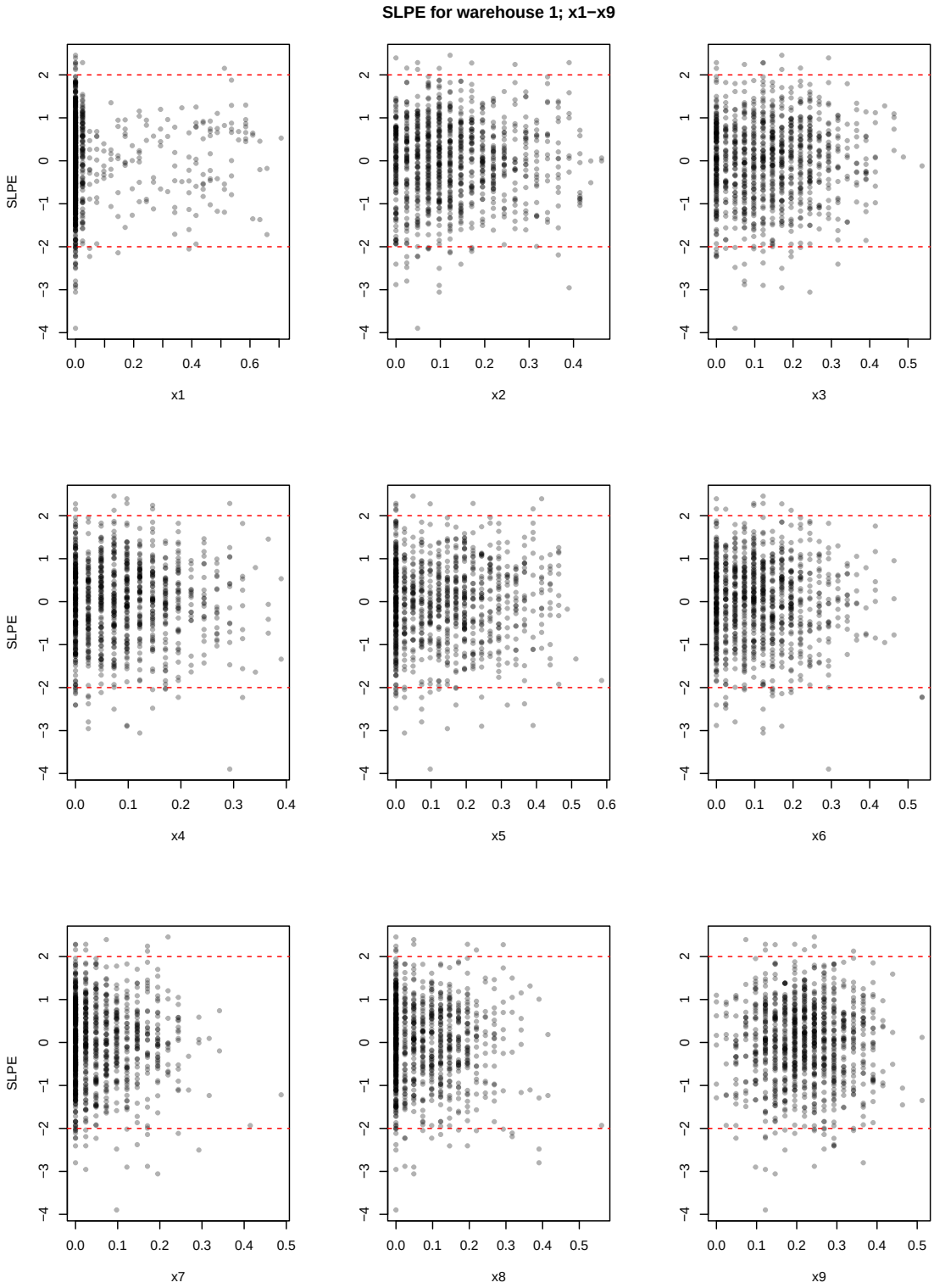

Figure A.7: SLPEs for the first 9 inputs of the wave 1 emulator, for warehouse 1 and $h(\mathbf{x}) = (1, \log(x_9 + 10^{-8}))$.

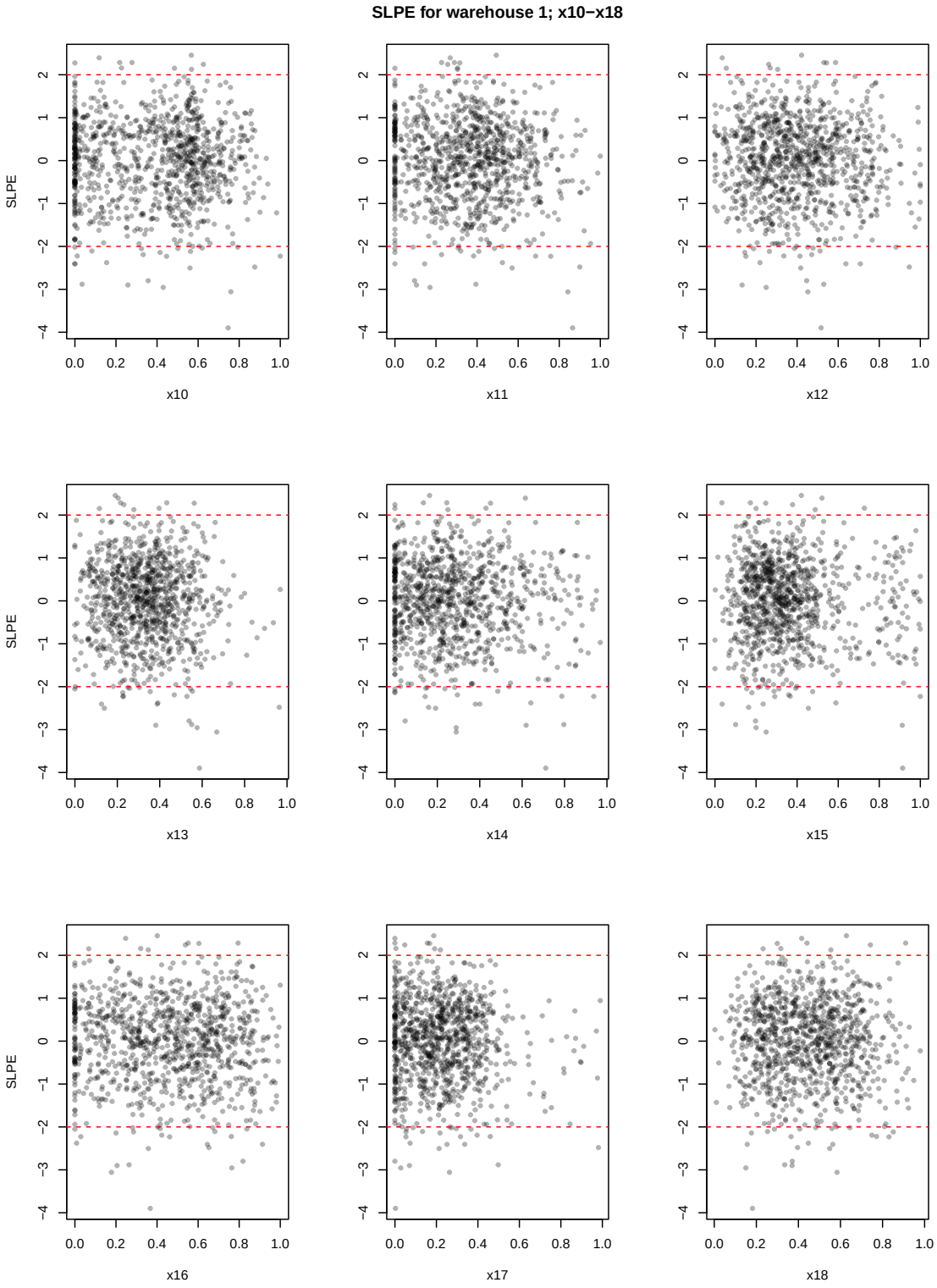

Figure A.8: SLPEs for the last 9 inputs of the wave 1 emulator, for warehouse 1 and $h(\mathbf{x}) = (1, \log(x_9 + 10^{-8}))$.

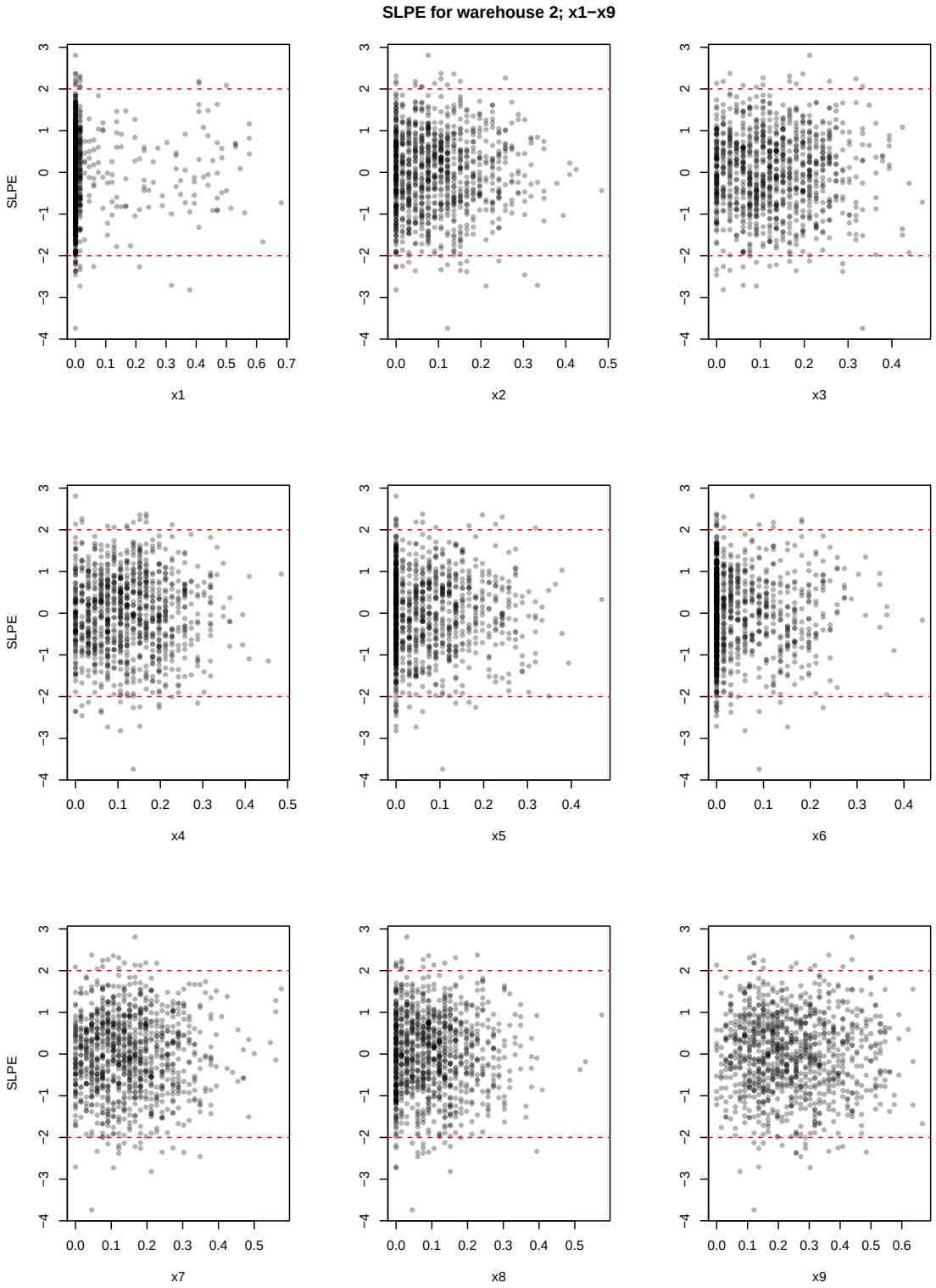

Figure A.9: SLPEs for the first 9 inputs of the wave 1 emulator, for warehouse 2 and $h(\mathbf{x}) = (1, \log(x_9 + 10^{-8}))$.

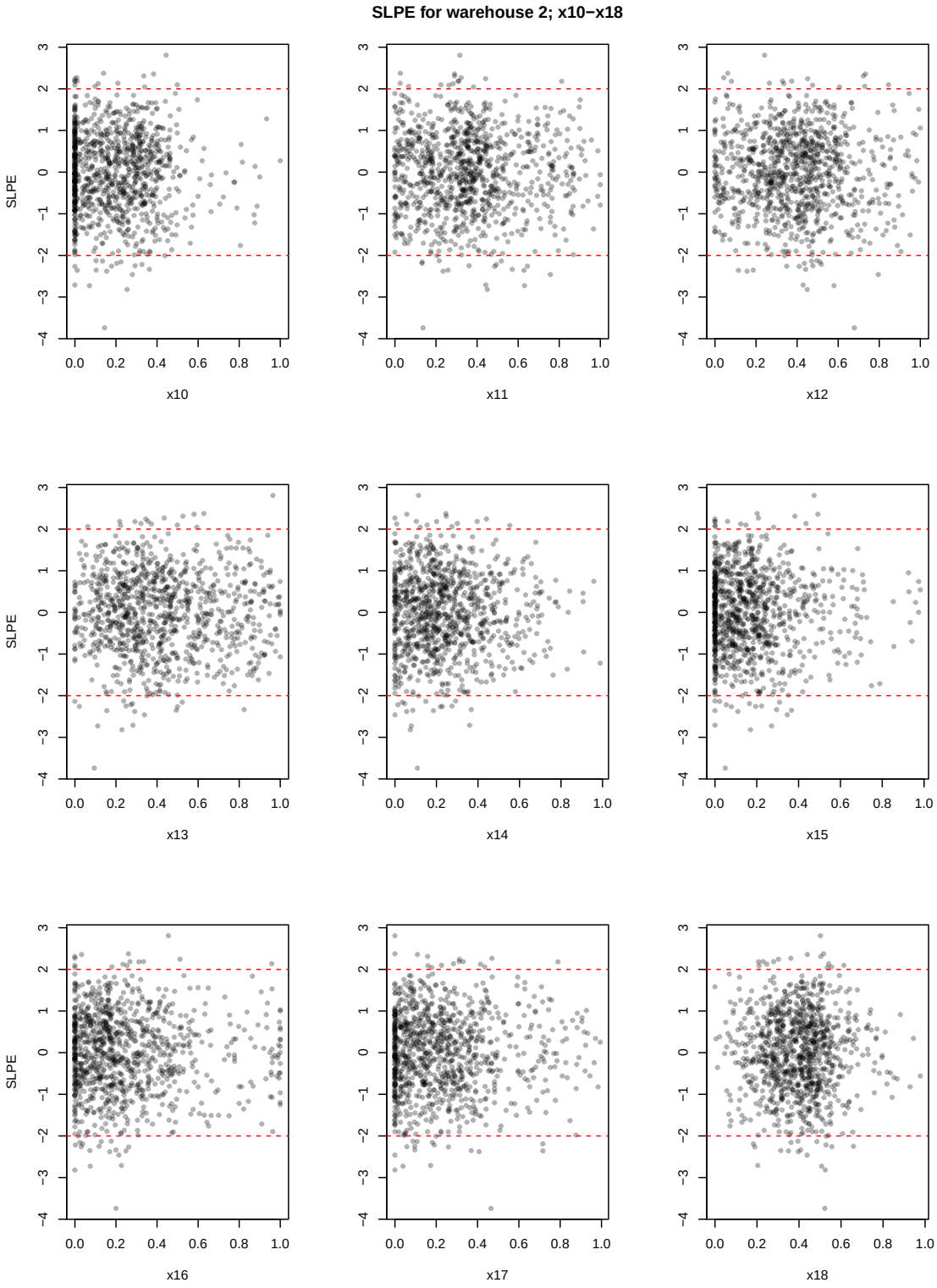

Figure A.10: SLPEs for the first 9 inputs of the wave 2 emulator, for warehouse 2 and $h(\mathbf{x}) = (1, \log(x_9 + 10^{-8}))$.

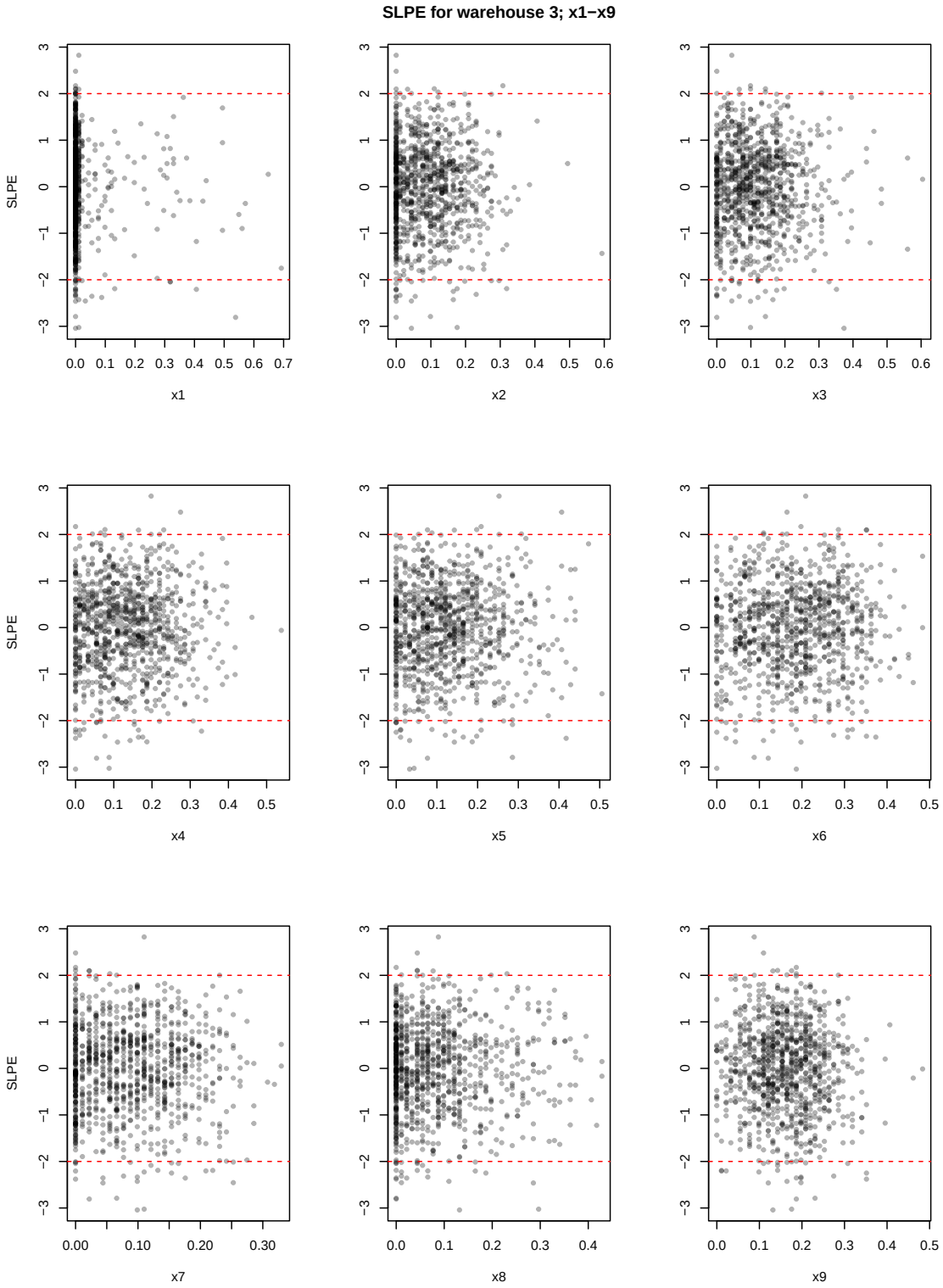

Figure A.11: SLPEs for the first 9 inputs of the wave 1 emulator, for warehouse 3 and $h(\mathbf{x}) = (1, \log(x_9 + 10^{-8}))$.

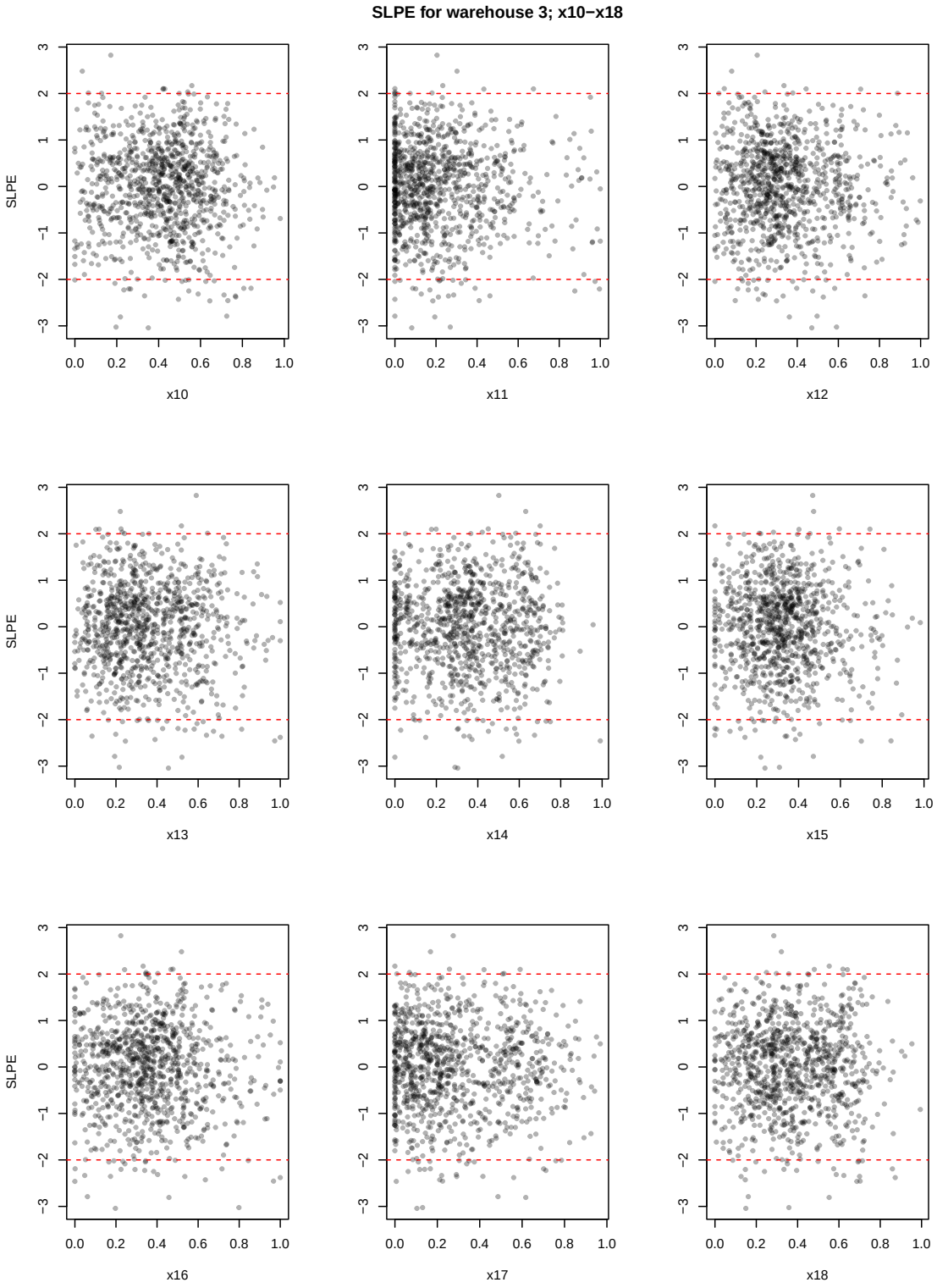

Figure A.12: SLPEs for the last 9 inputs of the wave 1 emulator, for warehouse 3 and $h(\mathbf{x}) = (1, \log(x_9 + 10^{-8}))$.

A.3 *Plots of SLPEs for the wave* 2 *emulators*

The following plots show the SLPEs plotted against the inputs for the wave 2 emulators. Since warehouse 3 was dropped from the analysis, this section has only 4 plots (a pair for warehouse 1, a pair for warehouse 2).

Some of the plots show a slight pattern, however, improving the complexity of the mean function *did not* improve the fit. We put this down the the asymmetry in the distribution of $y(\mathbf{x})$, which is present in Figure 7.14. In particular, the asymmetry in the residual plots below matches the general pattern of the bootstrap distributions in Figure 7.14; all the plots indicate (mild) left-skew.

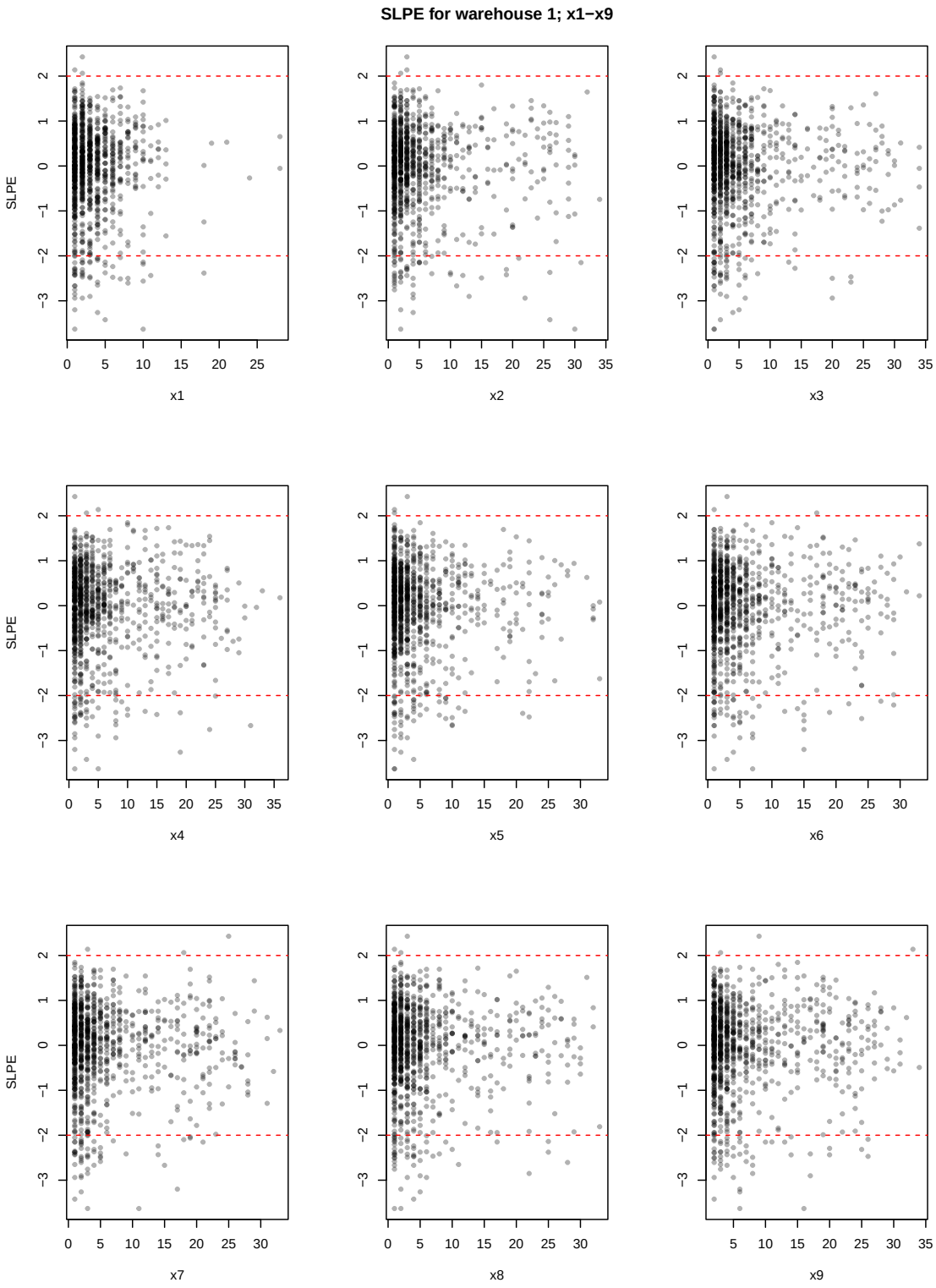

Figure A.13: SLPEs for the first 9 inputs of the warehouse 1, wave 2 emulator.

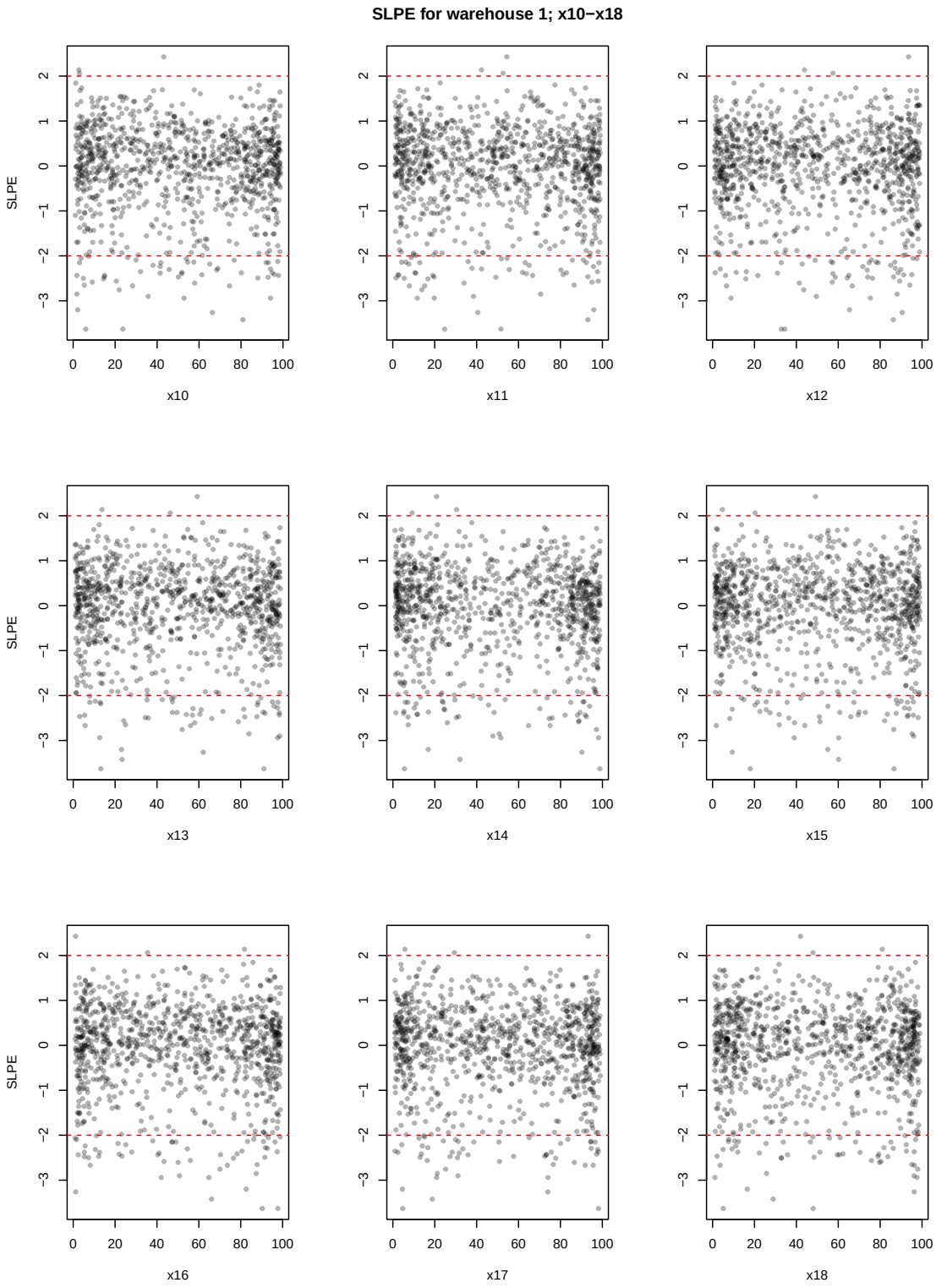

Figure A.14: SLPEs for the last 9 inputs of the warehouse 1, wave 2 emulator.

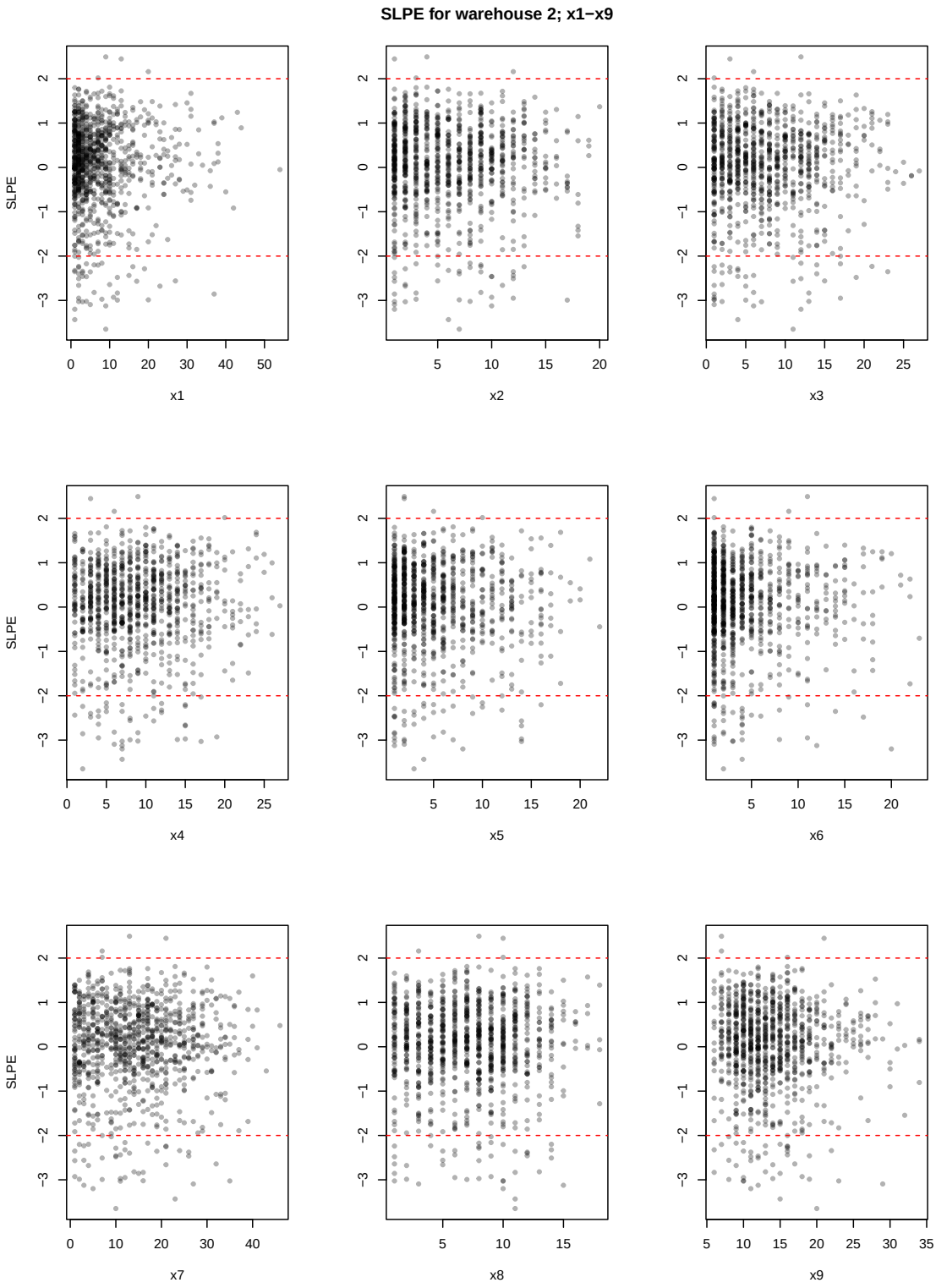

Figure A.15: SLPEs for the first 9 inputs of the warehouse 2, wave 2 emulator.

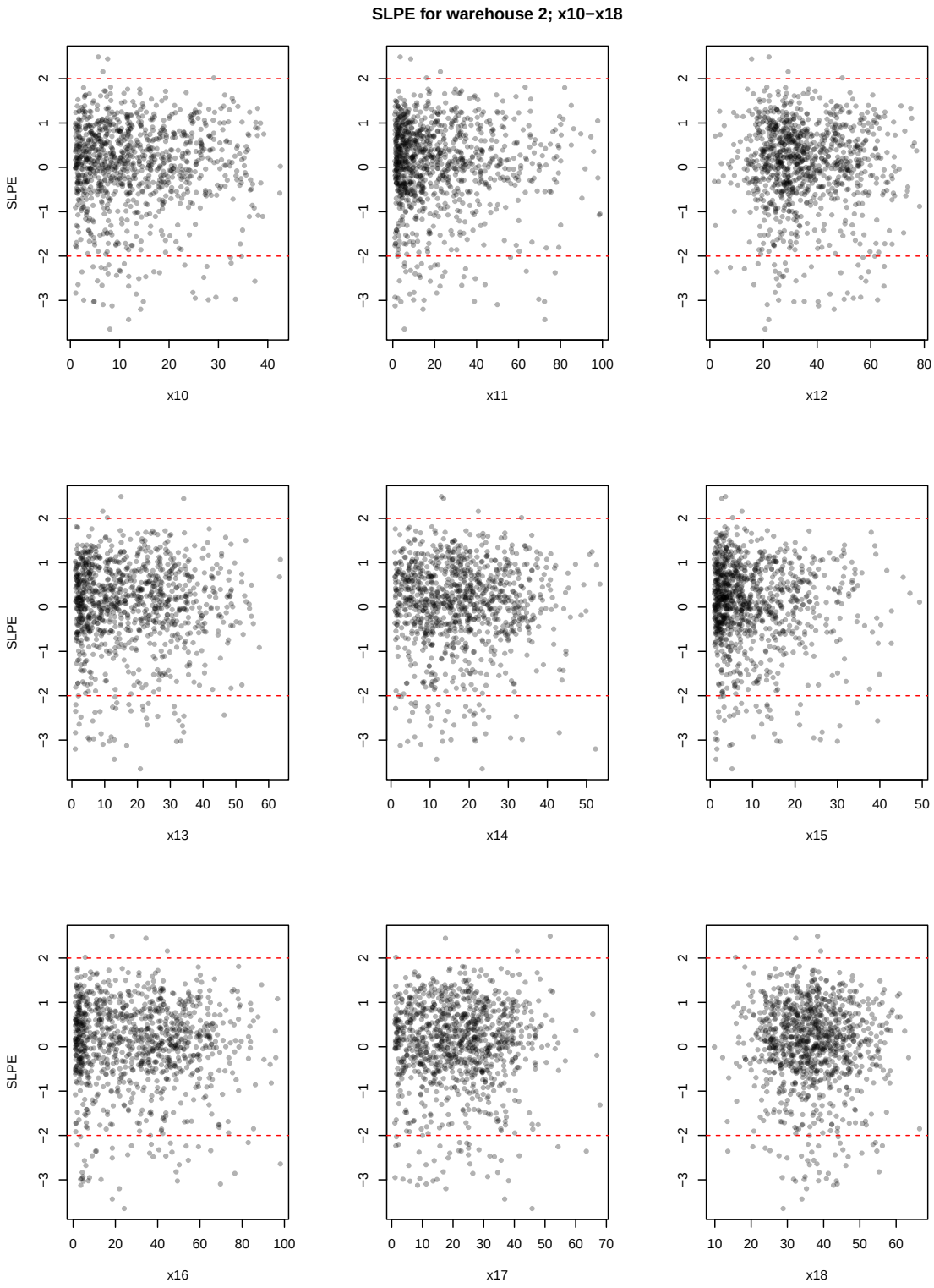

Figure A.16: SLPEs for the first 9 inputs of the warehouse 2, wave 2 emulator.

Bibliography

Adhikary, S. K., Muttil, N. & Yilmaz, A. G. 2017 Cokriging for enhanced spatial interpolation of rainfall in two Australian catchments. *Hydrological processes* **31** (12), 2143–2161.

Andrianakis, I. & Challenor, P. G. 2012 The effect of the nugget on Gaussian process emulators of computer models. *Computational Statistics and Data Analysis* **56** (12), 4215–4228.

Andrianakis, I., McCreesh, N., Vernon, I., McKinley, T. J., Oakley, J. E., Nsubuga, R. N., Goldstein, M. & White, R. G. 2017*a* Efficient history matching of a high dimensional individual-based hiv transmission model. *SIAM/ASA Journal on Uncertainty Quantification* **5** (1), 694–719.

Andrianakis, I., Vernon, I., McCreesh, N., McKinley, T., Oakley, J., Nsubuga, R., Goldstein, M. & White, R. 2017*b* History matching of a complex epidemiological model of human immunodeficiency virus transmission by using variance emulation. *Journal of the Royal Statistical Society. Series C, Applied Statistics* **66** (4), 717.

Ankenman, B., Nelson, B. L. & Staum, J. 2010 Stochastic Kriging for simulation metamodeling. *Operations Research* **58** (2), 371–382.

Ashraf, M., Loftis, J. C. & Hubbard, K. 1997 Application of geostatistics to evaluate partial weather station networks. *Agricultural and Forest Meteorology* **84** (3-4), 255–271.

Astfalck, L., Cripps, E., Gosling, J. & Milne, I. 2019 Emulation of vessel motion simulators for computationally efficient uncertainty quantification. *Ocean Engineering* **172**, 726–736.

Astudillo, R. & Frazier, P. 2020 Multi-attribute Bayesian optimization with interactive preference learning. In *International Conference on Artificial Intelligence and Statistics*, pp. 4496–4507. PMLR.

Baggaley, A. W., Boys, R. J., Golightly, A., Sarson, G. R. & Shukurov, A. 2012 Inference for population dynamics in the Neolithic period. *The Annals of Applied Statistics* **6** (4), 1352–1376.

Bailey, H., Brookes, K. L. & Thompson, P. M. 2014 Assessing environmental impacts of offshore wind farms: lessons learned and recommendations for the future. *Aquatic Biosystems* **10** (1), 1–13.

Baker, E. 2021 Emulation of stochastic computer models with an application to building design. PhD thesis, University of Exeter.

Baker, E., Barbillon, P., Fadikar, A., Gramacy, R. B., Herbei, R., Higdon, D., Huang, J., Johnson, L. R., Ma, P., Mondal, A. *et al.* 2022 Analyzing stochastic computer models: A review with opportunities. *Statistical Science* **37** (1), 64–89.

Baker, E., Challenor, P. & Eames, M. 2019 Diagnostics for stochastic Gaussian process emulators. *arXiv preprint arXiv:1902.01289* .

Baker, E., Challenor, P. & Eames, M. 2020*a* Future proofing a building design using history matching inspired level-set techniques. *Journal of the Royal Statistical Society: Series C (Applied Statistics)* .

Baker, E., Challenor, P. & Eames, M. 2020*b* Predicting the output from a stochastic computer model when a deterministic approximation is available. *Journal of Computational and Graphical Statistics* **29** (4), 786–797.

Barton, R. R. & Meckesheimer, M. 2006 Metamodel-based simulation optimization. *Handbooks in Operations Research and Management Science* **13**, 535–574.

Bastos, L. S. & O'Hagan, A. 2009 Diagnostics for Gaussian process emulators. *Technometrics* **51** (4), 425–438.

Becker, W., Oakley, J., Surace, C., Gili, P., Rowson, J. & Worden, K. 2012 Bayesian sensitivity analysis of a nonlinear finite element model. *Mechanical Systems and Signal Processing* **32**, 18–31.

Bedford, T. & Wilson, K. J. 2014 On the construction of minimum information bivariate copula families. *Annals of the Institute of Statistical Mathematics* **66** (4), 703–723.

Benter, W. 1994 Computer based horse race handicapping and wagering systems: a report. In *Efficiency of Racetrack Betting Markets*, pp. 183–198. World Scientific.

Bergstra, J., Bardenet, R., Bengio, Y. & Kégl, B. 2011 Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems* (ed. J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira & K. Weinberger), , vol. 24. Curran Associates, Inc.

Binois, M. & Gramacy, R. B. 2019 *hetGP: Heteroskedastic Gaussian Process Modeling and Design under Replication*. R package version 1.1.1.

Binois, M., Gramacy, R. B. & Ludkovski, M. 2018 Practical heteroscedastic Gaussian process modeling for large simulation experiments. *Journal of Computational and Graphical Statistics* **27** (4), 808–821.

Binois, M., Huang, J., Gramacy, R. B. & Ludkovski, M. 2019 Replication or exploration? sequential design for stochastic simulation experiments. *Technometrics* **61** (1), 7–23.

Boukouvalas, A. 2010 Emulation of random output simulators. PhD thesis, Aston University.

Bower, R. G., Goldstein, M. & Vernon, I. 2010 Galaxy formation: a Bayesian uncertainty analysis. *Bayesian analysis* **5** (4), 619–669.

Boys, R. J., Ainsworth, H. F. & Gillespie, C. S. 2018 Bayesian inference for a partially observed birth–death process using data on proportions. *Australian & New Zealand Journal of Statistics* **60** (2), 157–173.

Brynjarsdóttir, J. & O'Hagan, A. 2014 Learning about physical parameters: The importance of model discrepancy. *Inverse problems* **30** (11), 114007.

Carnell, R. 2022 *lhs: Latin Hypercube Samples*. R package version 1.1.5.

Carroll, J., McDonald, A. & McMillan, D. 2016 Failure rate, repair time and unscheduled O&M cost analysis of offshore wind turbines. *Wind Energy* **19** (6), 1107–1119.

Challenor, P. 2013 Experimental design for the validation of Kriging metamodels in computer experiments. *Journal of Simulation* **7** (4), 290–296.

Chapelle, O. & Li, L. 2011 An empirical evaluation of Thompson sampling. *Advances in Neural Information Processing Systems* **24**, 2249–2257.

Costello, J. 1971 On the number of points in regular discrete simplex (corresp.). *IEEE Transactions on Information Theory* **17** (2), 211–212.

Craig, P. S., Goldstein, M., Seheult, A. H. & Smith, J. A. 1997 Pressure matching for hydrocarbon reservoirs: a case study in the use of Bayes linear strategies for large computer experiments. In *Case Studies in Bayesian Statistics*, pp. 37–93. Springer.

Criqui, P. & Mima, S. 2012 European climate—energy security nexus: A model based scenario analysis. *Energy Policy* **41**, 827–842.

Cumming, J. A. & Goldstein, M. 2010 Bayes linear uncertainty analysis for oil reservoirs based on multiscale computer experiments. *The Oxford Handbook of Applied Bayesian Analysis* pp. 241–270.

Dalal, G., Bromiley, P. A., Kariki, E. P., Luetchens, S., Cootes, T. F. & Payne, K. 2022 Understanding current UK practice for the incidental identification of vertebral fragility fractures from CT scans: an expert elicitation study. *Aging Clinical and Experimental Research* pp. 1–10.

Dent, C. J., Goldstein, M., Wright, A. & Wynn, H. P. 2020 The use of multiple models within an organisation. *arXiv preprint arXiv:2008.11813* .

Dias, L. C., Morton, A. & Quigley, J. 2018 Elicitation: State of the art and science. In *Elicitation*, pp. 1–14. Springer.

Diessner, M., Guan, Y., Wilson, K. J. & Whalley, R. D. 2022 On the development of a Bayesian optimisation framework for complex unknown systems. *arXiv preprint arXiv:2207.09154* .

Domingo, D., Malmierca-Vallet, I., Sime, L., Voss, J. & Capron, E. 2020 Using ice cores and Gaussian process emulation to recover changes in the Greenland ice sheet during the last interglacial. *Journal of Geophysical Research: Earth Surface* **125** (5), e2019JF005237.

Drovandi, C., Nott, D. J. & Pagendam, D. E. 2021 A semiautomatic method for history matching using sequential Monte Carlo. *SIAM/ASA Journal on Uncertainty Quantification* **9** (3), 1034–1063.

Elfadaly, F. G. & Garthwaite, P. H. 2017 Eliciting Dirichlet and Gaussian copula prior distributions for multinomial models. *Statistics and Computing* **27** (2), 449–467.

European Food Safety Authority 2014 Guidance on expert knowledge elicitation in food and feed safety risk assessment. *EFSA Journal* **12** (6), 3734.

Finkelstein, M. 2007 On statistical and information-based virtual age of degrading systems. *Reliability Engineering & System Safety* **92** (5), 676–681.

Fisher, H. F., Boys, R. J., Gillespie, C. S., Proctor, C. J. & Golightly, A. 2022 Parameter inference for a stochastic kinetic model of expanded polyglutamine proteins. *Biometrics* **78** (3), 1195–1208.

Forrester, A. I., Sóbester, A. & Keane, A. J. 2007 Multi-fidelity optimization via surrogate modelling. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* **463** (2088), 3251–3269.

Frazier, P. I. 2018 A tutorial on Bayesian optimization. *arXiv preprint arXiv:1807.02811* .

Fricker, T. E., Oakley, J. E. & Urban, N. M. 2013 Multivariate Gaussian process emulators with nonseparable covariance structures. *Technometrics* **55** (1), 47–56.

Garbuno-Inigo, A., DiazDelaO, F. A. & Zuev, K. M. 2020 History matching with probabilistic emulators and active learning. *arXiv preprint arXiv:2004.07878* .

Garthwaite, P. H., Kadane, J. B. & O'Hagan, A. 2005 Statistical methods for eliciting probability distributions. *Journal of the American Statistical Association* **100** (470), 680–701.

Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A. & Rubin, D. B. 2013 *Bayesian Data Analysis*. CRC press.

Gentle, J. E. 2007 *Matrix Algebra Theory, Computations, and Applications in Statistics*. New York, N.Y. ; [London]: Springer.

Gneiting, T. 2002 Nonseparable, stationary covariance functions for space–time data. *Journal of the American Statistical Association* **97** (458), 590–600.

Gneiting, T. & Raftery, A. E. 2007 Strictly proper scoring rules, prediction, and estimation. *Journal of the American Statistical Association* **102** (477), 359–378.

Goldberg, P. W., Williams, C. K. & Bishop, C. M. 1998 Regression with input-dependent noise: A Gaussian process treatment. In *Advances in Neural Information Processing Systems*, pp. 493–499.

Goldstein, M. 2007 *Bayes Linear Statistics: Theory and Methods*. Hoboken, N.J.: John Wiley.

Goldstein, M. & Rougier, J. 2009 Reified bayesian modelling and inference for physical systems. *Journal of Statistical Planning and Inference* **139** (3), 1221–1239.

González-Ortega, J., Radovic, V. & Insua, D. R. 2018 Utility elicitation. In *Elicitation*, pp. 241–264. Springer.

Gramacy, R. & Lee, H. 2012 Cases for the nugget in modelling computer experiments. *Statistics and Computing* **22** (3), 713–722.

Gramacy, R. B. 2020 *Surrogates: Gaussian Process Modeling, Design and Optimization for the Applied Sciences*. Boca Raton, Florida: Chapman Hall/CRC.

Gramacy, R. B. & Lee, H. K. H. 2008 Bayesian treed Gaussian process models with an application to computer modeling. *Journal of the American Statistical Association* **103** (483), 1119–1130.

Gramacy, R. B. & Lee, H. K. H. 2011 Optimization under unknown constraints. In *Bayesian Statistics 9*. Oxford: Oxford University Press.

Gramacy, R. B. & Polson, N. G. 2011 Particle learning of Gaussian process models for sequential design and optimization. *Journal of Computational and Graphical Statistics* **20** (1), 102–118.

Grewal, P. S. & Grewal, P. S. 2013 Can cities become self-reliant in energy? a technological scenario analysis for Cleveland, Ohio. *Cities* **31**, 404–411.

Gutin, G. 2007 *The Traveling Salesman Problem and Its Variations*. Boston, MA: Springer US.

Harvey, N. J., Huntley, N., Dacre, H. F., Goldstein, M., Thomson, D. & Webster, H. 2018 Multi-level emulation of a volcanic ash transport and dispersion model to quantify sensitivity to uncertain parameters. *Natural Hazards and Earth System Sciences* **18** (1), 41–63.

Henderson, D. A., Boys, R. J., Krishnan, K. J., Lawless, C. & Wilkinson, D. J. 2009 Bayesian emulation and calibration of a stochastic computer model of mitochondrial DNA deletions in substantia nigra neurons. *Journal of the American Statistical Association* **104** (485), 76–87.

Higham, N. J. 2009 Cholesky factorization. *Wiley Interdisciplinary Reviews: Computational Statistics* **1** (2), 251–254.

Holvoet, K., van Griensven, A., Seuntjens, P. & Vanrolleghem, P. 2005 Sensitivity analysis for hydrology and pesticide supply towards the river in swat. *Physics and Chemistry of the Earth, Parts A/B/C* **30** (8-10), 518–526.

Jackson, S. E. & Vernon, I. 2023 Efficient emulation of computer models utilising multiple known boundaries of differing dimension. *Bayesian Analysis* **18** (1), 165–191.

Jackson, S. E., Vernon, I., Liu, J. & Lindsey, K. 2018 Bayesian uncertainty analysis establishes the link between the parameter space of a complex model of hormonal crosstalk in arabidopsis root development and experimental measurements. *Stat. Appl. Genet. Mol. Biol* **11**.

Jackson, S. E. & Woods, D. C. 2019 Bayes linear emulation of simulator networks. *arXiv preprint arXiv:1910.08003* .

Jones, D. R., Schonlau, M. & Welch, W. J. 1998 Efficient global optimization of expensive black-box functions. *Journal of Global optimization* **13** (4), 455–492.

Jones, M., Goldstein, M., Jonathan, P. & Randell, D. 2016 Bayes linear analysis for Bayesian optimal experimental design. *Journal of Statistical Planning and Inference* **171**, 115–129.

Joy, T. T., Rana, S., Gupta, S. & Venkatesh, S. 2016 Hyperparameter tuning for big data using Bayesian optimisation. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pp. 2574–2579. IEEE.

Kadane, J. & Wolfson, L. J. 1998 Experiences in elicitation. *Journal of the Royal Statistical Society: Series D (The Statistician)* **47** (1), 3–19.

Kan, A. R. & Timmer, G. 1989 Chapter IX Global optimization. *Handbooks in Operations Research and Management Science* **1**, 631–662.

Keeney, R. L. & Raiffa, H. 1976 *Decisions with Multiple Objectives: Preferences and Value Rradeoffs*. New York: Wiley.

Keller, L. R., Simon, J. & Wang, Y. 2009 Multiple-objective decision analysis involving multiple stakeholders. In *Decision Technologies and Applications*, pp. 139–155. Informs.

Kennedy, J. C., Henderson, D. A. & Wilson, K. J. 2023 Multilevel emulation for stochastic computer models with application to large offshore wind farms. *Journal of the Royal Statistical Society Series C: Applied Statistics* .

Kennedy, M. & O'Hagan, A. 2000 Predicting the output from a complex computer code when fast approximations are available. *Biometrika* **87** (1), 1–13.

Kennedy, M. & O'Hagan, A. 2001 Bayesian calibration of computer models. *Journal of The Royal Statistical Society Series B-Statistical Methodology* **63**, 425–450.

Kennedy, M. C., Anderson, C. W., Conti, S. & O'Hagan, A. 2006 Case studies in Gaussian process modelling of computer codes. *Reliability Engineering & System Safety* **91** (10-11), 1301–1309.

Kersting, K., Plagemann, C., Pfaff, P. & Burgard, W. 2007 Most likely heteroscedastic Gaussian process regression. In *Proceedings of the 24th international conference on Machine learning*, pp. 393–400. ACM.

Khalid, K., Ali, M., Abd Rahman, N. & Mispan, M. 2016 Application on one-at-a-time sensitivity analysis of semi-distributed hydrological model in tropical watershed. *International Journal of Engineering and Technology* **8** (2), 132–136.

Kim, T., Lee, J. & Choe, Y. 2020 Bayesian optimization-based global optimal rank selection for compression of convolutional neural networks. *IEEE Access* **8**, 17605–17618.

Kirkpatrick, S., Gelatt Jr, C. D. & Vecchi, M. P. 1983 Optimization by simulated annealing. *Science* **220** (4598), 671–680.

Kotidis, P., Demis, P., Goey, C. H., Correa, E., McIntosh, C., Trepekli, S., Shah, N., Klymenko, O. V. & Kontoravdi, C. 2019 Constrained global sensitivity analysis for bioprocess design space identification. *Computers & Chemical Engineering* **125**, 558–568.

Kucherenko, S., Klymenko, O. V. & Shah, N. 2017 Sobol' indices for problems defined in non-rectangular domains. *Reliability Engineering & System Safety* **167**, 218–231.

Lark, R. & Bishop, T. 2007 Cokriging particle size fractions of the soil. *European Journal of Soil Science* **58** (3), 763–774.

Lawson, A., Goldstein, M. & Dent, C. 2016 Bayesian framework for power network planning under uncertainty. *Sustainable Energy, Grids and Networks* **7**, 47 – 57.

Le Gratiet, L. & Cannamela, C. 2015 Cokriging-based sequential design strategies using fast cross-validation techniques for multi-fidelity computer codes. *Technometrics* **57** (3), 418–427.

Le Gratiet, L. & Garnier, J. 2014 Recursive co-Kriging model for design of computer experiments with multiple levels of fidelity. *International Journal for Uncertainty Quantification* **4** (5).

Ling, Y., Mullins, J. & Mahadevan, S. 2014 Selection of model discrepancy priors in Bayesian calibration. *Journal of Computational Physics* **276**, 665–680.

Marmin, S., Chevalier, C. & Ginsbourger, D. 2015 Differentiating the multipoint expected improvement for optimal batch design. In *International Workshop on Machine Learning, Optimization and Big Data*, pp. 37–48. Springer.

Marrel, A., Iooss, B., Da Veiga, S. & Ribatet, M. 2012 Global sensitivity analysis of stochastic computer models with joint metamodels. *Statistics and Computing* **22** (3), 833–847.

McKay, M. D., Beckman, R. J. & Conover, W. J. 1979 Comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics* **21** (2), 239–245.

Močkus, J. 1975 On Bayesian methods for seeking the extremum. In *Optimization Techniques IFIP Technical Conference*, pp. 400–404. Springer.

Morris, M. D. & Mitchell, T. J. 1995 Exploratory designs for computational experiments. *Journal of Statistical Planning and Inference* **43** (3), 381–402.

Neal, R. M. 2003 Slice sampling. *The Annals of Statistics* **31** (3), 705–767.

Nelson, W. 1980 Accelerated life testing-step-stress models and data analyses. *IEEE Transactions on Reliability* **29** (2), 103–108.

Nguyen, V. & Osborne, M. A. 2020 Knowing the what but not the where in Bayesian optimization. In *International Conference on Machine Learning*, pp. 7317–7326. PMLR.

Nguyen, V.-L., Shaker, M. H. & Hüllermeier, E. 2022 How to measure uncertainty in uncertainty sampling for active learning. *Machine Learning* **111** (1), 89–122.

Oakley, J. 2002 Eliciting Gaussian process priors for complex computer codes. *Journal of the Royal Statistical Society: Series D (The Statistician)* **51** (1), 81–97.

Oakley, J. 2021 *SHELF: Tools to Support the Sheffield Elicitation Framework*. R package version 1.8.0.

Oakley, J. & O'Hagan, A. 2002 Bayesian inference for the uncertainty distribution of computer model outputs. *Biometrika* **89** (4), 769–784.

Oakley, J. E. 2009 Decision-theoretic sensitivity analysis for complex computer models. *Technometrics* **51** (2), 121–129.

Oakley, J. E. & O'Hagan, A. 2004 Probabilistic sensitivity analysis of complex models: a Bayesian approach. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* **66** (3), 751–769.

Oakley, J. E. & O'Hagan, A. 2019 SHELF: the Sheffield Elicitation Framework (Version 4). `http://tonyohagan.co.uk/shelf`.

O'Hagan, A., Buck, C. E., Daneshkhah, A., Eiser, J. R., Garthwaite, P. H., Jenkinson, D. J., Oakley, J. E. & Rakow, T. 2006 *Uncertain Judgements: Eliciting Expert Probabilities*. Chichester: John Wiley.

Overstall, A. M. & Woods, D. C. 2016 Multivariate emulation of computer simulators: model selection and diagnostics with application to a humanitarian relief model. *Journal of the Royal Statistical Society. Series C, Applied statistics* **65** (4), 483.

Overstall, A. M. & Woods, D. C. 2017 Bayesian design of experiments using approximate coordinate exchange. *Technometrics* **59** (4), 458–470.

Owen, J., Vernon, I. & Hammersley, R. 2020 A Bayesian statistical approach to decision support for TNO OLYMPUS well control optimisation under uncertainty. In *ECMOR XVII*, pp. 1–27. European Association of Geoscientists & Engineers.

Pedersen, S. & Ahsan, D. 2020 Emergency preparedness and response: Insights from the emerging offshore wind industry. *Safety Science* **121**, 516–528.

Pham, H. 2006 *Springer Handbook of Engineering Statistics*, , vol. 49. Springer.

Pholdee, N. & Bureerat, S. 2015 An efficient optimum Latin hypercube sampling technique based on sequencing optimisation using simulated annealing. *International Journal of Systems Science* **46** (10), 1780–1789.

Plumlee, M. & Tuo, R. 2014 Building accurate emulators for stochastic simulations via quantile Kriging. *Technometrics* **56** (4), 466–473.

Pourmohamad, T. & Lee, H. K. 2021 *Bayesian Optimization with Application to Computer Experiments*. Springer.

Prangle, D., Harbisher, S. & Gillespie, C. S. 2022 Bayesian experimental design without posterior calculations: an adversarial approach. *Bayesian Analysis* **1** (1), 1–31.

Pronzato, L. & Zhigljavsky, A. 2020 Bayesian quadrature, energy minimization, and space-filling design. *SIAM/ASA Journal on Uncertainty Quantification* **8** (3), 959–1011.

Pukelsheim, F. 1994 The three sigma rule. *The American Statistician* **48** (2), 88–91.

Rainforth, T., Cornish, R., Yang, H., Warrington, A. & Wood, F. 2018 On nesting Monte Carlo estimators. In *Proceedings of the 35th International Conference on Machine Learning* (ed. J. Dy & A. Krause), *Proceedings of Machine Learning Research*, vol. 80, pp. 4267–4276. PMLR.

Rasmussen, C. E. & Williams, C. K. I. 2006 *Gaussian processes for Machine Learning*. Cambridge, Mass.: MIT Press.

Ríos Insua, S., Jiménez Martín, A., Mateos Caballero, A. & Prieto Romeau, T. 2003 A multiattribute decision support system with imprecise input and time-dependent modelling. *International Journal of Technology, Policy and Management* **3** (3/4), 230–250.

Riutort-Mayol, G., Bürkner, P.-C., Andersen, M. R., Solin, A. & Vehtari, A. 2023 Practical hilbert space approximate bayesian gaussian processes for probabilistic programming. *Statistics and Computing* **33** (1), 17.

Rocchetta, R., Zio, E. & Patelli, E. 2018 A power-flow emulator approach for resilience assessment of repairable power grids subject to weather-induced failures and data deficiency. *Applied Energy* **210**, 339–350.

Roijers, D. M. & Whiteson, S. 2017 Multi-objective decision making. *Synthesis Lectures on Artificial Intelligence and Machine Learning* **11** (1), 1–129.

Roth, S. M., Lee, B. S., Sharma, S., Hosseini-Shakib, I., Keller, K. & Haran, M. 2022 Flood hazard model calibration using multiresolution model output. *Environmetrics* p. e2769.

Rougier, J., Sexton, D. M., Murphy, J. M. & Stainforth, D. 2009 Analyzing the climate sensitivity of the hadsm3 climate model using ensembles from different but related experiments. *Journal of Climate* **22** (13), 3540–3557.

Rowe, G. & Wright, G. 1999 The Delphi technique as a forecasting tool: issues and analysis. *International Journal of Forecasting* **15** (4), 353–375.

Ryan, E. G., Drovandi, C. C., McGree, J. M. & Pettitt, A. N. 2016 A review of modern computational algorithms for Bayesian optimal design. *International Statistical Review* **84** (1), 128–154.

Sacks, J., Welch, W. J., Mitchell, T. J. & Wynn, H. P. 1989 Design and analysis of computer experiments. *Statistical Science* **4** (4), 409–423.

Saisana, M., Saltelli, A. & Tarantola, S. 2005 Uncertainty and sensitivity analysis techniques as tools for the quality assessment of composite indicators. *Journal of the Royal Statistical Society: Series A (Statistics in Society)* **168** (2), 307–323.

Saltelli, A. & Annoni, P. 2010 How to avoid a perfunctory sensitivity analysis. *Environmental Modelling & Software* **25** (12), 1508–1517.

Salter, J. M. & Williamson, D. 2016 A comparison of statistical emulation methodologies for multi-wave calibration of environmental models. *Environmetrics* **27** (8), 507–523.

Samsó, R., de Blas, I., Perissi, I., Martelloni, G. & Solé, J. 2020 Scenario analysis and sensitivity exploration of the MEDEAS Europe energy-economy-environment model. *Energy Strategy Reviews* **32**, 100582.

Santner, T. J., Williams, B. J., Notz, W. & Williams, B. J. 2003 *The Design and Analysis of Computer Experiments*, 1st edn. Springer.

Saraiva, J. P., Lima, B. S., Gomes, V. M., Flores, P. H., Gomes, F. A., Assis, A. O., Reis, M. R., Araújo, W. R., Abrenhosa, C. & Calixto, W. P. 2017 Calculation of sensitivity index using one-at-a-time measures based on graphical analysis. In *2017 18th International Scientific Conference on Electric Power Engineering (EPE)*, pp. 1–6. IEEE.

Schmidt, G., Mattern, R. & Schüler, F. 1981 Biomechanical investigation to determine physical and traumatological differentiation criteria for the maximum load capacity of head and vertebral column with and without protective helmet under effects of impact. *EEC Research Program on Biomechanics of Impacts, Final report, Phase III, Project G* **5**.

Schneider, J. J. & Kirkpatrick, S. 2006 *Stochastic Optimization*. Berlin: Springer.

Scott, S. L. 2010 A modern Bayesian look at the multi-armed bandit. *Applied Stochastic Models in Business and Industry* **26** (6), 639–658.

Seo, S., Wallat, M., Graepel, T. & Obermayer, K. 2000 Gaussian process regression: Active data selection and test point rejection. In *Mustererkennung 2000*, pp. 27–34. Springer.

Seshadri, P., Duncan, A., Thorne, G., Parks, G. & Girolami, M. 2020 Bayesian assessments of aeroengine performance. *arXiv preprint arXiv:2011.14698* .

Silverman, B. W. 1985 Some aspects of the spline smoothing approach to non-parametric regression curve fitting. *Journal of the Royal Statistical Society: Series B (Methodological)* **47** (1), 1–21.

Singh, P., Couckuyt, I., Elsayed, K., Deschrijver, D. & Dhaene, T. 2017 Multi-objective geometry optimization of a gas cyclone using triple-fidelity co-Kriging surrogate models. *Journal of Optimization Theory and Applications* **175** (1), 172–193.

Smith, A., Lovelock, J. & Parkes, A. 1954 Resuscitation of hamsters after supercooling or partial crystallization at body temperatures below 0°C. *Nature* **173** (4415), 1136–1137.

Smith, J. Q. 2010 *Bayesian Decision Analysis: Principles and Practice*. Cambridge University Press.

Snoek, J., Larochelle, H. & Adams, R. P. 2012 Practical Bayesian optimization of machine learning algorithms. *arXiv preprint arXiv:1206.2944* .

Sobol', I. M. 1993 Sensitivity estimates for nonlinear mathematical models. *Mathematical Modelling and Computational Experiments* **1** (4), 407–414.

Solin, A. & Särkkä, S. 2020 Hilbert space methods for reduced-rank gaussian process regression. *Statistics and Computing* **30** (2), 419–446.

Srinivas, N., Krause, A., Kakade, S. M. & Seeger, M. 2009 Gaussian process optimization in the bandit setting: No regret and experimental design. *arXiv preprint arXiv:0912.3995* .

Stan Development Team 2020 RStan: the R interface to Stan. R package version 2.21.2.

Stein, A. & Corsten, L. 1991 Universal Kriging and cokriging as a regression procedure. *Biometrics* pp. 575–587.

Stein, M. 1987 Large sample properties of simulations using Latin hypercube sampling. *Technometrics* **29** (2), 143–151.

Svalova, A., Helm, P., Prangle, D., Rouainia, M., Glendinning, S. & Wilkinson, D. J. 2021 Emulating computer experiments of transport infrastructure slope stability using Gaussian processes and Bayesian inference. *Data-Centric Engineering* **2**.

Swallow, B., Birrell, P., Blake, J., Burgman, M., Challenor, P., Coffeng, L. E., Dawid, P., De Angelis, D., Goldstein, M., Hemming, V. *et al.* 2022 Challenges in

estimation, uncertainty quantification and elicitation for pandemic modelling. *Epidemics* **38**, 100547.

Swersky, K., Snoek, J. & Adams, R. P. 2013 Multi-task Bayesian optimization. In *Advances in Neural Information Processing Systems* (ed. C. Burges, L. Bottou, M. Welling, Z. Ghahramani & K. Weinberger), , vol. 26, pp. 1–9. Curran Associates, Inc.

Syed, Z., Shabarchin, O. & Lawryshyn, Y. 2020 A novel tool for Bayesian reliability analysis using AHP as a framework for prior elicitation. *Journal of Loss Prevention in the Process Industries* **64**, 104024.

Teymur, O., Foley, C., Breen, P., Karvonen, T. & Oates, C. J. 2021 Black box probabilistic numerics. In *Advances in Neural Information Processing Systems* (ed. M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang & J. W. Vaughan), , vol. 34, pp. 23452–23464. Curran Associates, Inc.

Thompson, W. R. 1933 On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika* **25** (3/4), 285–294.

Tracht, K., Westerholt, J. & Schuh, P. 2013 Spare parts planning for offshore wind turbines subject to restrictive maintenance conditions. *Procedia Cirp* **7**, 563–568.

Tresidder, E., Zhang, Y. & Forrester, A. 2012 Acceleration of building design optimisation through the use of Kriging surrogate models. *Proceedings of Building Simulation and Optimization* pp. 1–8.

Tripathy, R. K. & Bilionis, I. 2018 Deep uq: Learning deep neural network surrogate models for high dimensional uncertainty quantification. *Journal of Computational Physics* **375**, 565–588.

Tusar, M. I. H. & Sarker, B. R. 2022 Spare parts control strategies for offshore wind farms: a critical review and comparative study. *Wind Engineering* **46** (5), 1629–1656.

Vernon, I., Goldstein, M. & Bower, R. 2014 Galaxy formation: Bayesian history matching for the observable universe. *Statistical Science* **29** (1), 81–90.

Vernon, I., Goldstein, M., Bower, R. G. *et al.* 2010 Galaxy formation: a Bayesian uncertainty analysis. *Bayesian Analysis* **5** (4), 619–669.

Vernon, I. & Gosling, J. P. 2022 A Bayesian Computer Model Analysis of Robust Bayesian Analyses. *Bayesian Analysis Advanced Publication* pp. 1 – 33.

Vernon, I., Jackson, S. E. & Cumming, J. A. 2019 Known boundary emulation of complex computer models. *SIAM/ASA Journal on Uncertainty Quantification* **7** (3), 838–876.

Vernon, I., Owen, J., Aylett-Bullock, J., Cuesta-Lazaro, C., Frawley, J., Quera-Bofarull, A., Sedgewick, A., Shi, D., Truong, H., Turner, M. *et al.* 2022*a* Bayesian emulation and history matching of JUNE. *medRxiv* .

Vernon, I., Owen, J. & Carter, J. 2022*b* Bayesian emulation for computer models with multiple partial discontinuities. *arXiv preprint arXiv:2210.10468* .

Volodina, V. & Challenor, P. 2021 The importance of uncertainty quantification in model reproducibility. *Philosophical Transactions of the Royal Society A* **379** (2197), 20200071.

Volodina, V. & Williamson, D. 2020 Diagnostics-driven nonstationary emulators using kernel mixtures. *SIAM/ASA Journal on Uncertainty Quantification* **8** (1), 1–26.

Wang, X., Nott, D. J., Drovandi, C. C., Mengersen, K. & Evans, M. 2018 Using history matching for prior choice. *Technometrics* **60** (4), 445–460.

Wheatcroft, E., Wynn, H., Dent, C. J., Smith, J. Q., Copeland, C. L., Ralph, D. & Zachary, S. 2019 The scenario culture. *arXiv preprint arXiv:1911.13170* .

White, J. T. 2018 A model-independent iterative ensemble smoother for efficient history-matching and uncertainty quantification in very high dimensions. *Environmental Modelling & Software* **109**, 191–201.

Williams, C. J., Wilson, K. J. & Wilson, N. 2021 A comparison of prior elicitation aggregation using the classical method and SHELF. *Journal of the Royal Statistical Society: Series A (Statistics in Society)* **184** (3), 920–940.

Williamson, D. & Goldstein, M. 2012 Bayesian policy support for adaptive strategies using computer models for complex physical systems. *Journal of the Operational Research Society* **63** (8), 1021–1033.

Williamson, D., Goldstein, M., Allison, L., Blaker, A., Challenor, P., Jackson, L. & Yamazaki, K. 2013 History matching for exploring and reducing climate model parameter space using observations and a large perturbed physics ensemble. *Climate dynamics* **41** (7), 1703–1729.

Williamson, D. & Vernon, I. 2013 Efficient uniform designs for multi-wave computer experiments. *arXiv preprint arXiv:1309.3520* .

Williamson, D. B., Blaker, A. T. & Sinha, B. 2017 Tuning without over-tuning: parametric uncertainty quantification for the NEMO ocean model. *Geoscientific Model Development* **10** (4), 1789–1816.

Wilson, A., Dent, C. & Goldstein, M. 2018*a* Quantifying uncertainty in wholesale electricity price projections using Bayesian emulation of a generation investment model. *Sustainable Energy, Grids and Networks* **13** (C), 42–55.

Wilson, A. & Nickisch, H. 2015 Kernel interpolation for scalable structured gaussian processes (kiss-gp). In *Proceedings of the 32nd International Conference on Machine Learning* (ed. F. Bach & D. Blei), *Proceedings of Machine Learning Research*, vol. 37, pp. 1775–1784. Lille, France: PMLR.

Wilson, J. T., Hutter, F. & Deisenroth, M. P. 2018*b* Maximizing Acquisition Functions for Bayesian Optimization. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, p. 9906–9917. Red Hook, NY, USA: Curran Associates Inc.

Wilson, J. T., Moriconi, R., Hutter, F. & Deisenroth, M. P. 2017 The reparameterization trick for acquisition functions. In *NIPS Workshop on Bayesian Optimization*.

Wilson, K. J. & Farrow, M. 2021 Assurance for sample size determination in reliability demonstration testing. *Technometrics* **63** (4), 523–535.

Wilson, K. J., Henderson, D. A. & Quigley, J. 2018*c* Emulation of utility functions over a set of permutations: sequencing reliability growth tasks. *Technometrics* **60** (3), 273–285.

Wu, J., Poloczek, M., Wilson, A. G. & Frazier, P. 2017 Bayesian optimization with gradients. In *Advances in Neural Information Processing Systems* (ed. I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan & R. Garnett), , vol. 30, pp. 1–12. Curran Associates, Inc.

Xu, M., Wilson, A. & Dent, C. 2016 Calibration and sensitivity analysis of long-term generation investment models using Bayesian emulation. *Sustainable Energy, Grids and Networks* **5** (C), 58–69.

Yates, S. & Warrick, A. 1987 Estimating soil water content using cokriging. *Soil Science Society of America Journal* **51** (1), 23–30.

Zhang, B., Cole, D. A. & Gramacy, R. B. 2021 Distance-distributed design for gaussian process surrogates. *Technometrics* **63** (1), 40–52.

Zhang, B., Gramacy, R. B., Johnson, L. R., Rose, K. A. & Smith, E. 2022 Batch-sequential design and heteroskedastic surrogate modeling for delta smelt conservation. *The Annals of Applied Statistics* **16** (2), 816–842.

Zhang, Q., Ishihara, K. N., Mclellan, B. C. & Tezuka, T. 2012 Scenario analysis on future electricity supply and demand in Japan. *Energy* **38** (1), 376–385.

Zitrou, A., Bedford, T. & Walls, L. 2016 A model for availability growth with application to new generation offshore wind farms. *Reliability Engineering and System Safety* **152** (C), 83–94.

Zitrou, A., Bedford, T. & Walls, L. 2021 Modeling epistemic uncertainty in offshore wind farm production capacity to reduce risk. *Risk Analysis* .

Zitrou, A., Bedford, T., Walls, L., Wilson, K. & Bell, K. 2013 Availability growth and state-of-knowledge uncertainty simulation for offshore wind farms. In *22nd ESREL conference 2013*.