Emulation of Stochastic Computer Models with an Application to Building Design

Submitted by

Evan Baker

to the University of Exeter as a thesis for the degree of

Doctor of Philosophy in Mathematics

In January 2021

I certify that all material in this thesis which is not my own work has been identified and that no material has previously been submitted and approved for the award of a degree by this or any other University.

Signature  .....

Abstract

Computer models are becoming increasingly common in many areas of science and engineering; aiding in the understanding of scientific processes and providing critical information for important decisions. These models often rely on complex mathematics, and they can use a large amount of computing power to perform a single simulation. This greatly limits the usefulness of these models, because many simulations are needed for most real world problems.

A common solution to this is to train a second, statistical, model using a small set of initial simulations. This statistical model is simpler and quicker to run (and is often known as an emulator, with the computer model known as a simulator). This may initially seem like a convoluted approach, but it has shown great promise and continues to gain use in practice.

Properly designing emulators often depends on properties of the simulator in question, and so tailoring emulators to the specific problem at hand is essential. Stochastic simulators are one type of computer model which give randomly different outputs each time they are run, even if they are run for the exact same scenario (i.e. the exact same inputs). This thesis deals primarily with stochastic simulators, and how to build and use emulators for these. These simulators can be very difficult to build emulators for, as the emulator will need to learn both the underlying trends and the structure of the randomness.

This thesis also uses the engineering design of buildings to exemplify some of the issues and the developed techniques. Before a building is made, engineers can simulate different properties of the building (such as its internal temperature), and use that to make modifications to the design. We argue in this thesis that this process should be done stochastically, modifying the simulators to produce random outputs as a result of the random nature of weather (which affects the internal properties of any building). This then provides motivation for the stochastic emulation techniques

and also acts as an interesting case-study.

Outside of these two guiding objectives, the first three chapters in this thesis (after the introduction) can generally be read independently: we develop techniques for checking the quality of a stochastic emulator; we develop a methodology for improved stochastic emulation by using deterministic (non-stochastic) simulations; and we propose a framework for deciding on an acceptable building design. The remaining chapters then discuss some attributes of specific emulation techniques, and provide concluding remarks.

Acknowledgements

I obviously want to thank my supervisor, Peter Challenor, for putting up with (and answering) my unending questions. If I can call myself a statistician now, it is because of your help. I also want to thank my second supervisor, Matt Eames, for also answering my many questions and providing help when needed. I'd also like to thank Danny Williamson, who is probably responsible for getting me interested in research in the first place.

I also have to thank the PhDs at Exeter who started before me: Victoria, Louise, and Wenzhe. You all seemed to know what you were doing and were so relaxed, which was very reassuring. Gossip was (and is) also very fun. Everyone else in Laver also deserves a big thankyou. Whether this be simultaneously talking about video games and statistics with Oliver, chatting over cake with climate and stats researchers, or going for long walks with the people up in 901; you all made me feel part of a community, which was great.

I will always be appreciative of my Mum and Dad, who have always been supportive. Femke has also had to put up with me being grumpy at times, and she still supported me, which I am very grateful for. It's probably not sensible for these acknowledgements to just be pages and pages of names and thankyous, but I do want to thank all my friends and family; I wouldn't get anything done without help.

Publications

At the time of submission, the following articles have been published as a result of the work done for this thesis:

- Baker, E., Challenor, P., and Eames, M. (2020a). Predicting the output from a stochastic computer model when a deterministic approximation is available. *Journal of Computational and Graphical Statistics*, 29(4):786–797
- Baker, E., Challenor, P., and Eames, M. (2021). Future proofing a building design using history matching inspired level-set techniques. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 70(2):335–350

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Building Simulation

Before constructing a building, we sometimes want to simulate certain properties of the building (for example, the expected heating bills for whoever owns the building) to check that its design is fit for purpose. Additionally, simulations can be used to help with the design of the building itself - adjusting certain properties to obtain a better design. This can be useful for both the construction of new buildings and the modification of existing buildings.

EnergyPlus (Crawley et al., 2000) is one such tool for simulating building designs. Given a building design (and other attributes such as the weather conditions), EnergyPlus calculates the energy usage of the building (as well as other properties such as the internal temperature). Like many simulation programs, EnergyPlus can take a non-trivial amount of time to run (with the specific time depending on the size and complexity of the building design). This reduces the usefulness of these tools, as more comprehensive analyses often require many simulations. A common solution (in the wider simulation community) is to construct a surrogate model for the computer model that is faster to run and captures most of the important relationships (Sacks et al., 1989; Kennedy and O'Hagan, 2001; Sullivan, 2015). To mitigate the confusion arising from talking about two distinct models, the surrogate model is often referred to as an emulator, and the computer model is often referred to as a simulator. This idea is powerful, allowing more information to be learnt from the simulator, and emulators have been built and used for building performance simulators already (Kim et al., 2013; Yan et al., 2013; Kim and Park, 2016; Kim, 2016).

With EnergyPlus (and other building performance simulators), there are often many inputs that are less interesting, but are still important. For example, the outside weather has a large impact on the specific energy performance of a building for any given year, but it is rarely of interest on its own (it is at least not an attribute of the building that could be changed or controlled). Similar arguments can be made for other inputs, such as the activity of occupants inside the building. Additionally, these ancillary inputs are often very complex and difficult to parametrise efficiently. Together, these properties mean that these inputs are often fixed in practice. For example, the weather is usually input as some 'typical' (or some reasonably extreme) fixed values (Eames, 2016; Eames et al., 2016).

Fixing these inputs does allow those inputs of greater interest to be more easily interrogated, especially if the fixed values are carefully chosen. However, a complete quantification of uncertainty is lost, and so any resulting conclusions are somewhat dubious as a result. With an emulator, one possible strategy might be to include these ancillary inputs as standard inputs, and then perform the necessary uncertainty analysis afterwards (Oakley and O'Hagan, 2002), but this can greatly increase the input dimensionality of the problem (if the ancillary inputs can be parametrised at all).

An alternative strategy, and the one followed within this work, is to randomly generate these ancillary inputs (for example, random weather generators are common tools (Richardson, 1981; Peleg et al., 2017)), and use these draws as inputs to the simulator each time it is run. This means that the simulations (when taking the entire process as a simulation) are now stochastic: repeat simulations will produce different outputs even when using the same inputs. This is a known way of creating a stochastic simulator (Marrel et al., 2012).

Using a stochastic version of EnergyPlus like this better quantifies the real uncertainties in the system, which may help reduce the observed performance gap between simulated buildings and constructed buildings (Imam et al., 2017). The downside is that stochastic systems are harder to emulate, with the emulation of stochastic simulators its own sub-field in the wider subject of uncertainty quantification (Baker et al., 2020b). As such, the focus of much of this thesis is on stochastic emulation more generally.

We do note here that, more recently, there has also been introductory work emulating a stochastic version of EnergyPlus by Wate et al. (2020). Their work

involves coupling a stochastic occupancy simulator with EnergyPlus, and then using a stochastic emulator to quantify the various uncertainties, which bears many resemblances to the initial set-up in this thesis.

1.2 Stochastic Emulation

Statistically modelling complex computer models is a widely researched idea, often revolving around the use of a Gaussian process (Sacks et al., 1989; Kennedy and O'Hagan, 2001; O'Hagan, 2006). In the deterministic literature, the widespread use of a Gaussian process is motivated by its flexibility in capturing many different possible relationships, its comprehensive and analytical quantification of uncertainty, and because it can interpolate observed simulations. Not all simulators are deterministic however, and randomness (stochasticity) can also be present. This stochasticity can be present due to approximations needed to numerically evaluate the simulator (Herbei and Berliner, 2014), because randomness is a core attribute of the simulator (often the case with agent based models, Johnson (2010)), or because certain inputs are being provided randomly (as in our case). Regardless, the key attribute of such simulators is that they provide a different output each time they are run, even if the exact same user-defined inputs are used. This makes the standard, noiseless, Gaussian process emulator used in deterministic problems unsuitable for stochastic problems, where the interpolation of simulations is undesirable.

However, with deterministic problems (and indeed, in the wider statistical modelling and machine learning literature, Rasmussen and Williams (2006)), it is common to include a 'nugget' parameter which accounts for any noise in the data. For deterministic problems, this is usually a very small value included for numerical stability reasons (Andrianakis and Challenor, 2012), although it has also been argued to provide improved prediction performance in general (Gramacy and Lee, 2012). Either way, this nugget parameter is essential for stochastic problems.

For reference, the Gaussian process model formulation, with the notation that will be used throughout this thesis, is provided below:

$$y(\mathbf{x}) \sim GP(m(\mathbf{x}), K(\mathbf{x}, \mathbf{x}') + \sigma^2), \tag{1.1}$$

where y is the simulator output for a vector of inputs $\mathbf{x}$. Formally, a Gaussian process (GP) is a stochastic process where, for every set of inputs $(\mathbf{x}_1, \ldots, \mathbf{x}_N)$ and every finite

N, the outputs $(y(\mathbf{x}_1), \ldots, y(\mathbf{x}_N))$ are multivariate normally distributed with mean $m(\mathbf{x}_1), \ldots, m(\mathbf{x}_N))$ and covariance matrix K_N with entries $K(\mathbf{x}_i, \mathbf{x}_j)$ (Rasmussen and Williams, 2006). Less formally, the mean function $m(\mathbf{x})$ captures global trends, and can be used to provide prior beliefs about the shape of the relationship; and the covariance function $K(\mathbf{x}, \mathbf{x}')$ captures local structures, providing a flexible overall shape. The σ^2 term is the nugget parameter in deterministic experiments, and represents the intrinsic variance of the simulator in stochastic experiments.

Two important assumptions are made by this model with regard to stochastic simulators. The first assumption is that the intrinsic variability of the simulator is normally distributed. The normal assumption makes sense in deterministic settings, where it only represents a modellers personal uncertainty (the epistemic uncertainty) as to what future simulations could be. In stochastic settings, the nugget parameter extends the normal assumption to also apply to the true intrinsic variance of the simulator (the aleatoric uncertainty). This isn't always valid, as sometimes there can be multi-modality, high levels of skewness, or other non-Gaussian attributes. In this thesis we (mostly) continue to make the normal assumption, as it is still relevant for our problems, and continues to be a widely made assumption (due in part to its attractive properties, the widespread prevalence of approximately normally distributed quantities in practice, and its reasonable degree of robustness). Modifications to Gaussian processes to account for non-normal output also exist, whether this involves using a latent Gaussian process with a different likelihood for the data (Rasmussen and Williams, 2006), or using Gaussian processes to help non-parametrically model other distributions (Plumlee and Tuo, 2014; Fadikar et al., 2018).

The second important assumption made is that the intrinsic variance σ^2 is constant for all $\mathbf{x}$. This can also be a strange assumption to make, despite its prevalence throughout standard statistics (for example, with linear regression). The Gaussian process structure in Equation (1.1) can provide a flexible shape for the mean of a stochastic simulator, but it is constrained to only a single constant value for the variance of the stochastic simulator. For complex systems, as are often represented with simulators, this assumption can be invalid.

Boukouvalas et al. (2014a) consider a simple parametric form for the intrinsic variance. Whilst variance processes might (be believed to) be simpler, or of lesser importance, than the mean process, such simple parametric forms can still be overly

restrictive.

A more relaxed assumption, and one more intuitively consistent, is to also model the intrinsic variance with a Gaussian process. This is not a novel concept, dating back at least as far as Goldberg et al. (1998), with additional research attention since (Kersting et al., 2007; Boukouvalas and Cornford, 2009; Ankenman et al., 2010; Binois et al., 2018). Mathematically, this idea can be written as:

$$\begin{aligned} y(\mathbf{x}) &\sim GP\big(m(\mathbf{x}), K(\mathbf{x}, \mathbf{x}') + \sigma^2(\mathbf{x})\big); \\ \log(\sigma^2(\mathbf{x})) &\sim GP(m_\sigma(\mathbf{x}), K_\sigma(\mathbf{x}, \mathbf{x}')), \end{aligned} \tag{1.2}$$

where the logarithm of the intrinsic variance is modelled as a Gaussian process instead of the intrinsic variance itself to enforce positivity. Prediction for new points X^*, conditional on observed simulations $\mathbf{y}$ at inputs X, can then be obtained using the standard Gaussian process predictive equations (Rasmussen and Williams, 2006):

$$\begin{aligned} y(X^*) \mid \mathbf{y}, \sigma^2(X^*), \sigma^2(X) &\sim N\big(\mathcal{M}(X^*), \mathcal{V}(X^*)\big); \\ \mathcal{M}(X^*) &= m(X^*) + K(X^*, X)\big(K(X, X) + \sigma^2(X)I\big)^{-1}\big(\mathbf{y} - m(X)\big), \\ \mathcal{V}(X^*) &= K(X^*, X^*) + \sigma^2(X^*)I - K(X^*, X)\big(K(X, X) + \sigma^2(X)I\big)^{-1}K(X, X^*), \end{aligned} \tag{1.3}$$

where $K(X^*, X)$ is a matrix whose (i, j) entry equals the covariance between the i^{th} row of X^* and the j^{th} row of X (and similarly for $K(X, X)$, $K(X^*, X^*)$, and later covariance functions and matrices)), $m(X^*)$ is a vector whose i^{th} entry equals the mean function value for the i^{th} row of X^* (and similarly for $m(X)$, and later mean functions and matrices), and I is the identity matrix. The predictions for the intrinsic variance at new points can be obtained similarly:

$$\begin{aligned} \log(\sigma^2(X^*)) \mid \log(\sigma^2(X)) &\sim N\big(\mathcal{M}_\sigma(X^*), \mathcal{V}_\sigma(X^*)\big); \\ \mathcal{M}_\sigma(X^*) &= m_\sigma(X^*) + K_\sigma(X^*, X)K_\sigma(X, X)^{-1}\big(\log(\sigma^2(X)) - m_\sigma(X)\big), \\ \mathcal{V}_\sigma(X^*) &= K_\sigma(X^*, X^*) - K_\sigma(X^*, X)K_\sigma(X, X)^{-1}K_\sigma(X, X^*). \end{aligned} \tag{1.4}$$

Fitting (and predicting with) this model relies on values for the intrinsic variance at the input points ($\sigma^2(X)$), which are usually unknown. If a simulation is replicated many times, then the sample variance at the simulated input point x_i can be used as a substitute for the intrinsic variance at that point ($\sigma^2(x_i)$), ignoring any uncertainty in this estimate. This is the strategy taken in Ankenman et al. (2010), with maximum likelihood estimation used for the remaining Gaussian process parameters. This

approach is commonly known as stochastic Kriging. The main issue with this strategy is that many replicates are needed to obtain accurate sample estimates for the values of $\sigma^2(x_i)$ (in their illustrative example, Ankenman et al. (2010) start with 20 replicates at each point).

A different strategy is to treat the values of the intrinsic variance at the training input points as a set of additional parameters which need to be learnt.

Goldberg et al. (1998) does this in a fully Bayesian way, using a Markov chain Monte Carlo (MCMC) sampler. This approach can be effective, and does not need replicated simulations to work. Whilst this fully Bayesian approach quantifies the uncertainty in the variance parameters, MCMC for this complex model can be prohibitively slow in practice.

Point estimates can instead be used for these variance parameters, which loses the complete quantification of uncertainty, but provides increased computational efficiency. In Chapters 3 and 4, these variance parameters (and all other Gaussian process hyperparameters) are taken as their maximum a posteriori (MAP) estimates, learnt using the optimizing function in Stan (Stan Development Team, 2016).

`hetGP` (Binois and Gramacy, 2017) provides R software for this model structure, using the implementation from Binois et al. (2018). Their implementation again uses point estimates for variance parameters, structuring the intrinsic variance values at the training inputs to instead only be the predictive mean of a Gaussian process on different latent variables.

More specifically, they express the top-level covariance function $K(\mathbf{x}, \mathbf{x}')$ as $\sigma_Z C(\mathbf{x}, \mathbf{x}')$, where σ_Z is an epistemic uncertainty term and C is a correlation function, and aim to learn values for $\lambda^2(\mathbf{x}) = \sigma^2(\mathbf{x})/\sigma_Z^2$. With the variance level covariance function $K_\sigma(\mathbf{x}, \mathbf{x}')$ expressed in the form $\sigma_g C_g(\mathbf{x}, \mathbf{x}')$ and a zero-mean for the variance level mean function $m_\sigma(\mathbf{x})$, intrinsic variance values at inputs X^* are taken as $\log(\lambda^2(X^*)) = C_g(X^*, X_n)\left(C_g(X_n, X_n) + gR^{-1}\right)^{-1}\boldsymbol{\Delta}$. Here, $\boldsymbol{\Delta}$ is a vector of hyperparameters to be learnt (of length n), g is another parameter to be learnt, and R is a diagonal matrix with the i^{th} diagonal entry equal to the number of replicate simulations for the i^{th} training input. The added gR^{-1} term encourages smooth estimates for the intrinsic variances. This formulation provides a fixed functional form for the intrinsic variances, but again does not quantify the uncertainty in the latent variance parameters.

The `hetGP` package also makes use of simplified predictive mean, predictive

variance, and likelihood equations, which are derived to only require the inversion of an $n \times n$ matrices (instead of the usual $N \times N$), facilitating computational speed-ups when replicates are present (but still not requiring them). Maximum likelihood estimation is then used to estimate all unknown parameters.

This implementation is used in Chapters 2 and 5.

As a general rule for stochastic emulators, even when replicates aren't required by the specific implementation, more simulations in general are often needed to obtain a good emulator, due to the noisy nature of the simulations. Loeppky et al. (2009) recommends 10 simulations per input dimension for deterministic simulators, whereas in an investigation between different design considerations, Binois et al. (2019) uses up to 25 times this number. More information about stochastic emulation can be found in Baker et al. (2020b).

1.3 Thesis Outline

- In Chapter 2 we outline the importance of validating an emulator after it is built. We also show how current diagnostic tools developed for deterministic emulators are not always well-suited to emulators of stochastic models. We then develop new diagnostic tools developed specifically for the stochastic setting, which use replicated simulations to validate the mean predictions and variance estimates separately.

- In Chapter 3 we develop the idea that a deterministic simulator and a stochastic simulator for the same system can be linked together, providing an overall improved emulator. EnergyPlus has both a deterministic version (the default) and a stochastic version (where the ancillary input is randomised), providing the motivation for this method. Stochastic simulators typically demand additional simulation, compared to deterministic simulators, to provide accurate conclusions. Our developed method mitigates this problem, allowing capable emulators to still be built when the simulator is complex and the simulation budget is small.

- In Chapter 4 we return to the building performance simulator. Here, we provide a case study, showing how a practitioner could efficiently find a set of building designs which meet some required criteria. The developed methodology used is

a modified version of history matching, iteratively homing in on the acceptable buildings. This goal is different to optimisation, as a set of acceptable buildings are found, rather than just one, allowing a practitioner to choose a final building based on other preferences (such as its appearance or ease of construction).

- Chapter 5 is more open-ended, discussing history matching more generally, experimenting with the different options available, and comparing to alternatives.
- Chapter 6 then provides concluding remarks.

Chapter 2

Diagnostics for Stochastic Emulators

2.1 Introduction

Consider the toy stochastic simulator in Equation (2.1)

$$\begin{aligned} y(x) &= \sin(16x) + \cos(24x) + 8x + \epsilon; \\ \epsilon &\sim N\big(0, (0.1 + 0.9x)^2\big). \end{aligned} \tag{2.1}$$

Figure 2.1 shows two sets of simulated data for this simulator, and their respective emulator fits using the `hetGP` package.

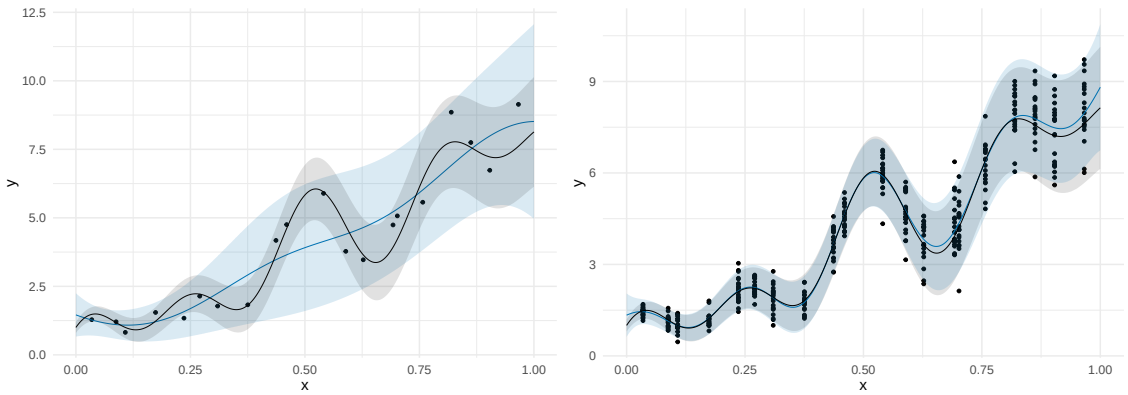


Figure 2.1: Two heteroscedastic Gaussian process emulators, fit to two different datasets, both from the same stochastic simulator. Blue represents the mean and 2 standard deviation intervals from the emulator. Black represents the truth.

Also shown is the truth. It should be immediately apparent that the emulator on the left is substantially less useful than the emulator on the right. The emulator

on the right appears to be a fairly good substitute for the simulator, accurately capturing the mean and variance. The emulator on the left however, fails to capture any of the nuance in the mean, and generally overestimates the variance. In fact, as figure 2.2 shows, the uncertainty intervals around the mean are also inaccurate, as the true mean is regularly outside the emulator's 95% prediction intervals for the mean.

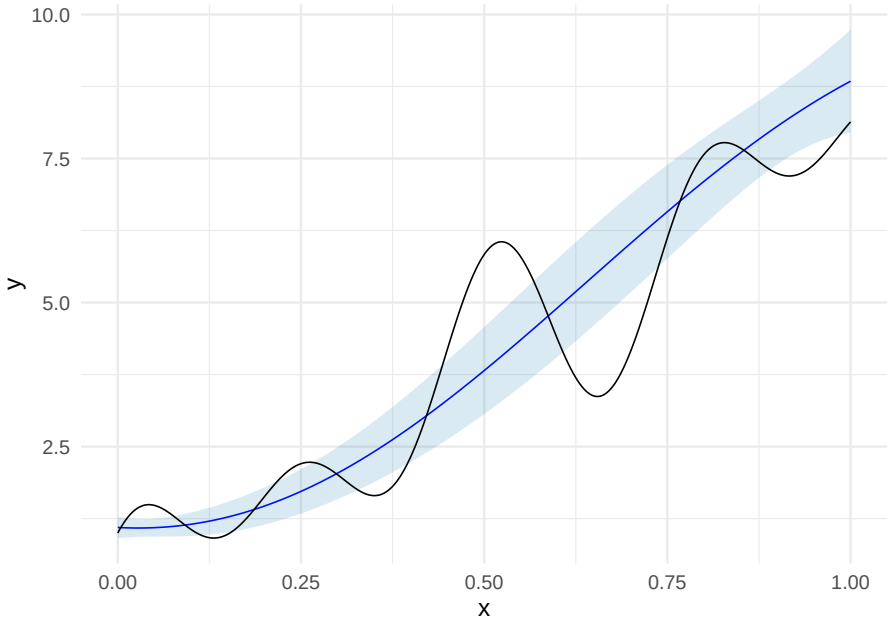


Figure 2.2: The mean and 2 standard deviation prediction intervals for the mean from the 'bad' emulator from the left of Figure 2.1 (in blue). Also superimposed is the true mean of the simulator (in black).

This means that, not only is the emulator on the left less useful than the emulator on the right, but it is also inaccurate. This type of emulator flaw, where an overestimated intrinsic variance leads to a poor estimation of the mean, is not a new discovery, and it has already been documented in the deterministic emulation literature where the inclusion of intrinsic variance (for numerical reasons) can cause this issue (Andrianakis and Challenor, 2012).

Whilst details depend on the specific application and goal, using the emulator on the left could lead to incorrect conclusions, especially if the mean-variance structure is important. It is therefore important that, after the construction of an emulator, tests are done to ensure that the emulator is fit for purpose.

In practice, identifying flaws can be difficult. Problems are often of higher dimension, and access to the truth is rarely (if ever) available, and so simple plots

like those above cannot be used to check the quality of an emulator. However, other checks can be done using a set of 'validation data' (i.e. a set of data from the simulator, preferably not the same set used to fit the emulator), to assess the quality of an emulator. The more the emulator's predictions and the validation data match up, the better confidence one can have in the emulator.

For example, standardised errors can be calculated and used to check for local emulator misspecifications (Bastos and O'Hagan, 2009). The standardised errors are equal to:

$$\frac{y_i^* - \mathcal{M}(\mathbf{x}_i^*)}{\sqrt{\mathcal{V}(\mathbf{x}_i^*)}}, \tag{2.2}$$

where y_i^* is a validation simulation for the inputs $\mathbf{x}_i^*$. These should be normally distributed with mean zero and variance 1, and so, as a rule-of-thumb, any points with absolute value larger than 2 should be investigated as potential emulator flaws. Plotting these against each input dimension can also be used to identify any trends or local patterns.

Bastos and O'Hagan (2009) also provide modified standardised errors which account for the correlation between points, such as the Cholesky errors, which can be calculated as:

$$G^{-1}(\mathbf{y}^* - \mathcal{M}(X^*)), \tag{2.3}$$

where $\mathbf{y}^*$ is the set of validation simulations, for inputs X^*, and G^{-1} is the standard deviation matrix obtained from performing pivoted Cholesky decomposition (such that $P^T\mathcal{V}(X^*)P = GG^T$, G is an upper triangular matrix, and P is a permutation matrix). These no longer have the same interpretation when plotted against input variables, although other insights are possible (for example, when plotted against the pivoting order, large (or small) values to the left of the plot indicate poor variance estimation and large (or small) values to the right indicate overestimation (or underestimation) of the correlation parameters (Bastos and O'Hagan, 2009).

We can also use the standardised errors to plot a QQ plot, which should appear close to a 45 degree line if the emulator is accurate.

Bastos and O'Hagan (2009) also suggest using the Mahalanobis distance (Equation (2.4)) to score an emulator's predictions. Should the observed distance appear too large or too small, then confidence in the emulator is decreased.

$$(\mathbf{y}^* - \mathcal{M}(X^*))^T(\mathcal{V}(X^*))^{-1}(\mathbf{y}^* - \mathcal{M}(X^*)). \tag{2.4}$$

Additionally, Al-Taweel (2018) developed a similar diagnostic for checking the overall fit of the emulator via the credible interval diagnostic. With this, for a given α, the $100\alpha\%$ central credible interval can be obtained from the emulator, and the number of validation simulations which lie within this interval can be calculated (and should be close to $100\alpha\%$). This can be repeated for many values of α. Plotting these percentages against α should then also look like a 45 degree straight line. Samples drawn from the emulator can be used to obtain confidence bands for each credible interval statistic, and these can be superimposed onto the plot.

After obtaining 50 validation data points from the toy simulator in Equation (2.1), the diagnostic plots in Figure 2.3 can be obtained for the 'bad' emulator from before.

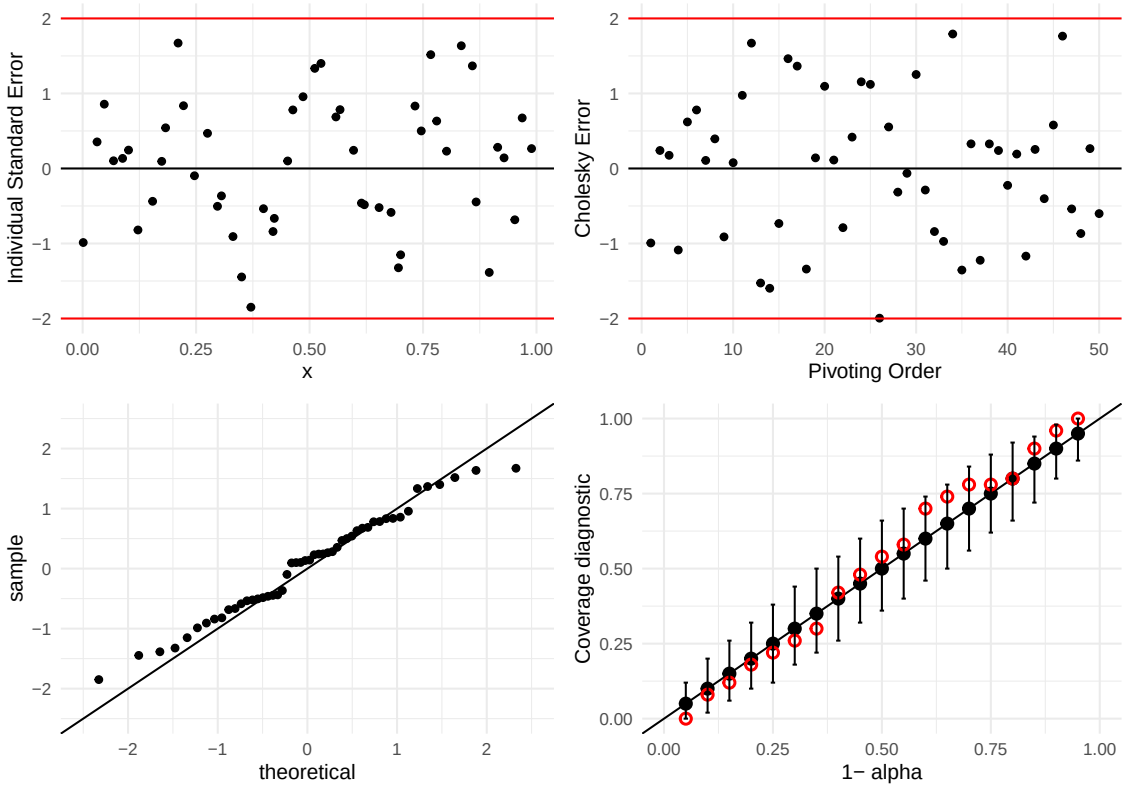


Figure 2.3: Four diagnostic plots from Bastos and O'Hagan (2009) and Al-Taweel (2018) for the 'bad' emulator. The top left is a plot of individual standardised errors against x; top right is a plot of the pivoted Cholesky errors against the pivoting order; bottom left is a QQ plot of the standardised errors; bottom right is a credible interval diagnostic plot.

The Mahalanobis distance is equal to 44.95 and its reference interquartile range (obtained by sampling data points from the emulator and calculating the Mahalanobis

distance for them) is (43.25 55.94). In general, these diagnostics do not flag anything wrong with the 'bad' emulator. A keen eye might notice that none of the standardised errors have a larger absolute value than 2, which, with 50 validation points, is mild evidence towards the total emulator uncertainty being too large, but small standardised errors are often considered to be a good result. Additionally, one might also notice some mild structure in the individual standardised errors, but this can often be the case in good fitting 1D emulators.

As a reference point, Figure 2.4 presents these diagnostics, but for the 'good' emulator.

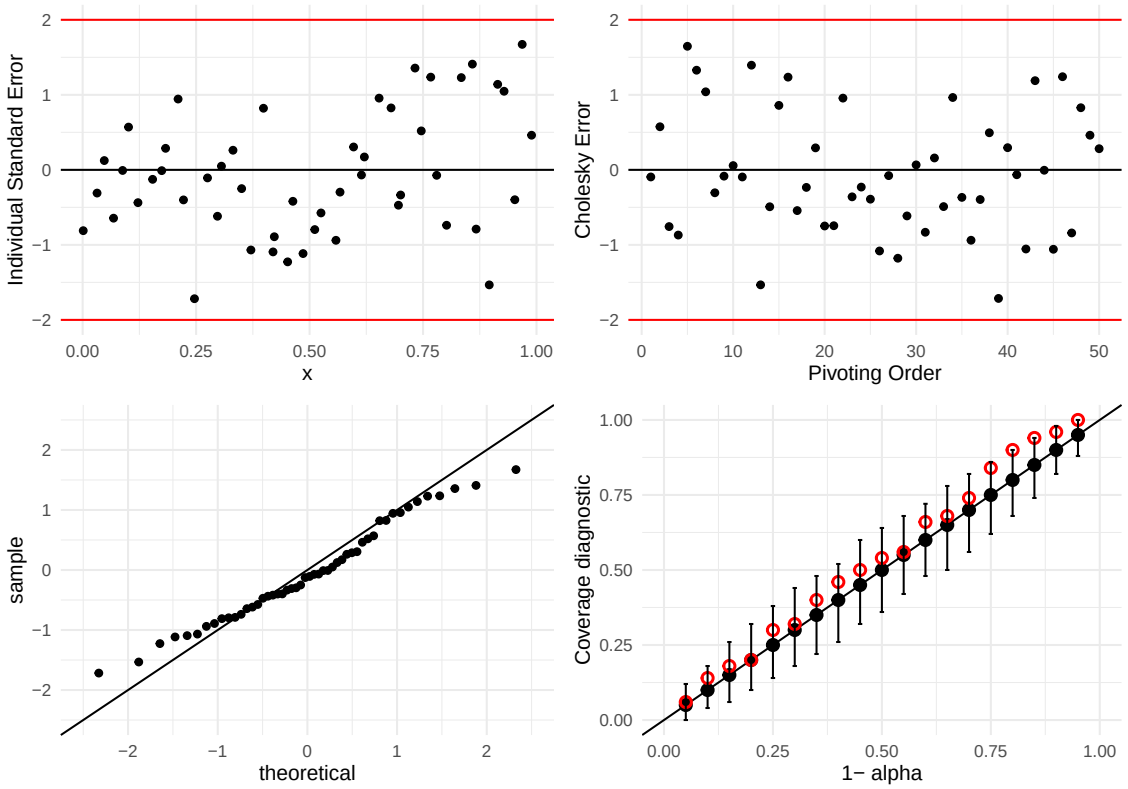


Figure 2.4: The four diagnostic plots from Bastos and O'Hagan (2009) and Al-Taweel (2018) for the 'good' emulator. The top left is a plot of individual standardised errors against x; top right is a plot of the pivoted Cholesky errors against the pivoting order; bottom left is a QQ plot of the standardised errors; bottom right is a credible interval diagnostic plot.

The Mahalanobis distance is equal to 33.09 and its reference interquartile range is (42.90, 56.32). The 'good' diagnostic results are essentially indistinguishable from the 'bad' diagnostic results. In fact, these seem slightly worse because the Mahalanobis distance is outside the reference interval for the 'good' emulator but not the 'bad'

emulator. This provides evidence that the diagnostics developed for deterministic emulators are not always appropriate for stochastic emulators. Additionally, even if the deterministic diagnostics were to identify problems (which is possible, especially with an abundance of validation data), it is not always clear how to properly interpret such results. For example; a large standardised error could signify a poorly estimated mean process, an underestimated intrinsic variance process, or both. It would be essentially impossible to recognize an acceptably accurate mean process but an overestimated intrinsic variance. The presence of intrinsic simulator variance inevitably makes diagnostic tools developed for deterministic emulators (which do not distinguish between intrinsic variance and epistemic uncertainty) less useful.

2.2 Stochastic Emulator Diagnostics

The discussion above suggests that a component-wise diagnostic procedure could be useful; that is, specifically checking the mean and variance (and normality) of a stochastic emulator separately and independently. With this framing, diagnostics developed for deterministic emulators can be viewed as diagnostics for checking the overall accuracy of an emulator.

Any diagnostic method ultimately relies heavily on the information contained within the validation data. It is not possible to obtain values of the mean and variance from a simulator (at least, not often), and so directly validating the mean process and variance process is not possible. However, insights into the mean process and variance process can be obtained by replicating simulations. Replication enables the calculation of *sample* means and *sample* variances, which can then be compared with the predictions and estimates from an emulator. The diagnostic tools developed here provide techniques for using sample means and sample variances to validate and diagnose a stochastic emulator.

We will first outline diagnostic methods for assessing the quality of the mean.

2.2.1 Assessing the Mean

Sample means cannot be directly compared with the mean of an emulator, as the sample mean will never exactly equal the true mean even with a perfect emulator (with no epistemic uncertainty), due to sampling variation (unless the simulator is

deterministic). If the intrinsic variance of the simulator is quite small, or the number of samples is very large, then this effect will be quite small. If the intrinsic variance is large, or the number of samples is small, then a sample mean can be a poor direct substitute for the true mean.

Luckily, if a random variable Z is normally distributed with mean μ and variance σ^2 ($Z \sim N\big(\mu, \sigma^2\big)$), then the distribution of the sample mean $\bar{Z}$ of n samples is known to still be normal, but with a smaller variance ($\bar{Z} \sim N\big(\mu, \sigma^2/n\big)$). This means that emulator predictions for a sample mean can be easily obtained: standard Gaussian process predictions are Gaussian (Rasmussen and Williams, 2006), and therefore, an emulator's prediction for the *sample* mean remains Gaussian, obtained via Equation (2.5):

$$\bar{y}(X^*)|\mathbf{y}, \sigma^2(X^*), \sigma^2(X) \sim N\big(\mathcal{M}(X^*), \mathcal{V}(X^*)\big); \tag{2.5}$$
$$\mathcal{M}(X^*) = m(X^*) + K(X^*, X)\big(K(X,X) + \sigma^2(X)I\big)^{-1}\big(\mathbf{y} - m(X)\big),$$
$$\mathcal{V}(X^*) = K(X^*, X^*) + \sigma^2(X^*)R^* - K(X^*, X)\big(K(X,X) + \sigma^2(X)I\big)^{-1}K(X, X^*),$$

where $\bar{y}$ is the sample mean, X^* is now the matrix of input points for the validation data, and R^* is a diagonal matrix with entries $1/r_i^*$ corresponding to the reciprocal of the number of simulator runs used to calculate the i^{th} sample mean

With this, emulator predictions for a sample mean can be obtained analytically, which can be compared to the observed sample mean. Any mismatch then implies an issue with the predicted mean process of the emulator. Because these predictions are still normally distributed, all the same diagnostic techniques from the deterministic setting remain valid, and can thus be easily re-tooled to apply to the sample means.

Obtaining different validation data for the toy stochastic simulator in Section 2.1, this time with 10 unique x values each run 5 times, we can produce the standardised error plot for the 'bad' emulator, but this time for the sample means (Figure 2.5).

This uses the same total number of validation points as the previous diagnostics did in Section 2.1, but now it is very clear that there is a problem. Out of 10 sample means, two lie outside the two standard deviation interval, which is unlikely to happen by chance. Also, one of the sample mean standardised errors has a value over three, which is an indication of a serious mismatch between the emulator and the truth. Overall, these sample mean standardised errors provide reasonable evidence that the emulator's mean is inaccurate. This is good for two reasons: firstly, they correctly identify a problem when previous diagnostics did not; secondly, they also

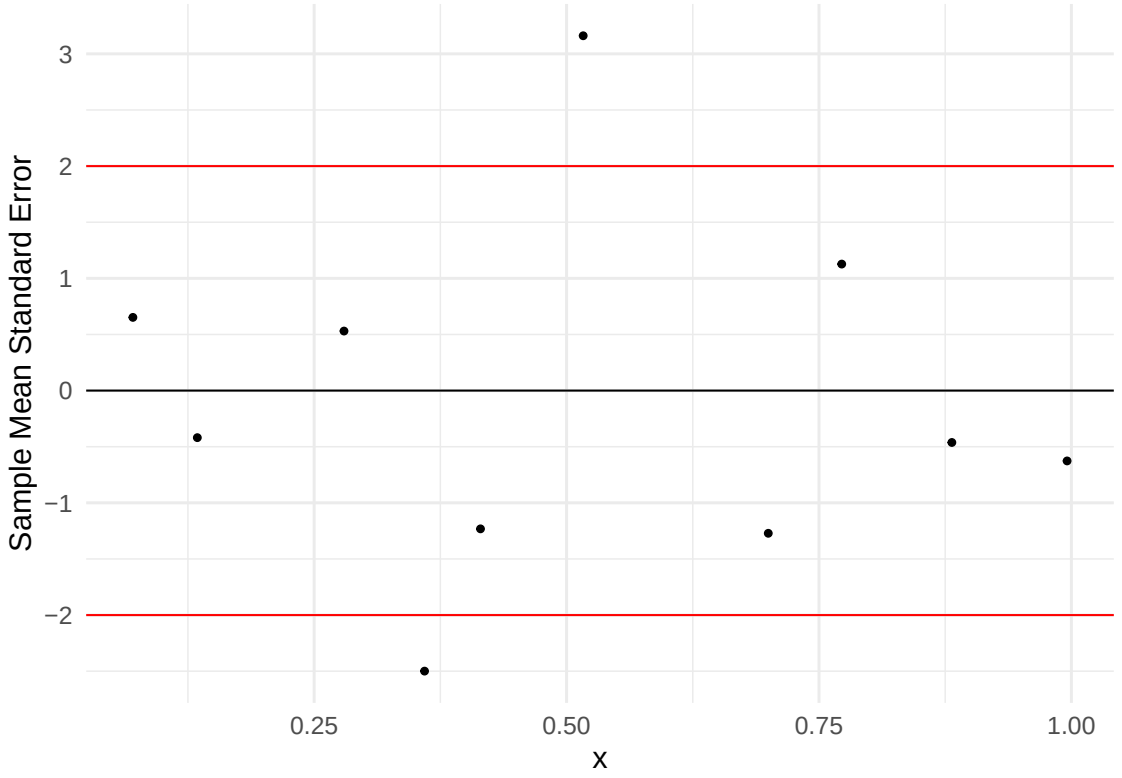


Figure 2.5: Sample mean standardised errors against x for the 'bad' emulator.

provide information as to *what* is wrong with the emulator.

With knowledge of the truth (as we have access to Figure 2.2), we can see that the two standardised errors with absolute value larger than two match up with locations where the true mean lies outside the uncertainty bound for the emulator's mean. However, there is no standardised error with absolute value larger than two around $x = 0.75$ which is another location where the truth and the emulator's mean do not match. This is because there is still residual noise in the sample means, and so not every problem has been identified, but a substantially better job has been done than before.

An issue with these mean diagnostics however, is that they assume we have knowledge of the true intrinsic variance values, whereas we use the emulator's intrinsic variance estimates as a substitute. As a result, should the emulator's intrinsic variance be estimated poorly, then errors can appear in the mean diagnostics even if there are no actual errors in the mean process, or errors may not appear in the mean diagnostics even if there are actual errors in the mean process.

For example, if the bad emulator had estimated the intrinsic variance to be five times larger than what it currently estimates, but was identical otherwise, the mean

should still be shown to be poor. However, as Figure 2.6 reveals, this is not the case.

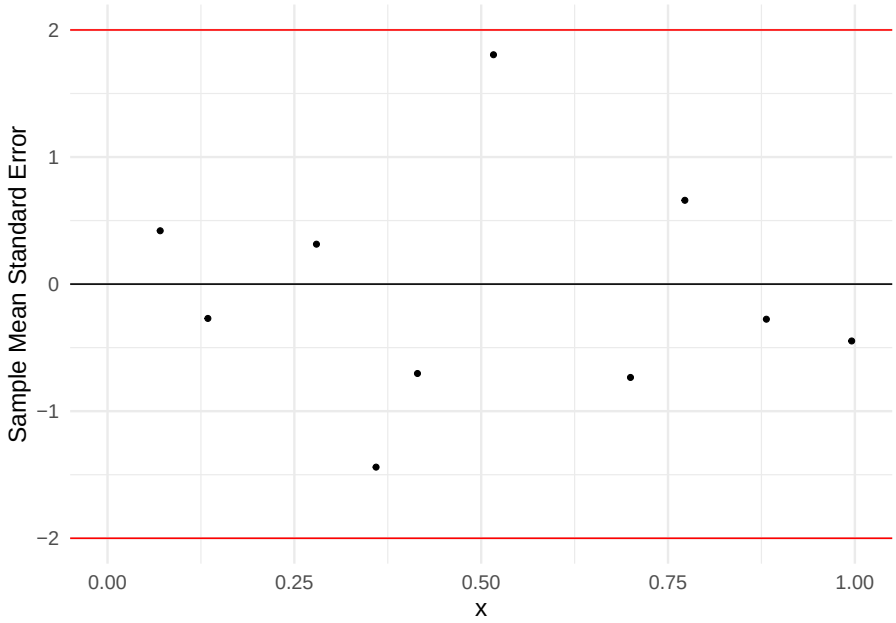


Figure 2.6: Sample mean standardised errors against x for the 'bad' emulator, if the intrinsic variance estimates were five times larger.

With a much larger estimated intrinsic variance, the natural sample variation from the observed sample means is much less than expected, and so no observed sample means are unexpectedly large or small, wrongly suggesting the mean is acceptable. This is clearly problematic, and we were lucky it did not happen naturally with the previous example. The opposite problem can also occur, with phantom problems in the mean appearing if the emulator seriously underestimates the variance.

To avoid this problem, we want to test the emulator's mean in isolation and avoid using the emulator's intrinsic variance estimates in the validation of the mean. We can assume the intrinsic variance process is unknown (rather than assuming it is known and equivalent to the emulator's intrinsic variance estimates) and instead use the sample variances from the replicated validation simulations to obtain the sample mean distributions (which can then be compared to the observed sample means).

Directly substituting the sample variances as a substitute for $\sigma^2(X^*)$ in Equation (2.5) is likely to lead to different problems, as then the mean diagnostics would depend on the accuracy of the sample variances (which could be more inaccurate than the emulator's intrinsic variance estimates). As such, the inaccuracy of the sample variances needs to be accounted for in the predictive distributions for the sample means. For normally distributed variables like this, when the true intrinsic

variance is unknown and sample variances are used as a substitute, the distribution for sample means is known and can be obtained from:

$$(\bar{y}_i(\mathbf{x}_i^*) - \mathcal{M}(\mathbf{x}_i^*))\frac{\sqrt{r_i^*}}{\hat{S}_i} \sim t_{r_i^*-1}, \tag{2.6}$$

where $\hat{S}_i^2$ is the observed sample variance (Stuart and Ord, 1994). This can then be combined with predictive samples for the mean from the emulator to obtain a new predictive distribution for the sample means.

This distribution is no longer normal (except when r_i^* is large, as then the $t_{r_i^*-1}$ distribution would tend towards the normal distribution, but we would not expect this in practice), and so comprehensive diagnostics such as the standardised errors are no longer possible. What we instead develop is an alternative to the individual standardised errors which can be applied to any distribution.

Standardised errors (as previously and in Bastos and O'Hagan (2009)) were a useful diagnostic tool, as they readily show how unexpected any given simulation is, assuming the emulator is correct. Since every standardised error should have the same distribution $(N\big(0,1\big))$, a consistent scale is obtained, making it easy to compare and identify trends and unexpected values. It is with this consistent scale in mind that we develop a new diagnostic.

If the probability distribution can be obtained for a given observation, such as a sample mean or sample variance, then the cumulative distribution should be available, and therefore one can calculate the probability of observing a value less than the realised observation. For a sample mean, that is: $P(\bar{y}(\mathbf{x}_i^*) \leq \hat{\bar{y}}(\mathbf{x}_i^*))$ where $\hat{\bar{y}}(\mathbf{x}_i^*)$ is the observed sample mean. Unexpectedly large observed sample means will have a large value of $P(\bar{y}(\mathbf{x}_i^*) \leq \hat{\bar{y}}(\mathbf{x}_i^*))$ and unexpectedly small observed sample variances will have a small value of $P(\bar{y}(\mathbf{x}_i^*) \leq \hat{\bar{y}}(\mathbf{x}_i^*))$. It is with this as a foundation that we define the "unexpectedness" U as:

$$U = 2(0.5 - P(Z \leq \hat{Z})), \tag{2.7}$$

where Z is some random quantity (e.g. the sample mean at a given coordinate), $\hat{Z}$ is an observation for that random quantity, and 2 and 0.5 are scaling factors [1]. As $P(Z \leq \hat{Z}))$ is bound between 0 and 1, U is always bound between 1 and -1. A large

[1]Note that an unexpectedness is very similar to a re-scaled p-value. We choose to use a different term to encourage a slightly different use-case, but we do acknowledge that this difference is basically arbitrary.

positive value for U corresponds with an unexpectedly small value for the observation, and thus the emulator likely overestimates Z; and a large negative value implies an unexpectedly large value for Z and thus the emulator likely underestimates Z.

Unexpectedness values can be interpreted in much the same way as individual standardised errors (albeit inverted). For normally distributed observations; any U with absolute value greater than 0.95 is (roughly) the same as any standardised error with absolute value greater than 2, as the 2 standard deviation intervals match (roughly) with the 95% prediction intervals, and the 0.95 unexpectedness intervals match exactly with the 95% prediction intervals. Similarly, any U with absolute value greater than 0.995 is very unlikely and corresponds to an absolute standard error of -2.8. Differences between the two plots in the normal case are there, especially for the extremes, as the unexpectedness should be uniformly distributed between -1 and 1, and the standardised errors should look standard normally distributed about 0, but a lot of information is conveyed in the same way. A good set of unexpectedness values would not reveal any clear trend, and there should not be too many extreme values (with the number of acceptable extreme values depending on the number of unexpectedness values: roughly 5% of values should have an absolute value larger than 0.95, 1% larger than 0.99 and so on).

Figure 2.7 plots the unexpectedness values against x for the 'bad' emulator, whilst still assuming the emulator's intrinsic variance to be correct (i.e. with normally distributed sample means).

Here, the unexpectedness looks a lot like Figure 2.5 but flipped. Two unexpectedness values have absolute value greater than 0.95 and one is smaller than -0.995, which again suggests the emulator's mean is inaccurate.

The unexpectedness has been designed in such a way that the plots are flipped compared to the standardised error plots, because now large values of U imply the emulator's mean is too large, and small values imply it is too small (similarly for the sample variance unexpectedness later), which is perhaps a more natural interpretation. If the alternative is preferred, the scale can be flipped. The downside to the unexpectedness compared to the standardised errors is that extremely unexpected values are not quite as obvious - an unexpectedness value of 0.999 is much worse than 0.95, but appears similar when plotted. As such, we instead recommend direct interrogation of extreme unexpectedness to check their value. Alternative solutions could involve transforming the unexpectedness to be normally distributed (using

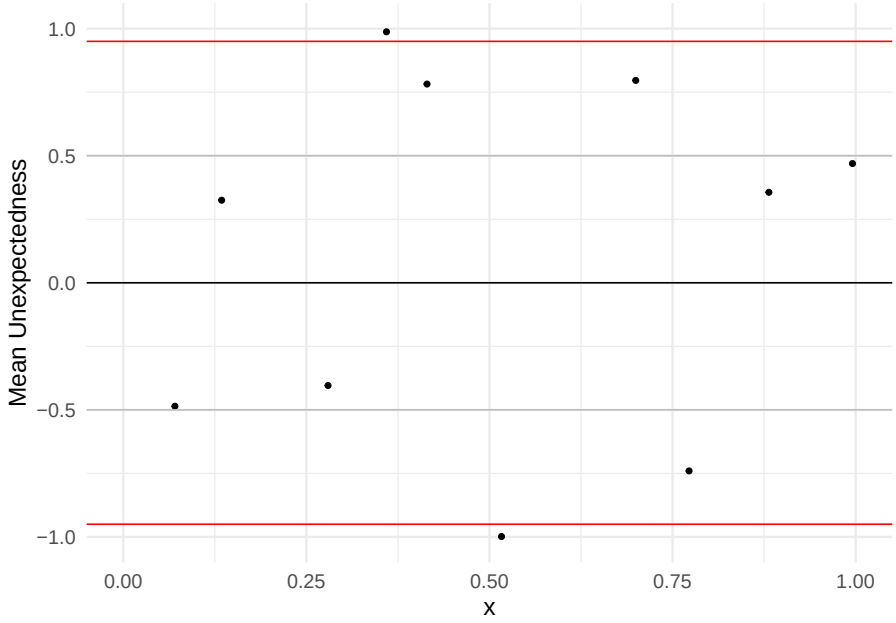


Figure 2.7: Sample mean unexpectedness against x for the 'bad' emulator, wrongly assuming the emulator's intrinsic variance estimates are correct.

the inverse normal cumulative distribution function) and so extreme values would appear as they do in standardised errors; or superimposing the values of extreme points onto plots.

Returning now to the example where the intrinsic variance is artificially inflated to 5 times is value, we can obtain the mean unexpectedness values using the sample variances (i.e., the sample means are t distributed). Figure 2.8 shows these sample mean unexpectedness values.

We can see from this plot that using the observed sample variances to calculate the sample mean distribution leads to more robust sample mean checks, as they are able to identify a poorly estimated mean process even if the emulator's intrinsic variance is large: as before, two unexpectedness values are outside the (-0.95, 0.95) interval, and one is as low as -0.999. These results are the exact same as before, because using the sample variances to calculate the sample mean distribution (and assuming t distributed variation) leads to mean diagnostics which are independent of the emulator's intrinsic variance estimates. This is a useful adjustment, as it allows us to check the emulator's mean in relative isolation. From this point onwards, this strategy is what we refer to as the sample mean unexpectedness diagnostics.

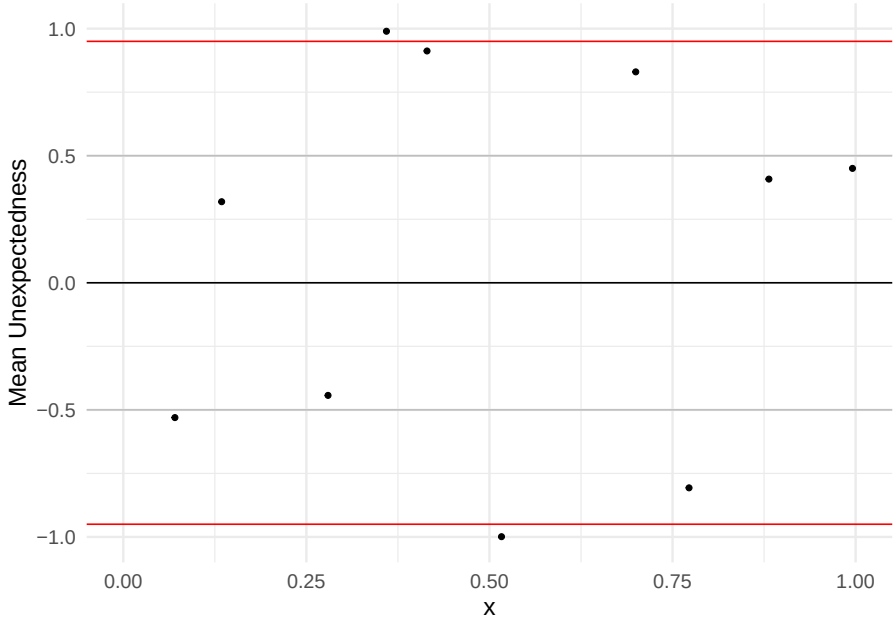


Figure 2.8: Sample mean unexpectedness against x for the 'bad' emulator if the intrinsic variance estimates were five times larger, using the sample variances to calculate the sampling distribution.

2.2.2 Assessing the Variance

With unexpectedness, diagnosing problems in the variance process can follow roughly the same procedure as with the mean: obtain the emulator predictive distribution for the sample variances and compare to the observed sample variances. For the sample variance, the sampling distribution is again known, and again not normal. Given the intrinsic variance $\sigma^2(\mathbf{x}_i^*)$, the distribution for the sample variance S^2 is known (Stuart and Ord, 1994) and can be obtained from:

$$\frac{(r_i^* - 1)S^2(\mathbf{x}_i^*)}{\sigma^2(\mathbf{x}_i^*)} \sim \chi^2_{r_i^*-1}. \tag{2.8}$$

A strength of unexpectedness is that it can be applied to almost any quantity as long as it is continuous and its distribution is known (or can be empirically estimated). As such, we can easily apply it to the sample variances, using Equation (2.8). The plot on the left in Figure 2.9 plots the sample variance unexpectedness against x for the 'bad' emulator.

From this it is clear that the emulator overestimates the variance generally - nine out of ten of the unexpectedness values are positive, and one has a value larger than 0.95 (the value is 0.9964, which is quite extreme). Together with the mean unexpectedness plot, these diagnostics reveal information that wasn't known before -

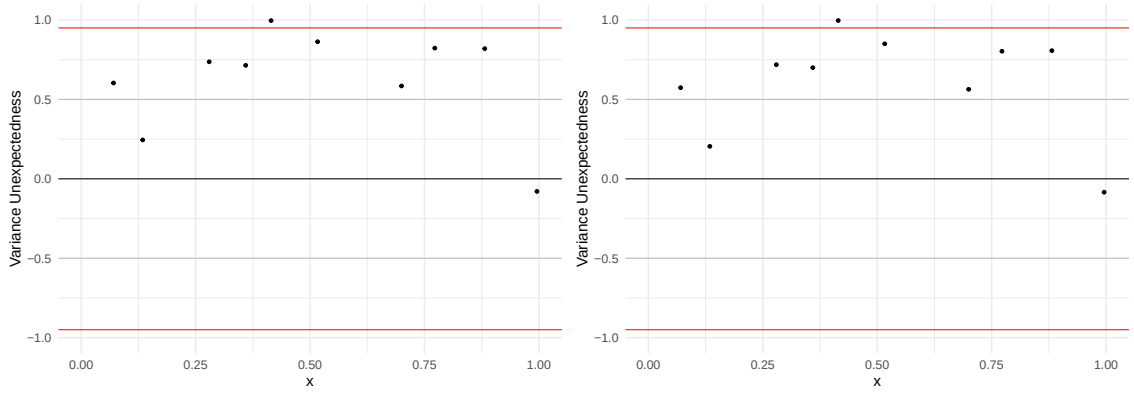


Figure 2.9: Sample variance unexpectedness against x for the 'bad' emulator. Left is the uncorrected version, right is the corrected version.

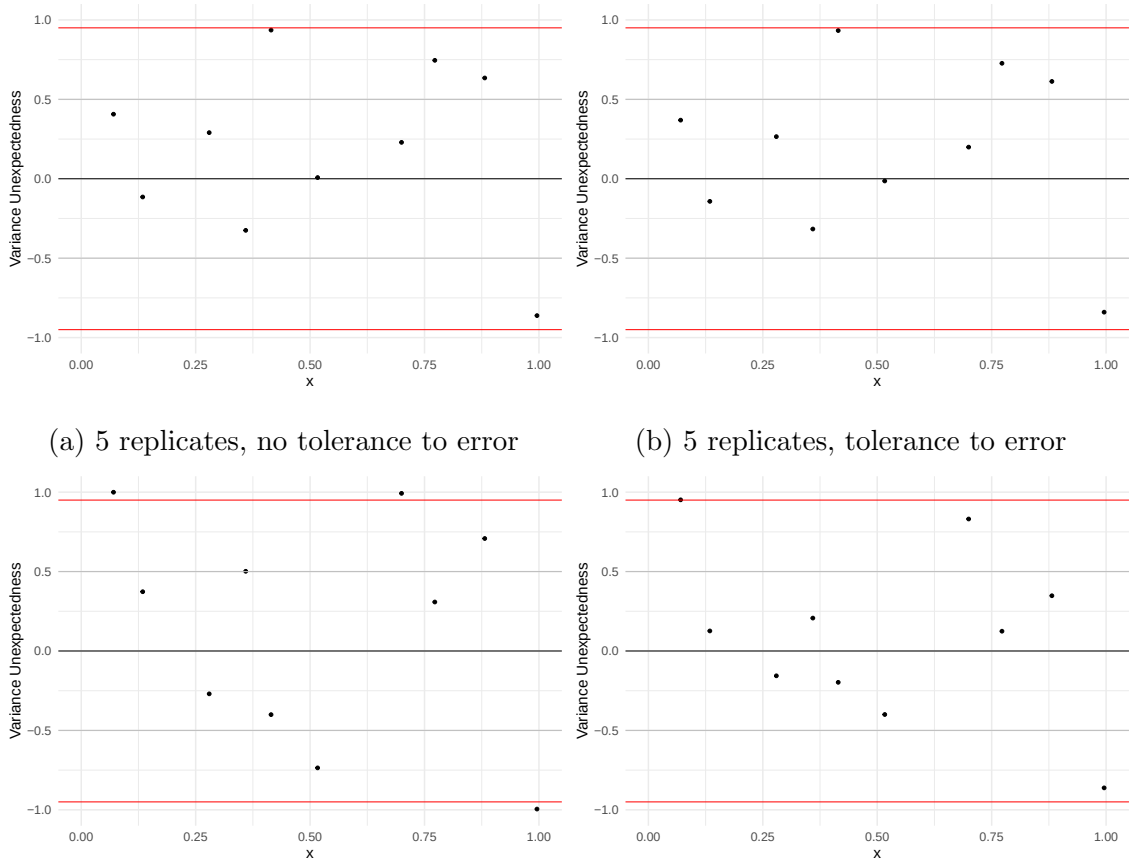
the mean is poorly estimated, and the variance is systematically overestimated.

However, there is a problem with any variance process diagnostic for heteroscedastic Gaussian process emulators: most implementations do not feature any (or have reduced) epistemic uncertainty around the variance (Kersting et al., 2007; Boukouvalas and Cornford, 2009; Binois et al., 2018). This is done for computational reasons, but it does raise an important philosophical question when it comes to validation: if there is no uncertainty around the variance estimates, does this mean every variance estimate is always wrong? Variances lie in the continuous domain, and so no estimate will ever be perfectly accurate. Without uncertainty, this means that, technically, every variance estimate is always wrong, and so we should always (given enough data), invalidate every heteroscedastic Gaussian process emulator, including the 'good' one previously.

We reject this notion, as it is not practical, in favour of a modification. We introduce a "tolerance to error" as a substitute for epistemic uncertainty around the variance. We do so by specifying a distribution that represents our tolerance to error for the variance estimates, and using that as if it were the epistemic uncertainty. We suggest substituting the standard deviation estimates with the distribution $U(0.8\sigma, 1.2\sigma)$, which corresponds with a tolerance to any true standard deviation which is within 20% of our estimate. Tolerance to error is a subjective notion, and so a practitioner might want to use a different distribution to quantify their tolerance to error. A uniform distribution was chosen here arbitrarily and implies indifference between a perfect variance estimate and an imperfect estimate that is within the specified bounds. A triangular distribution tolerance-to-error would perhaps be more

reasonable.

To clarify the practical necessity of this tolerance to error, and the effects of it, Figure 2.10 plots four sample variance unexpectedness plots.



(a) 5 replicates, no tolerance to error

(b) 5 replicates, tolerance to error

(c) 100 replicates, no tolerance to error

(d) 100 replicates, tolerance to error

Figure 2.10: Sample variance unexpectedness against x for the 'good' emulator. Top left is for 5 replicates per point and without any tolerance to error; top right is for 5 replicates per point and with tolerance to error; bottom left is for 100 replicates per point and without any tolerance to error; bottom right is for 100 replicates per point and with tolerance to error.

Each of these plots are for the same 'good' emulator, and should all imply the emulator is accurate. The first column shows two different unexpectedness plots, one for when there are five replicates in the validation data and one for when there are 100. For five replicates, no problems seem to present themselves (except perhaps that the variance might be mildly overestimated). For 100 replicates however, it seems as if the emulator's variance is poor, with two unexpectedness values larger than 0.95, and one less than -0.95. This is because, with 100 replicates, the sampling variation in the sample variance is greatly reduced, and so the effects of having no epistemic

uncertainty around the variance is revealed. Figure 2.1 showed that this emulator has an acceptable fit, and so it is problematic that this diagnostic would flag issues. The second column shows the effects of including the tolerance to error. When there are few replicates, the results remain essentially the same, as the sampling variation masks the tolerance to error. When there are many replicates, however, we no longer wrongly identify problems. The earlier plot in the right of Figure 2.9 also presents the unexpectedness plot for the 'bad' emulator including the tolerance to error. In this case, the inclusion of tolerance to error does not prevent the real issues in the 'bad' emulator from being identified. As such, the inclusion of tolerance to error seems to be a good inclusion for the variance diagnostics, improving the robustness of these tools.

2.2.3 Assessing the Normality Assumption

With stochastic simulators, normality can be a strong assumption. Checking this assumption is therefore important if one is to have any confidence in a resulting emulator. A QQ plot of the standardised residuals is a standard method for testing the normality assumption, and is also recommended by (Bastos and O'Hagan, 2009). The issue with QQ plots is that they only check for *conditional* normality; i.e. whether the observed data is normally distributed around the mean, assuming the mean and variance estimates are correct.

This means that a QQ plot should reveal problems if the simulator is not normal, *or* if the mean and/or variance estimates are flawed. Similarly, because the mean (and variance) diagnostics rely on the normality assumption to some degree, these can wrongly reveal problems when instead the normality assumption is broken. This would be acceptable if the only purpose of diagnostics were to raise issues when they exist, and to allow further study, but an important purpose of diagnostics is to also *diagnose* what the problems are. As such, it would be helpful to have some diagnostics which assess the normality assumption in isolation, rather than assessing the conditional normality assumption.

Following a similar strategy as those used to assess the mean and variance, we make use of replicated simulations to check the normality assumption. If it is affordable, a single heavily replicated simulation could allow for visual assessment of the normal assumption via a histogram. However, this would be an inefficient use of

simulation time, as this set of runs would not provide any information about the mean or variance across the input space, nor would it reveal issues if the normality assumption is only broken in a different region of input space. As such, we would also like normality diagnostics which can use the same data as the other diagnostics, and which can assess the normality assumption throughout the input space.

Because normally distributed variables are characterised entirely by their first two moments (the mean and the variance), the later moments are the same for all normally distributed quantities. These are what we will use to check the normality assumption.

The third moment is the skewness, and represents the asymmetry of the distribution. A distribution with a positive skew will have a longer upper tail, and a distribution with a negative skew will have a longer lower tail. An important attribute of the normal distribution is that it is symmetric, and so for the normal distribution, the skewness should be 0. In much the same way as before, we cannot obtain values of the simulator skewness directly, only sample skewness from replicated simulations, and we can compare these to reference distributions to see how unexpected the sample values are. The (standardised) sample skewness can be given by:

$$\frac{1/n \sum (y_j - \bar{y})^3}{\hat{S}^3}, \tag{2.9}$$

where $\bar{y}$ is the sample mean and S is the sample standard deviation (Stuart and Ord, 1994). Taking the *standardised* sample skewness removes the impact of the variance. To obtain a reference distribution for this, we can generate data points using our emulator, obtain a generated value of the sample skewness, and repeat to obtain an empirical reference distribution for the sample skewness. We can then use this reference distribution and the observed sample skewness to calculate the unexpectedness using Equation (2.7). Large values suggest the emulator needed to have assumed a distribution with a larger skew, and small values the opposite (in other words, large unexpectedness values suggest the simulator has negative skew, and small values suggest the simulator has positive skew).

We can repeat this exact procedure, but also for the fourth moment: the kurtosis. The kurtosis is a measure of how large the tails are. Kurtosis values are often compared to the normal distribution, which has a kurtosis of 3 (3 is often subtracted from the kurtosis, making the normal distribution the base case with an excess kurtosis of 0). Distributions with a kurtosis less than the normal distribution will

have smaller tails and a lesser propensity to extreme values. Distributions with a kurtosis greater than the normal distribution will have larger tails and a greater propensity to extreme values. The (standardised) sample (excess) kurtosis can be given by:

$$\frac{1/n \sum (y_j - \bar{y})^4}{\hat{S}^4} - 3, \tag{2.10}$$

where $\bar{y}$ is the sample mean and S is the sample standard deviation (Stuart and Ord, 1994). And so the kurtosis unexpectedness values can be calculated along with the skewness unexpectedness values. Large kurtosis unexpectedness values suggest the emulator needed to have assumed a distribution with larger tails, and small values the opposite (in other words, large unexpectedness values suggest the simulator has a negative excess kurtosis, and small values suggest the simulator has a positive excess kurtosis). Together, these higher moments provide information about the simulator's distribution, and should allow for key non-normality structures to be noticed (such as heavy skewness or bimodality). Even higher moments could be used as well, but the fifth moment (and above) is less interpretable, less easily understood, and requires more data than the skewness and kurtosis.

As examples, consider the toy simulator from Equation (2.1), which we know to be normal. Using the "good" emulator (although the "bad" emulator results will be identical), and the uniformly replicated validation data set from Section 2.2, we obtain the skewness unexpectedness values and the kurtosis unexpectedness values in Figure 2.11.

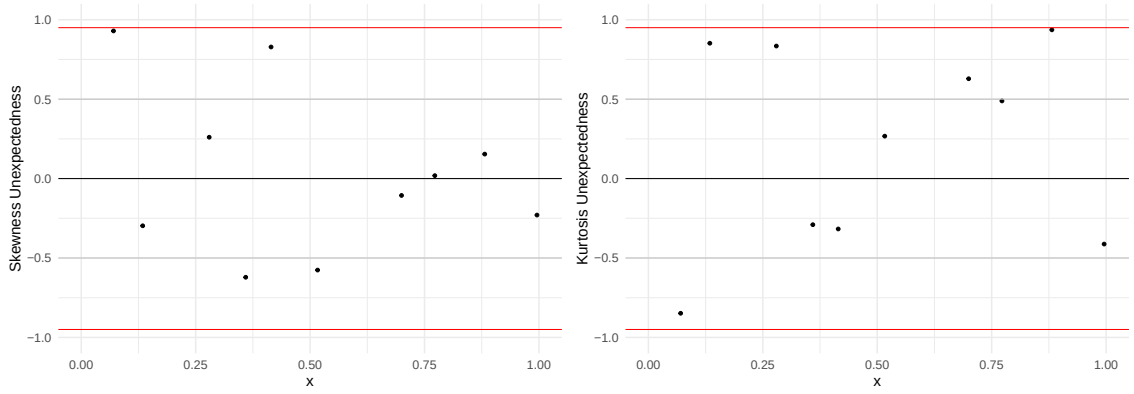


Figure 2.11: Skewness unexpectedness and kurtosis unexpectedness for the "good" emulator in Section 2.1.

These diagnostics (correctly) don't reveal any issues with this emulator. Both the skewness and the kurtosis unexpectedness look approximately uniform between

-1 and 1, with no values larger than 0.95 or smaller than -0.95, and are (reasonably) evenly allocated above and below 0.

As a different example, consider the same toy simulator, but where the intrinsic noise is instead gamma distributed (given again in Equation (2.11)).

$$\begin{aligned} y(x) &= \sin(16x) + \cos(24x) + 8x + \epsilon; \\ \epsilon &\sim \text{Gamma}\big(1, 1\big). \end{aligned} \tag{2.11}$$

Following the same procedure (training a heteroscedastic GP using 20 unique locations chosen via the same maximin Latin hypercube, each run 20 times, and validating with 10 validation coordinates each run 5 times) should lead to diagnostics which flag issues, and they do:

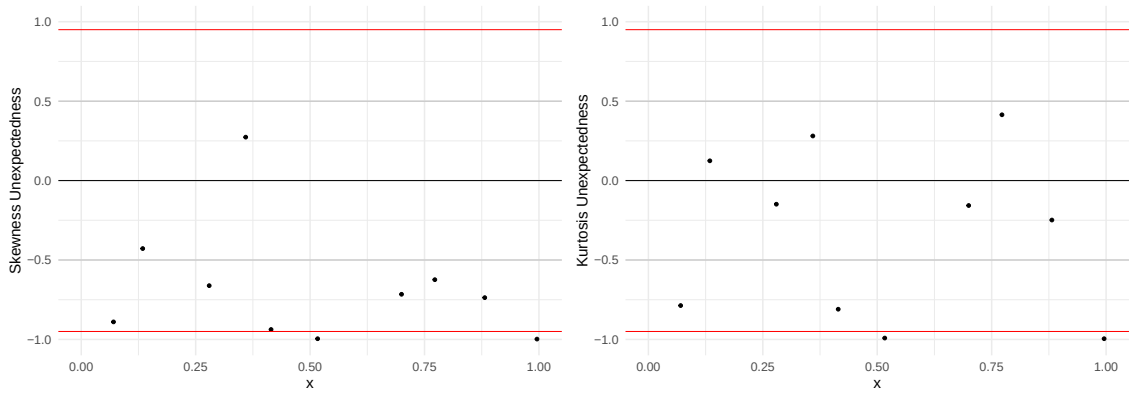


Figure 2.12: Skewness unexpectedness and kurtosis unexpectedness for the emulator of the gamma distributed toy simulator in Equation (2.11).

The skewness unexpectedness plot in Figure 2.12 reveals the skewness of the emulator to be too small (and so the skewness of the simulator is larger than the normal distribution). 9 out of 10 of the skewness unexpectedness values are less than 0, and 2 are less than -0.995. The kurtosis also reveals issues, with two unexpectedness values less than -0.95, suggesting a similar problem where the kurtosis of the simulator is larger than the normal distribution. Both of these statements are correct, and so we are then warned that the normality assumption may be insufficient.

This is reassuring, and Figure 2.12 now also acts as a guide for how gamma distributed noise can appear with the provided diagnostics. Less skew noise (whether gamma or otherwise) may be less obvious to spot (at least without more validation data), but serious differences from the normal case are indeed independently diagnosable now.

As another example, consider the modified toy simulator in Equation (2.12):

$$\begin{aligned} y(x) &= \sin(16x) + \cos(24x) + 8x + \epsilon; \\ \epsilon &\sim N\big(round(\tau) * 3, (0.1 + 0.9x)^2\big); \\ \tau &\sim \text{Uniform}\big(0, 1\big). \end{aligned} \tag{2.12}$$

For low values of x, there is a large degree of bimodality - there are two distinct, normally distributed peaks. As x gets larger, the bimodality becomes less pronounced, as the noise of each peak grows and they begin to merge together. This is a slightly more complex non-normality, as the non-normality changes through the input space. Nonetheless, the normality diagnostics are able to identify problems here.

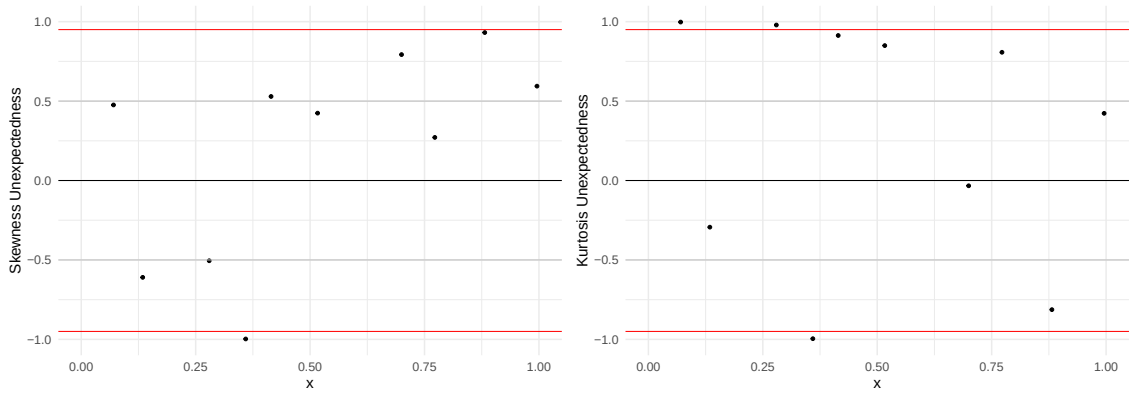


Figure 2.13: Skewness unexpectedness and kurtosis unexpectedness for the emulator of the bimodal distributed toy simulator in Equation (2.12).

Figure 2.13 reveals this bimodality in a slightly less understandable way, but it does still reveal normality issues. The skewness appears mostly acceptable, except for one point where the skewness is less than -0.95 (1 out of 10 is not unbelievable however). This low value is because, of the validation runs for this point, 4 are for the lower peak, and one for the upper peak; this gives the impression of a large degree of skew, and thus the sample skewness is unexpectedly quite high here. The kurtosis provides more obvious evidence for non-normality, and raises the issue, but it is less intuitive alone as to why an issue is raised.

2 of the kurtosis unexpectedness values are larger than 0.95, and one less than -0.95, which is more than one would expect if the simulator was indeed normal. The two that are larger than 0.95 are the natural result - the bimodal distribution for low x does indeed have a low kurtosis (and thus the unexpectedness values should be large). The one less than -0.95 is due to the aforementioned perceived skew for this point - with 4 points tightly clustered together, and one outlier, the kurtosis is quite

large, by chance (and so the unexpectedness is quite small). By coincidence this kurtosis unexpectedness is small, but it is still caused by the underlying bimodality, even if not in a way one would expect.

A second feature one could identify from the kurtosis diagnostic, is that there are no problematic unexpectedness values for large x. This is a true feature, as mentioned before, as the bimodality is less pronounced for large values of x. Structures like this however are difficult to truly spot with a low number of unique validation coordinates, and are also difficult to trust without an abundance of unique validation coordinates.

One thing that is noticeable from this example, is that whilst the normality diagnostics can reveal problems with the normality assumption, the specifics of these problems are not always identified. With bimodality, the kurtosis plots can reveal problems, but a large (or small) kurtosis need not always be caused by bimodality (a t distribution also has a different kurtosis than the normal distribution). From this point, if one wishes to know how and why a simulator is not normally distributed, a single heavily replicated run may be the only option. Because the distribution could change throughout the input space, and therefore only exhibit non-normality in a specific region of input space, the heavily replicated point can be chosen as the same input point as one of the previously obtained problematic skewness or kurtosis unexpectedness values. Nonetheless, a single heavily replicated point is not an efficient use of a simulation budget, and so should only be used as a last resort, or when the simulation budget is sufficiently large. Further distinguishing between the different ways in which a stochastic simulator can be non-normal in an efficient way remains an open question.

Overall, whilst these normality diagnostics are reasonably data efficient, they are more data hungry than the mean and variance diagnostics, as higher order moments are more difficult to precisely estimate. We need at least 3 runs per point to calculate the sample skewness, and at least 4 for the sample kurtosis. As such, if both of these diagnostics are desired, then 3 replicates are needed.

For these examples, obtaining the reference distributions was done via sampling from the emulator. Technically, since the sample skewness values and sample kurtosis values do not depend on the mean or variance of the normal distribution, any normal distribution could have been used to obtain the reference distribution (the $N(0, 1)$ distribution, for example). This would have been computationally cheaper to obtain, and should not produce differing results.

Normality Diagnostics Tolerance

An issue with assessing the normality assumption, is that whilst many real life processes appear roughly normal, they are rarely exactly normal. Emulators, and statistical models more generally, typically have no uncertainty about the distribution: the data is assumed to be normal (or whichever distribution is assumed) with 100% certainty. As such, as the number of replicates increases and the aleatoric uncertainty from the *sample* skewness and the *sample* kurtosis decreases, the diagnostics will ultimately always reveal the normal distribution is invalid. This is technically correct; the normal assumption is usually an approximation and thus is never completely valid, but it can often be practical. This is a similar problem as with the variance diagnostics, where many implementations will take a point estimate for the variance, and thus should also always technically be rejected. We solve the normality problem in a similar way; by adding a tolerance to error to the normality assumption.

Adding a tolerance to error to the normality assumption is difficult, as there is no general, parametrised, distribution of which the Gaussian is a subset. However, there do exist extensions to the normal distribution which allow for skewness other than 0, and (excess) kurtosis other than 0. We can then add a tolerance to error using these extended distributions, and proceed as before.

For the skewness diagnostics, we can make use of the skew normal distribution (O'Hagan and Leonard, 1976):

$$f(x) = \frac{2}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \int_{-\infty}^{\alpha\left(\frac{x-\mu}{\sigma}\right)} \frac{1}{\sqrt{2\pi}} e^{-\frac{t^2}{2}} dt.$$

This is the normal distribution with an additional parameter α, which affects the skew. A skew normal distribution with an α value of 0 exactly equals the normal distribution. The skew of the skew normal equals:

$$\frac{4-\pi}{2} \frac{(\delta\sqrt{2/\pi})^3}{(1-2\delta^2/\pi)^{3/2}}; \tag{2.13}$$

where

$$\delta = \frac{\alpha}{\sqrt{1+\alpha^2}}.$$

We can then specify our tolerance to skewness error as $U(-0.5, 0.5)$, say, draw samples from this distribution, numerically invert them using the above equation, and then use the resulting α values to draw samples from the skew normal distribution (along with the emulator predictions for the mean and variance) to use in our calculation of

the reference distribution for the sample skewness. This is then the equivalent of saying that we are ambivalent to true skewness values that are between -0.5 and 0.5, and so this is our tolerance to skewness error in the underlying intrinsic simulator variability.[2]

We can similarly do the same for the kurtosis unexpectedness, using the generalised normal distribution (Nadarajah, 2005):

$$f(x) = \frac{\beta}{2\sigma\Gamma(1/\beta)} e^{-\left(\frac{|x-\mu|}{\sigma}\right)^{\beta}},$$

where Γ represents the gamma function. The generalised normal distribution is similar to the skew normal distribution, in that it contains an additional parameter β, which controls the kurtosis. A generalised normal distribution with a β value of 2 exactly equals the normal distribution. The (excess) kurtosis of the generalised normal equals:

$$\frac{\Gamma(5/\beta)\Gamma(1/\beta)}{\Gamma(3/\beta)^2} - 3. \tag{2.14}$$

We can then follow the same procedure as for the skewness unexpectedness: specify our tolerance to kurtosis error (e.g. $U(-0.5, 0.5)$), draw samples from this distribution, numerically invert them using the above equation, and then use the resulting β values to draw samples from the generalised normal distribution. This sampling procedure can then be used to obtain a reference distribution for the sample kurtosis, and thus can be used to obtain a kurtosis unexpectedness value.

To show the utility of this tolerance to error, consider the toy simulator in Equation (2.15):

$$\begin{aligned} y(x) &= \sin(16x) + \cos(24x) + 8x + \epsilon; \\ \epsilon &\sim \text{Gamma}\big(50, 10\big). \end{aligned} \tag{2.15}$$

This is the same toy simulator as before in Equation (2.1) but now with a gamma distributed noise term. However, this noise term is also reasonably normal, as we can see from Figure 2.14, which shows 1000 runs for a single location.

We can see here, that although this simulator is not normally distributed, and does exhibit some degree of skew (in fact, the true skewness is 0.25, and the true excess kurtosis is 0.09375), it is at least visually, quite normally distributed. So,

[2] $U(-0.5, 0.5)$ is subjective and depends on the specific application. In this case the values were chosen as values where the resulting theoretical distributions still (subjectively) appeared normal in histograms.

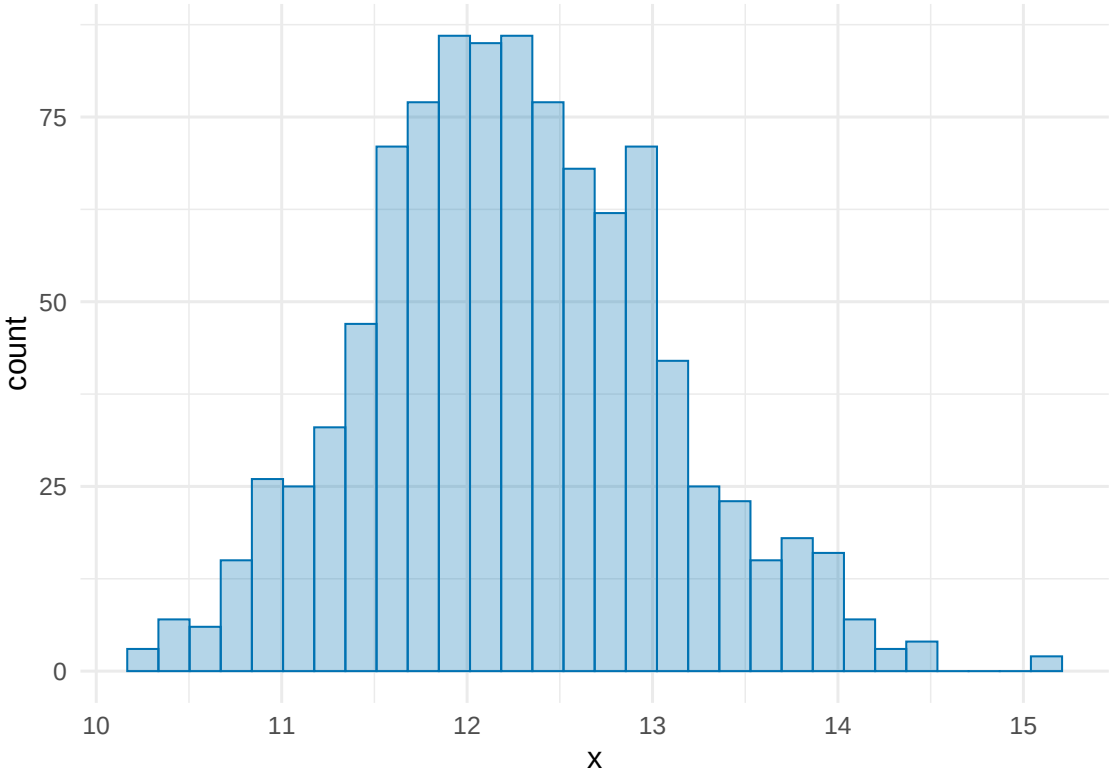
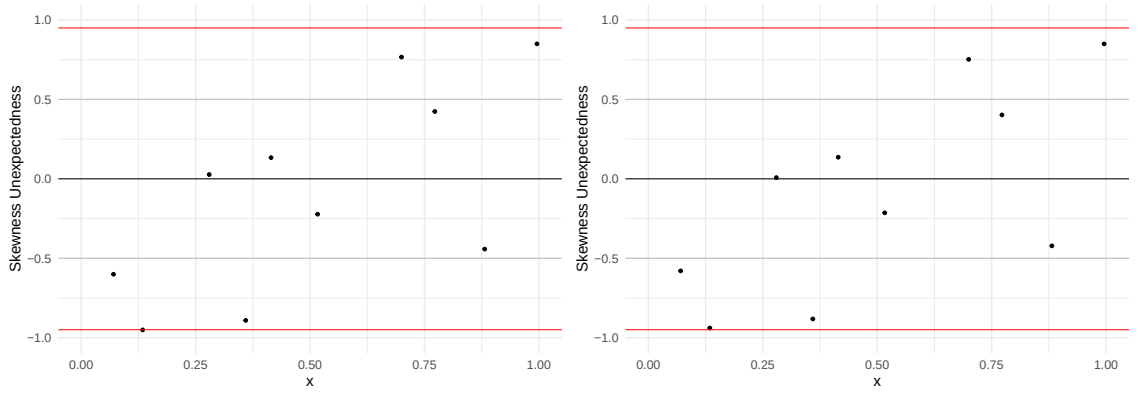


Figure 2.14: Histogram of $Gamma(64, 10)$ toy simulations from Equation (2.15).

depending on our tolerance to error, we may or may not wish to allow for a normal assumption to be used. Assuming that we do believe that a normal assumption is acceptable, we would not want diagnostics to reveal major issues.

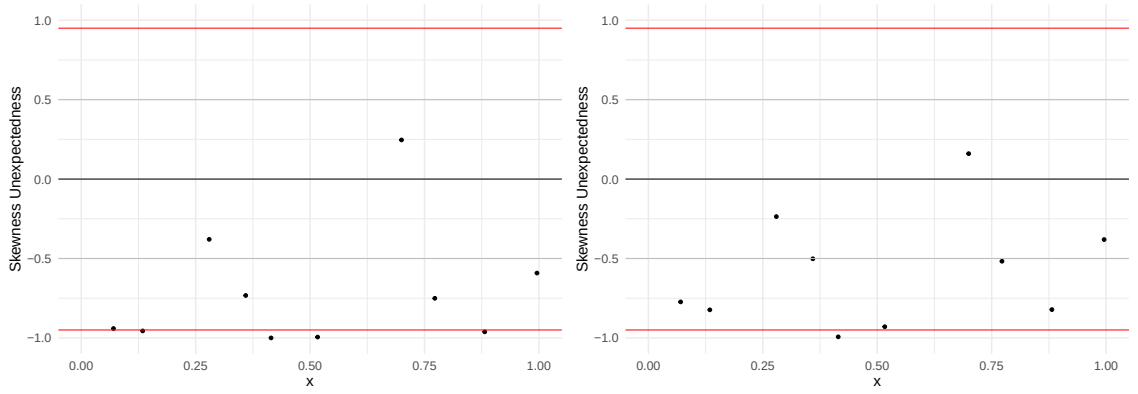
Fitting an emulator to 20 unique locations, each replicated 20 times, we can then perform the normality diagnostics. Figure 2.15 presents the skewness diagnostics with and without tolerance to error, for two sets of validation data; one set of 10 unique locations each run 5 times, and another replicated 100 times. Figure 2.16 presents these same diagnostics but for the kurtosis.

These plots (and this modification more generally) mirror those for the variance in Figure 2.10. We can see that, for only 5 replicates, including tolerance to error makes essentially no difference. However, when there are many (100) replicates, not including tolerance to error reveals very striking issues (4 skewness unexpectedness values are less than -0.95, and 2 kurtosis unexpectedness values are less than -0.95), even though this simulator is reasonably normal. Including tolerance to error mitigates this problem, and only 1 skewness unexpectedness value is less than -0.95, and one kurtosis unexpectedness value. This suggests that including tolerance to error can be a useful idea.



(a) 5 replicates, no tolerance to error

(b) 5 replicates, tolerance to error

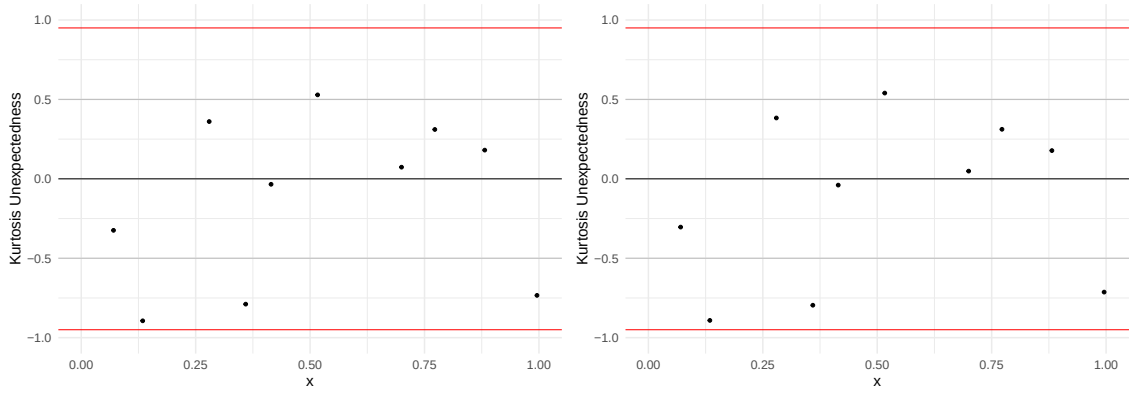


(c) 100 replicates, no tolerance to error

(d) 5 replicates, tolerance to error

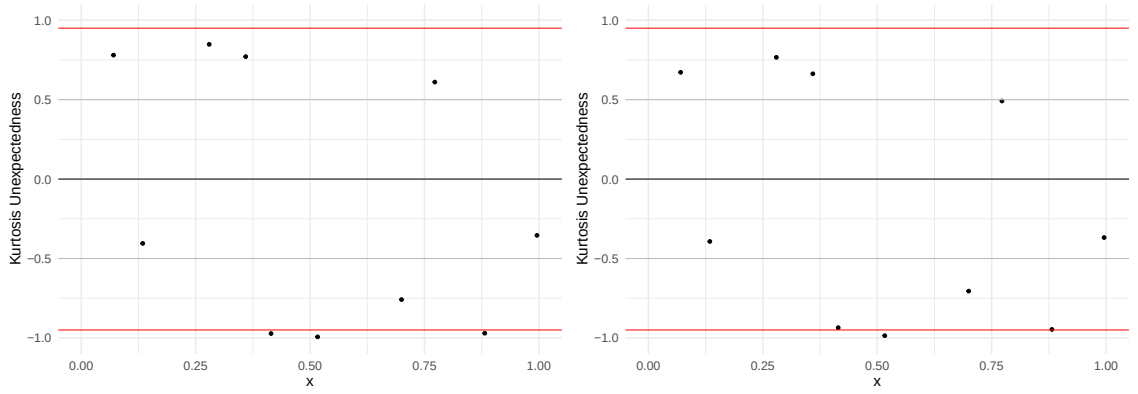
Figure 2.15: Sample skewness unexpectedness against x for the $Gamma(50, 10)$ emulator. Top left is for 5 replicates per point and without any tolerance to error; top right is for 5 replicates per point and with tolerance to error; bottom left is for 100 replicates per point and without any tolerance to error; bottom right is for 100 replicates per point and with tolerance to error.

Importantly, however, tolerance to error does not affect the proportion of points that have positive or negative unexpectedness. So in these examples, even with tolerance to error, we can still see that, on average, the sample skewness is positive (9 out of 10 of the skewness unexpectedness value are less than 0). This is an important point, and will be common with the variance diagnostics and even the mean diagnostics. For example, an exceptional emulator, where the sample mean is essentially perfect but *always* slightly too large, will be revealed in the diagnostics as having a generally overly large mean (but shouldn't have many unexpectedness values larger than 0.95 (or smaller than -0.95) This makes extreme values, and the proportion of extreme values, more useful than the proportion of positive or negative values. In this way, tolerance to error (or epistemic uncertainty when available)



(a) 5 replicates, no tolerance to error

(b) 5 replicates, tolerance to error



(c) 100 replicates, no tolerance to error

(d) 100 replicates, tolerance to error

Figure 2.16: Sample kurtosis unexpectedness against x for the $Gamma(50, 10)$ emulator. Top left is for 5 replicates per point and without any tolerance to error; top right is for 5 replicates per point and with tolerance to error; bottom left is for 100 replicates per point and without any tolerance to error; bottom right is for 100 replicates per point and with tolerance to error.

provides a degree of robustness to spurious extreme unexpectedness results.

Interpreting Normality Diagnostics

We discussed previously how the developed normality diagnostics can provide an analogue to the mean and variance diagnostics but for the normality assumption - providing information about *how* the normality assumption might be broken, and information about *where* the assumption is broken. Their limited capability regarding the latter attribute (where the normality assumption is broken) was already discussed, as without a large validation budget it can be difficult to determine whether a perceived local misspecification is real or an artefact of the random nature of the diagnostics. Here we will also discuss their limited capability regarding the former

(how the normality assumption is broken).

Whilst these diagnostics aim to check whether the skewness and the kurtosis are correct, the way they do so uses the normality assumption. Because the reference distribution for the skewness (and the kurtosis) is constructed by assuming the underlying distribution is normal, it is possible to obtain unexpected skewness (or kurtosis) values for a distribution which is not skewed (or has an excess kurtosis of 0) if it is sufficiently non-normal in other ways. We say this previously with the bimodal example, as there the simulator was symmetric, but the low amount of validation data and the bimodality tricked the skewness diagnostic into believing it was also skewed at one location.

For a more extreme example, consider the simulator in Equation (2.16)

$$\begin{aligned} y(x) &= \sin(16x) + \cos(24x) + 8x + \epsilon; \\ \epsilon &\sim \text{Beta}\big(0.1, 0.1\big). \end{aligned} \tag{2.16}$$

This is a very non-normal simulator, which we can see from Figure 2.17, which shows 1000 runs for a single location.

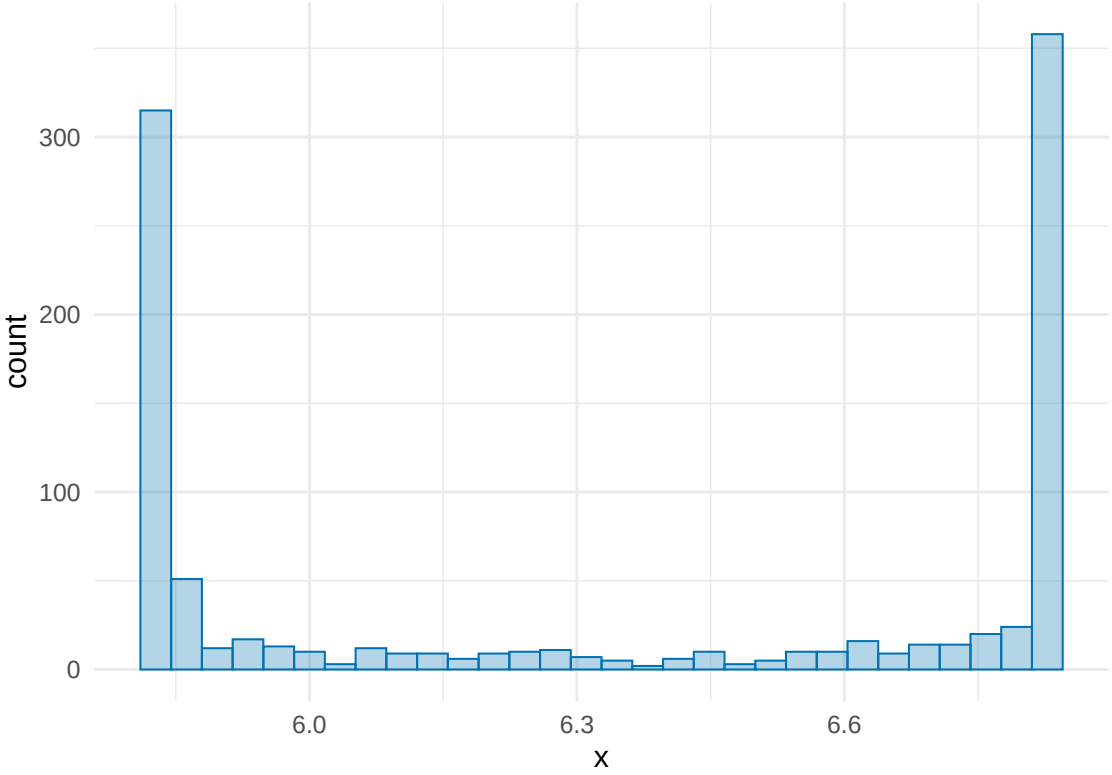


Figure 2.17: Histogram of $Beta(0.1, 0.1)$ toy simulations from Equation (2.16).

Here the distribution is symmetric, but it is incredibly non-normal, bounded above and below, with most of the distribution at each of these bounds.

The skewness diagnostics should not flag a problem (and the kurtosis diagnostics should), but this is not what happens here. Figure 2.18 shows the normality diagnostics for this example, after fitting an emulator to 20 unique locations each replicated 20 times, with one set of 10 unique validation locations each run 5 times.

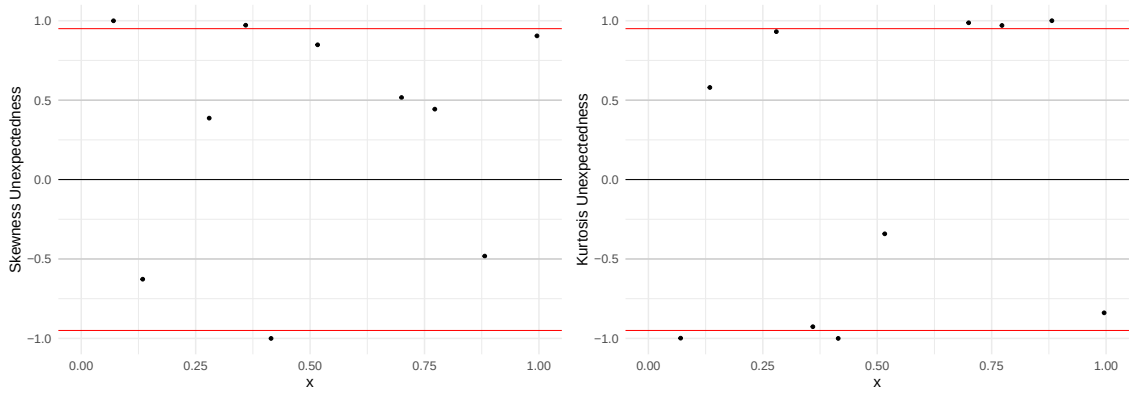


Figure 2.18: Skewness unexpectedness and kurtosis unexpectedness for the emulator of the Beta distributed toy simulator in Equation (2.16).

Here, the skewness diagnostics reports three unexpectedness values with absolute value larger than 0.95, and one larger than 0.999. This suggests that the simulator is skewed, which it is not. This is because, like with the bimodal simulator, multiple validation locations have, by chance, a majority of points in one of the peaks. This makes it seem as if the simulator is skewed, even if it is not. This is ultimately a flaw with the normality diagnostics, as while they aim to reveal specific information about the non-normality of the simulator, they can be report the incorrect type of non-normality (because they themselves also rely on the normality assumption).

For this specific example, increasing the number of validation replicates (up to 50) rectifies this issue, with no skewness unexpectedness with absolute value larger than 0.95 (and all 10 kurtosis unexpectedness values with absolute value larger than 0.95). But this degree of validation replication is unlikely to be practical, and may not always help with other types of non-normality.

The developed normality diagnostics are still able to flag issues when issues are present, but their ability to specifically diagnose the type of non-normality can be weak in some circumstances. Rather than assessing whether the skewness is 0, the skewness diagnostics instead check whether the sample skewness could be a sample skewness from normally distributed samples (which has a skewness of 0). A similar difference exists for the kurtosis diagnostic. The two statements are similar, but they

are not equivalent, which can lead to surprising results in extreme cases.

2.3 Building Example

We now return to EnergyPlus, using it as an example to apply these diagnostics to.

The specific building we use is a reference hospital design provided by the U.S. Department of Energy (Field et al., 2010). We use an 'ideal loads' heating and cooling system, where the building will always be perfectly heated (or cooled) to the correct temperature with a subsequent increase in the building's energy usage. For the weather generator we use a very simplistic mechanism: we randomly stitch together weeks of historical observed weather, stratified according to season. The specific historical record we use is for Plymouth, using data from 1961 to 2016, and interpolating missing data. We then also consider as inputs to be adjusted: the thickness of the wall concrete (varying from 0.05m to 0.75m), the thickness of wall insulation (varying from 0m to 0.4m), the roof insulation thickness (varying from 0m to 0.5m), the thickness of the ground concrete (0.05m to 0.75m) and the percentage size of the windows compared to the walls (25% to 75%). This is a simplified building model, and the choices made are not entirely realistic, but it can serve sufficiently well as an example. A more realistic example is used in Chapter 4.

The shape of this building is provided in Figure 2.19.

To begin with, we perform 250 simulations; each with a unique input setting, with inputs chosen via a maximin Latin hypercube (McKay et al., 2000). Our output of interest is the average energy usage of this building, which we model with a heteroscedastic GP using the `hetGP` package, with a zero mean function and a squared exponential covariance function. To then check the performance of this emulator, we apply the stochastic diagnostics outlined previously. Validation points are chosen using another maximin Latin hypercube of size 50, with each point replicated 3 times.

Figure 2.20 shows the sample mean unexpectedness for this building and 2.21 shows the variance unexpectedness.

The mean diagnostics have only one unexpectedness value with absolute value larger than 0.95. With 50 total points, this is reasonable, suggesting that the mean is captured adequately. For the variance however, 8 are less than -0.95, 5 of which are less than -0.995. This suggests that, despite the reasonable mean predictions,

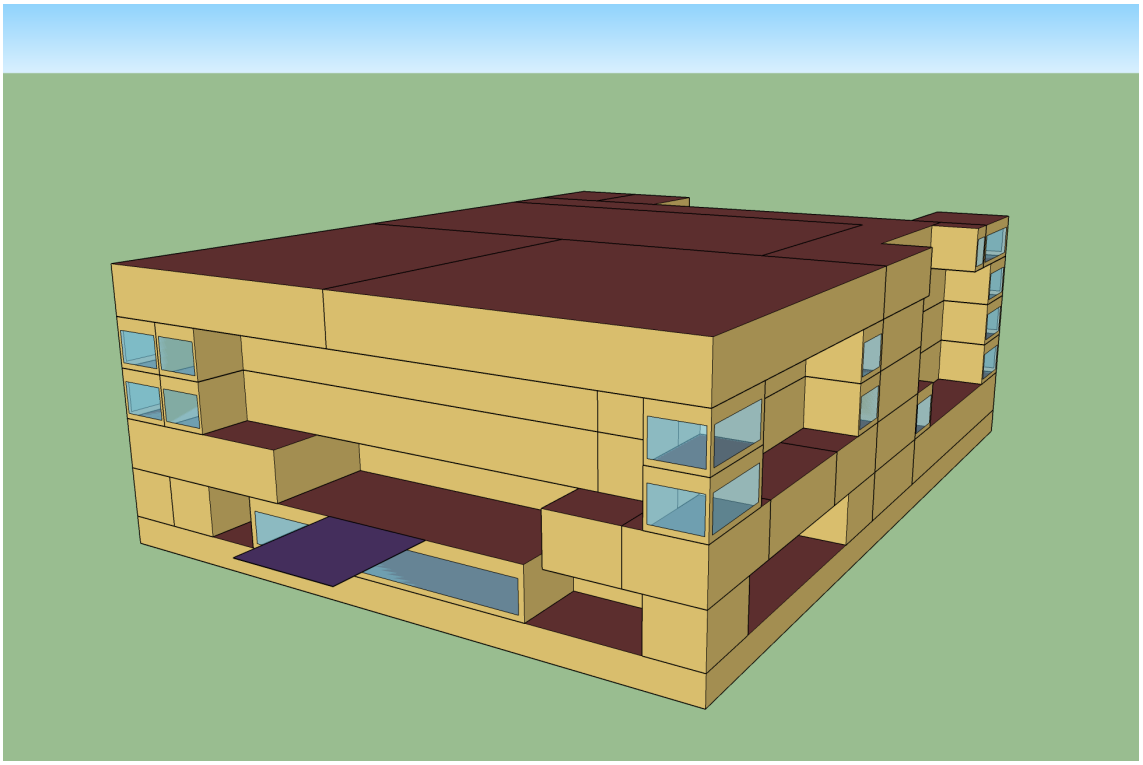


Figure 2.19: The geometry of the modelled reference hospital building.

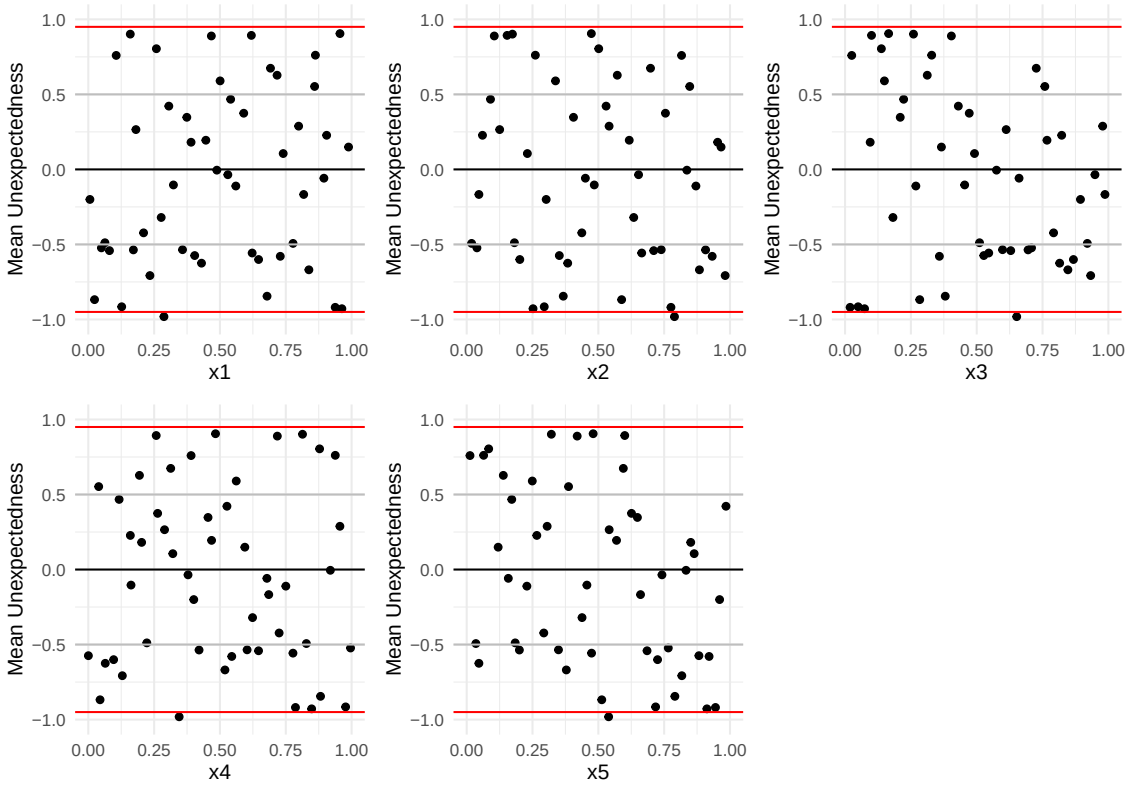


Figure 2.20: Sample mean unexpectedness for the emulator of the building simulator using 250 training points.

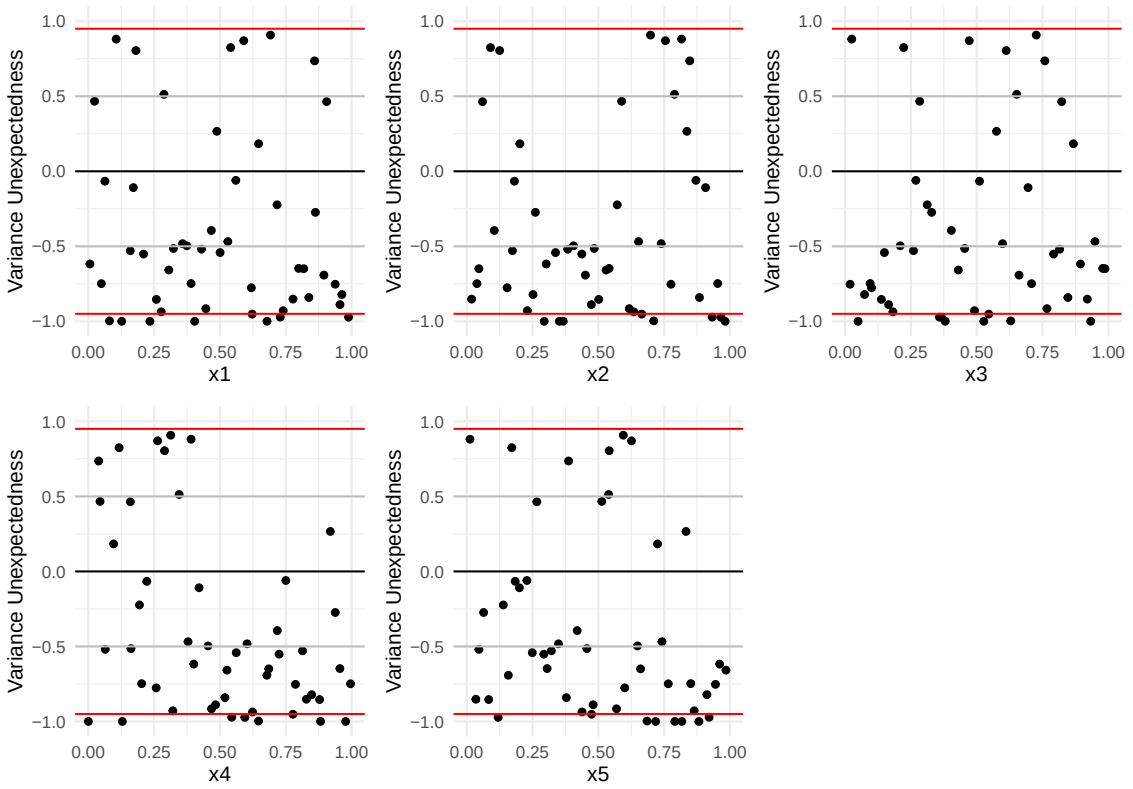


Figure 2.21: Sample variance unexpectedness for the emulator of the building simulator using 250 training points.

the variance is underestimated. The plots don't seem to reveal any obvious patterns to this underestimation, suggesting it is a global issue.

Whether these flaws are a problem in practice will depend on the specific problem and objectives at hand. For example, the expectation of the stochastic ocean simulator in Baker et al. (2020b) corresponds with the underlying truth, and so in that case a good mean prediction but a poor variance prediction would be acceptable. Similarly, in Chapter 4, the mean is the quantity of interest and so an underestimated variance would not be a problem. For this example however, we assume this flaw *is* a problem.

To fix this emulator, a straightforward option would be to simulate further. Additional simulations should improve the emulator, and could provide improved variance predictions. This may not be realised, however, if the underlying simulator variation is not normal. Non-normality could be the reason that the emulator is flawed, and it can also invalidate the diagnostics (potentially causing spurious results). We can use the normality diagnostics to check this, and therefore check whether the emulator variance flaw is real and whether the issue could be fixed with additional

simulation. Figure 2.22 and 2.23 show the skewness and kurtosis unexpectedness.

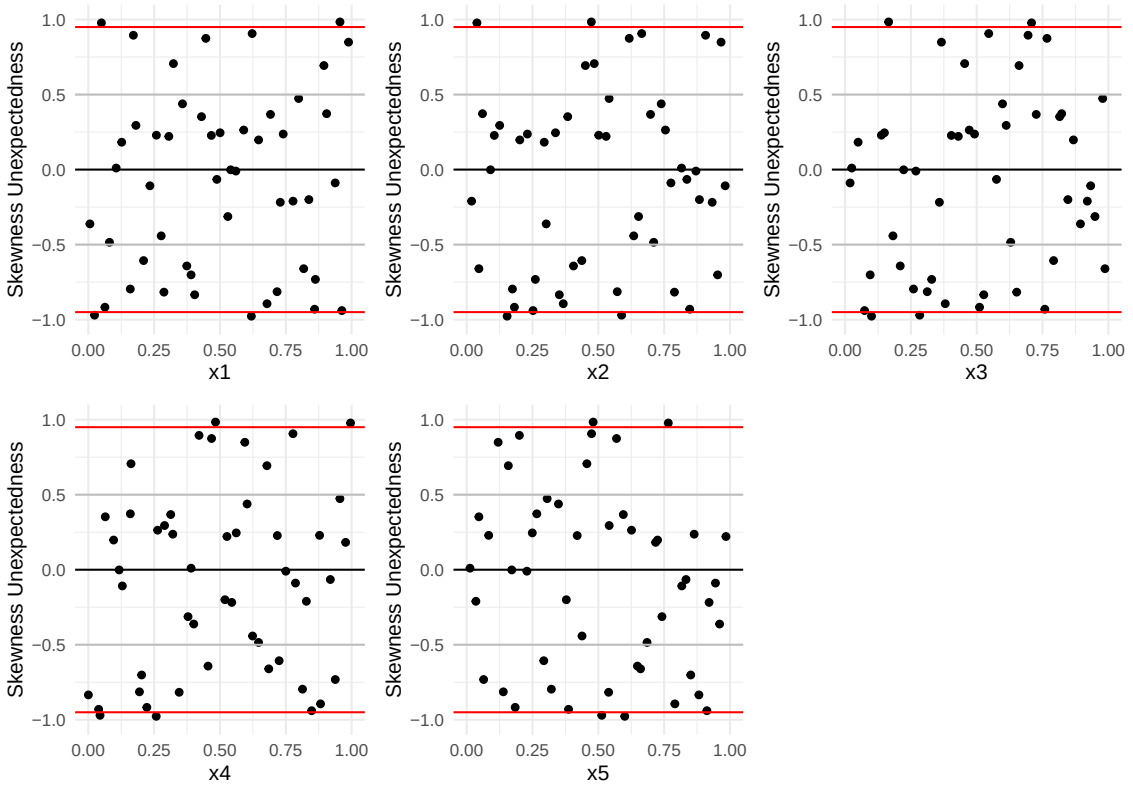


Figure 2.22: Sample skewness unexpectedness for the emulator of the building simulator using 250 training points.

We can see from these plots, that the skewness unexpectedness does not reveal any glaring issues, with 4 skewness unexpectedness values with absolute value larger than 0.95, and 2 kurtosis unexpectedness values with absolute value larger than 0.95. This suggests (but does not guarantee) that the simulator is still normally distributed, and the emulator simply did not capture the variance process correctly.

To improve this emulator, we combine the validation data with the training data, and simulate an additional 300 points to make up the training data for a new emulator. Because our goal is to specifically improve the variance, replication may be beneficial here. We have some prior belief that the variance process will be simpler than the mean process (requiring fewer space-filling runs), and replicated simulations provide some insight into the variance at those specific locations. The merged validation data is already replicated 3 times, and we use 50 of our additional simulations to make this 4 times. Our remaining 250 additional runs are chosen to be the same as the initial 250 runs used to train the initial emulator. Together, this should provide enough information to capture the underlying mean and variance

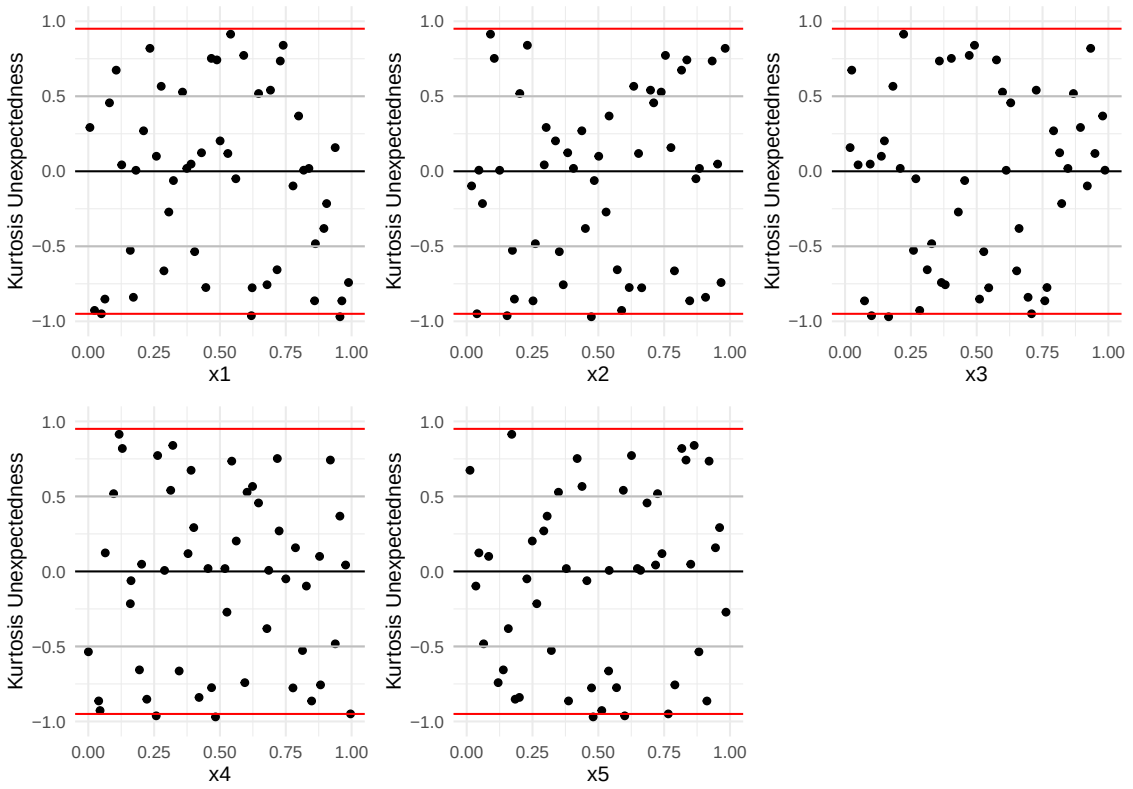


Figure 2.23: Sample kurtosis unexpectedness for the emulator of the building simulator using 250 training points.

processes accurately.

To then check this new emulator, we obtain another 200 validation points, again by a Latin hypercube of size 50, each replicated 3 times. Figures 2.24 and 2.25 show the same diagnostic plots as before.

These plots show a much improved emulator. Two mean unexpectedness values have absolute value greater than 0.95, and only one variance unexpectedness value has absolute value greater than 0.95, which is much more reasonable. There is however one mean unexpectedness value with a value of -0.9986, which is somewhat extreme, and would correspond to a standardised error of 3.19. It is not impossible for such a value to appear by chance, but it is less than ideal. Because the unexpected point is for a very small value of x_2 (0.000155), there could be a very local issue with the emulator's mean. In Chapter 3, we find that this parameter (x_2: wall insulation thickness) exhibits a steep change at low values, which may explain this local emulator flaw. Additionally, for this example none of the 750 training data points were for an x_2 value as low as 0.000155, and so it is not particularly surprising that the emulator may struggle here. We could improve this emulator (for example,

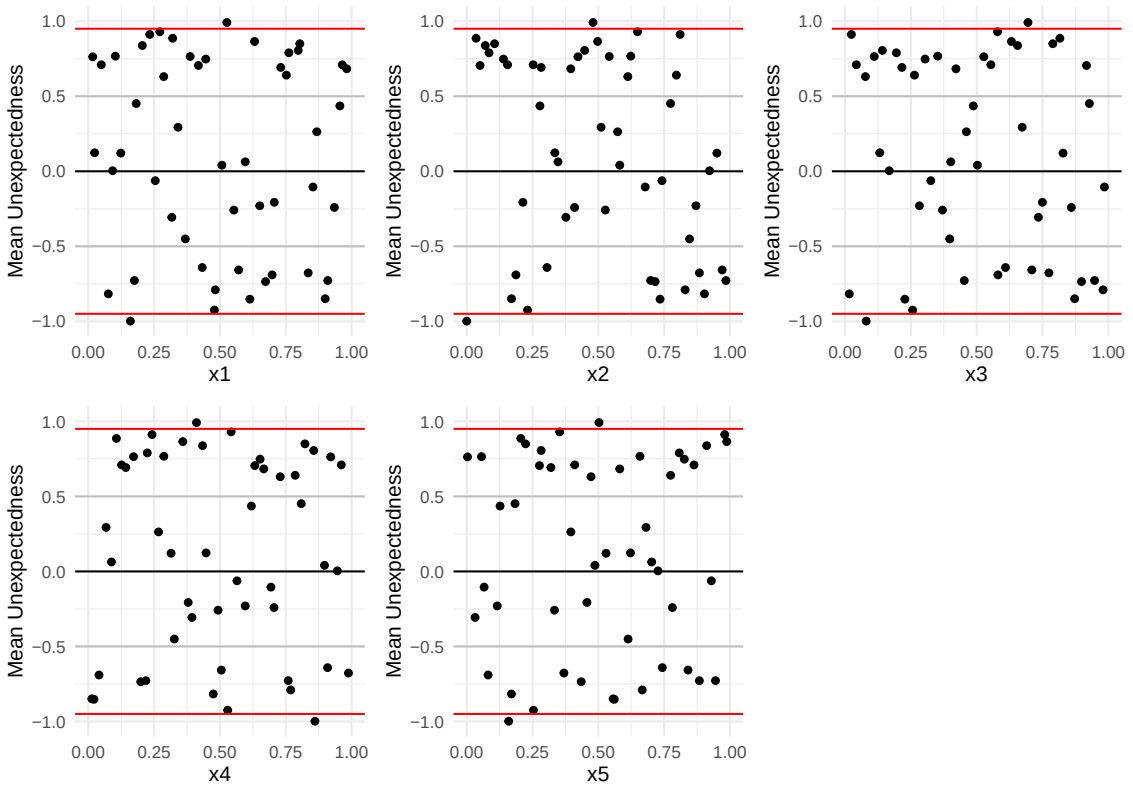


Figure 2.24: Sample mean unexpectedness for the emulator of the building simulator using the combined 750 training data points.

by simulating more for low values of x_2), however, we decide that this emulator is acceptable, and would simply recommend some caution when dealing with small values of x_2.

Normality diagnostics are not essential now, as we already concluded that the simulator is normally distributed. Regardless, one of the skewness unexpectedness values has absolute value larger than 0.95, and none of the kurtosis unexpectedness values, both of which are reasonable.

Overall, this example has shown how the developed diagnostics can aide in the construction of a capable statistical model, and provide insights into the strengths and weaknesses of a given emulator.

2.4 Validation Design

With the developed diagnostics, validation points are replicated to obtain sample statistics (such as the mean and variance). How many replicates are needed in the validation data remains an unresolved question. Tackling this problem in the contexts of *fitting* a stochastic emulator has already received research attention; for

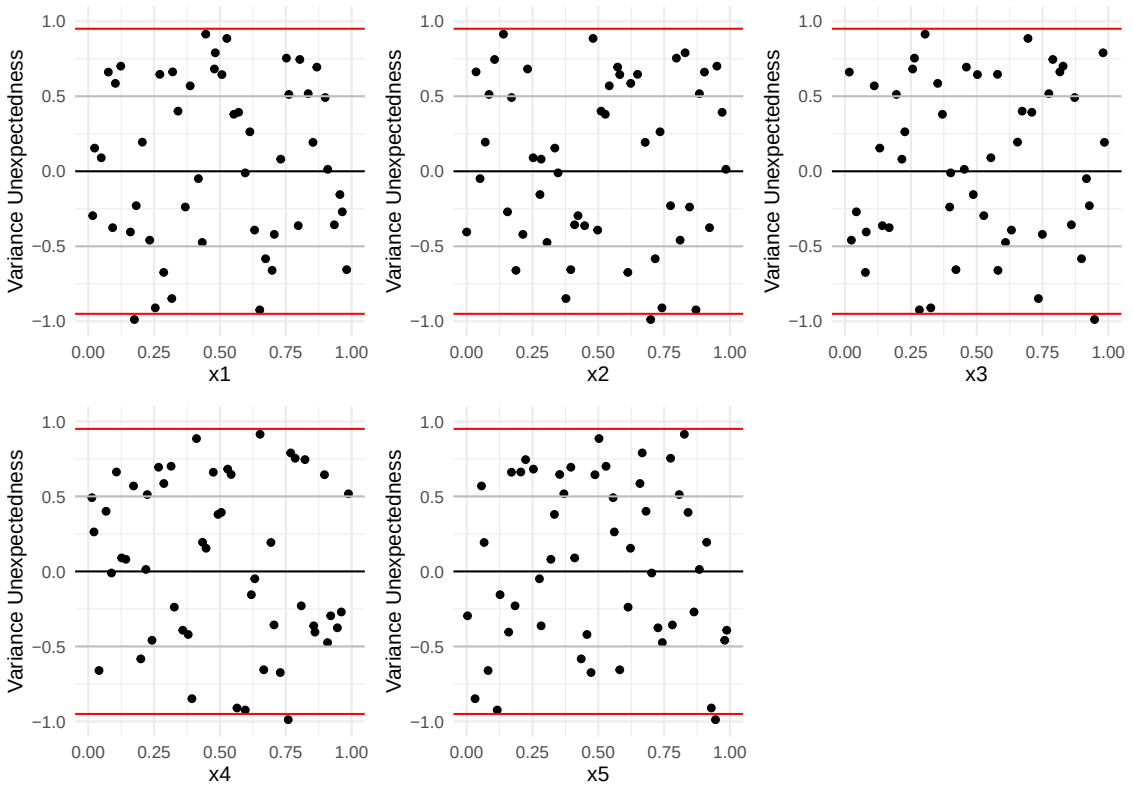


Figure 2.25: Sample variance unexpectedness for the emulator of the building simulator using the combined 750 training data points.

example, Binois et al. (2019) choose to replicate or 'explore' based on the integrated mean squared prediction error.

For validation, given that we want to check many different regions of the emulator, it is sensible to make sure that validation points are, to some degree, space-filling. Additionally, simulations could be replicated more in regions where the intrinsic variance is larger, as the resulting sample statistics are prone to greater error in these regions. There are many ways to do this, and what follows are two possible solutions.

2.4.1 Targeting the Mean

One mechanism for choosing validation points is as follows: Construct a Sobol sequence (Bratley and Fox, 1988), taking the first n_v points in the sequence as the unique validation points (where n_v is specified beforehand). The number of validation replicates for each is then chosen as the minimum number required to ensure the epistemic uncertainty around the mean will be smaller than the aleatoric sampling variation around the sample mean. That is, we select the number of validation runs r_i^* at a unique point $\mathbf{x}_i^*$ such that the inequality $\sigma^2(\mathbf{x}_i^*)/r_i^* < \sigma_\mu(\mathbf{x}_i^*)$ will hold, where

$\sigma_\mu(\mathbf{x}_i^*)$ is the noiseless uncertainty around the mean.

This criteria targets the mean, but improvements should also be felt for the sample variance as well. Overall, any design scheme which fills space and replicates more in regions of high intrinsic variability could have value. This criteria ensures that the epistemic uncertainty around the mean is not overwhelmed by the intrinsic noise; forcing a majority of the uncertainty to be the uncertainty that we are actually interested in validating.

If the outlined process above does not exhaust the validation budget, then additional space-filling points can be taken from the Sobol sequence and allocated the required minimum number of replicates. This can be repeated until the validation budget is eventually exhausted. If at any stage an additional point cannot be allocated its required number of replicates, the process is stopped. A practitioner may instead wish to use slightly more than the validation budget to allow the inclusion of this final point, depending on how flexible the validation budget is. Alternatively, the extra point could be included with only the remainder of the simulation budget.

On the other hand, if the initial process exhausts the simulation budget without being able to allocate the correct number of replicates, then we scale down the number of replicates at each point until the validation budget constraint is fulfilled. This results in the aleatoric uncertainty being larger than the epistemic uncertainty. The alternatives are to exceed the validation budget (which is not always possible), or reduce the minimum number of space-filling points (which can reduce the chances local emulator flaws are discovered).

This scheme would allow for only one run at a point, which would prevent the sample variance and normality diagnostics from being applied. Forcing the scheme to replicate at least 3 times per point will allow all the developed diagnostics to be applied. Forcing the scheme to replicate at least once will allow the sample mean and sample variance diagnostics to be applied.

Applying this validation design scheme to the 'bad' emulator (with a minimum of 5 unique validation points, and forcing at least one replicate) yields Table 2.1.

Here, the validation design required 48 validation runs. This is two less than before in Sections 2.2.1 and 2.2.2, but essentially the same amount. The general trend is that more central x require more replicates. Figure 2.26 plots the prediction intervals for the mean and the overall prediction intervals for the 'bad' emulator.

From this figure, it is clear the emulator believes the intrinsic variance is increasing

x	0.1250	0.1875	0.2500	0.3750	0.5000	0.6250	0.6875	0.7500	0.8750	0.9375
r^*	3	4	4	6	6	6	6	5	5	3

Table 2.1: A table showing the results of the validation design scheme for the 'bad' emulator.

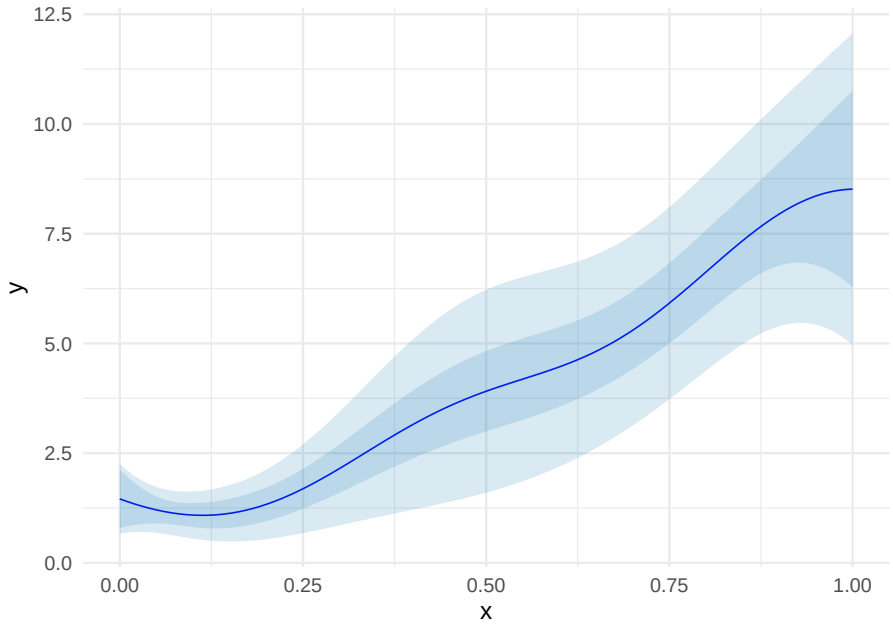


Figure 2.26: Two standard deviation intervals for the mean process and the overall process for the 'bad' emulator. Dark blue represents the mean process uncertainty and light blue represents the overall uncertainty.

as x increases. However, the epistemic uncertainty around the mean also increases (in general) as x increases. The difference between the two is thus largest when x is close to 0.5, hence the pattern seen in the validation design.

Note that for this example, requiring at least 3 replicates (so that both normality diagnostics could be used) would simply use the 2 remaining points in the validation budget to augment the existing design, no further changes would be needed. For this example, because we know the simulator is normal, not including such a requirement provides a more insightful example.

Figure 2.27 then plots the sample mean unexpectedness and the sample variance unexpectedness using this validation data set (for the 'bad' emulator).

These plots don't look too dissimilar to the previous stochastic diagnostic plots, providing the same conclusions. This is mostly because the number of replicates chosen by the validation design scheme do not differ wildly from the fixed five

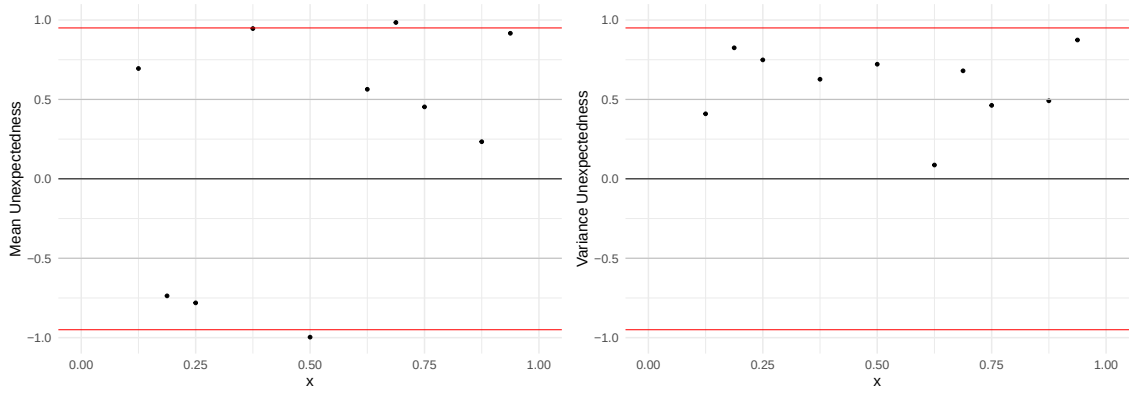


Figure 2.27: The sample mean unexpectedness plot (left) and the sample variance unexpectedness plot (right) for the 'bad' emulator using the validation design scheme.

used before. The advantages of this design scheme are that it reveals what the 'correct' number of replicates is (regardless of whether the 'correct' number can be obtained, and with 'correct' semi-arbitrarily defined), and that the use of a Sobol sequence allows easy augmenting of the initial unique validation point design. The disadvantages stem from its targeting of the mean - if the mean is of primary interest, this validation design scheme becomes less sensible. Additionally, using an emulator to such an extent to make judgements about how to independently assess the emulator seems tenuous.

This design targets the mean process because the mean is often of greater interest and thus scrutiny, but also because it is the only component with epistemic uncertainty (at least for the investigated emulator formulation). A similar validation scheme that targeted the sample variance would still be possible using a similar strategy, but would ultimately rely heavily on the subjective tolerance to error.

2.4.2 Simple Weighting

An alternative strategy is to simply weight the number of replications according to the predicted intrinsic variance. Both the sample mean and sample variance distributions depend on the intrinsic variance, and so this should improve both. Again, as with the previous design, we can also force the number of replicates to be at least 3 (as required for the kurtosis diagnostics), or 1 (as required by the variance diagnostics). When weighting the total number of replicates according to the intrinsic variance predictions, the number of desired replicates can be non-integers. This requires some degree of rounding, which can cause the target number of simulations to be missed.

We round-down the non-integers, and then re-add simulations according to which points were rounded-down the most.

Applying this scheme to the previous example, only requiring at least 2 runs per point, yields the validation design in Table 2.2.

x	0.07049055	0.13430022	0.27957167	0.35934015	0.41474998	0.51646108	0.69976717	0.77253228	0.88159390	0.99572530
r^*	2	2	3	5	5	6	6	6	7	8

Table 2.2: A table showing the results of the validation design scheme for the 'bad' emulator.

This design is then more straightforward than the previous - more replicates for larger x (which have larger intrinsic variance estimates).

Figure 2.27 then plots the sample mean unexpectedness and the sample variance unexpectedness using this validation data set (for the 'bad' emulator).

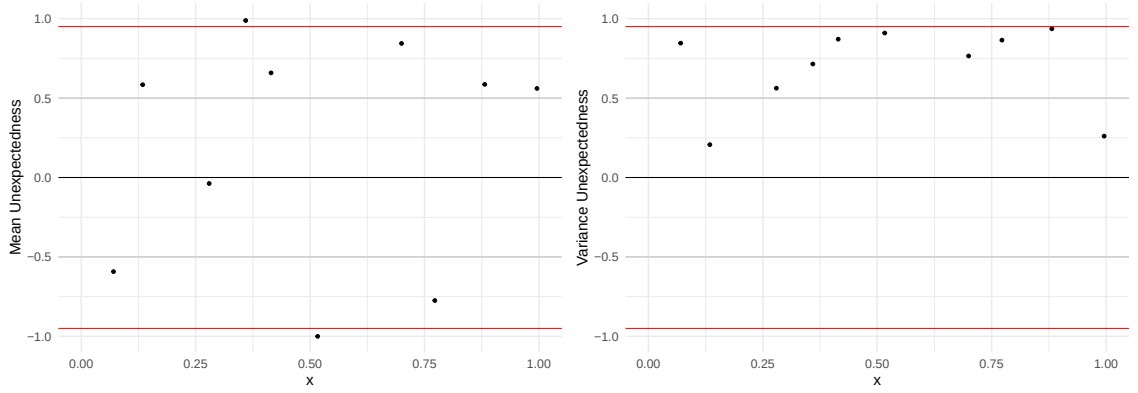


Figure 2.28: The sample mean unexpectedness plot (left) and the sample variance unexpectedness plot (right) for the 'bad' emulator using the variance weighted design scheme.

These results are not particularly different. One noticeable difference is that the variance diagnostics no longer show any extreme values (absolute value greater than 0.95), whereas the previous two validation data sets had one. This is likely due to sampling variability and not as a result of the validation design itself, as the validation design does check this region and replicates equally as much here as before. Additionally, one extreme value out of ten was not sufficient evidence for a serious problem, and therefore the conclusions do not change much regardless.

These designs do not substantially change the analysis for this toy example by much, but that does not mean they do not have the ability to impact other examples.

With examples where the validation budget is much larger, and examples where the intrinsic noise is much larger, these choices on where to replicate the validation runs is likely to have a greater impact. In general, the second design is perhaps preferable, because it is simpler and does not prioritise one of the diagnostics over the others. Additionally, without additional simulation in areas of high intrinsic variance, poorly estimated intrinsic variances in regions where the intrinsic variance is believed to be large would be more difficult to identify than in other regions. Targeted simulation where the intrinsic variance is believed to be large mitigates this issue.

In Chapter 4, where these diagnostics are again employed, neither validation design scheme is used. This is for two reasons, both of which can be relevant in other scenarios. The first is that the validation data is used to assess two different emulators; as such, using either validation design would prioritise validating one emulator over the other (or a new validation design would have to be developed to target both simultaneously). The second, is that both validation design schemes ignore the normality diagnostics. The skewness and the kurtosis do not depend on the intrinsic variance, and so do not benefit especially from targeted replication.

Nonetheless, validation design for stochastic emulators is an interesting topic for future research. There are multiple avenues for this future research (for example; using the lengthscales of both the mean and variance Gaussian processes to determine where unique points should be placed), but this is not pursued further here.

2.5 Leave-One-Out

In this chapter we assumed that an additional set of validation data could be obtained. This can be valuable, and the resulting data need not be wasted, as a final emulator can be constructed using the merged training and validation data.

However, in some circumstances it may not be possible to obtain a separate validation data set. In these cases, the training data could instead be used to directly validate the emulator. However, the emulator's ability to interpolate (and extrapolate) is not tested with such a strategy. This doesn't particularly matter for problems where the data points fill space sufficiently well, as the interpolative capability of the emulator is less important. However, in higher dimensions it can become difficult to sufficiently fill space, and so the ability of the emulator to interpolate (and extrapolate) becomes more important.

An alternative strategy, again using the training data, is to leave out one data point (or with the diagnostics that leverage replicated simulations; leave out all the simulations for one training location), refit the emulator, and use the new emulator to predict that location. Repeating this for every training data point (or every training location) can then yield many of the same diagnostics as with dedicated validation data. This strategy can be used to assess how well the emulator performs, including testing its interpolation ability, without a dedicated validation dataset.

The leave-one-out procedure can be a useful tool, and it has been used to assess the quality of emulators in practice (Williamson et al., 2017; Mohammadi et al., 2019). However, when possible, a distinct validation data set is preferable. Consider the example emulator in the left of Figure 2.29.

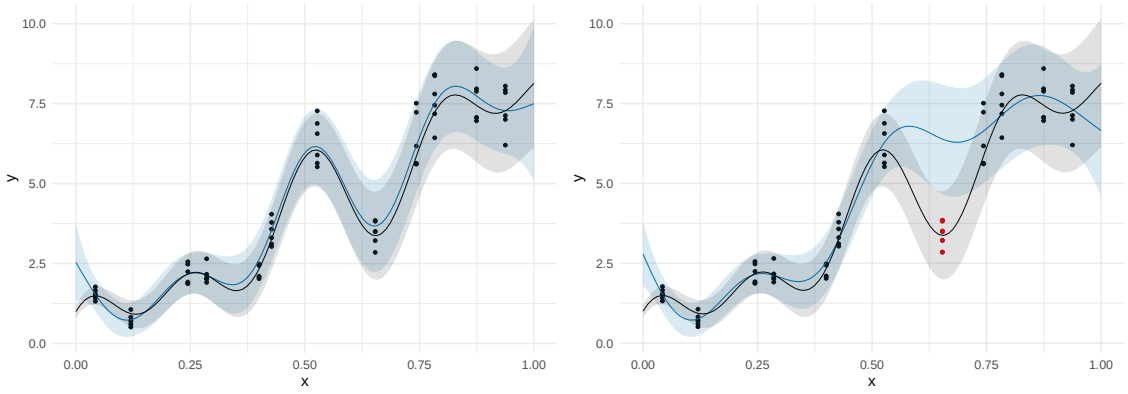


Figure 2.29: On the left is a heteroscedastic Gaussian process emulator (blue) and the truth(black). On the right is the same emulator after removing one unique training location (shown in red).

This emulator is a reasonably good surrogate for the simulator, and as such diagnostics should reflect that. However, if performing leave-one-out validation, we obtain the sample mean unexpectedness values in Figure 2.30.

Here there is one very extreme unexpectedness value (0.99998, which corresponds to a standardised error of -4.26), suggesting a serious local problem. However, this local problem is not present in reality. The reason for this is obvious in the plot in the right of Figure 2.29. Each time a training location is removed, the resulting emulator is actually different than the one we want to validate. In this case, the removal of one specific location results in the emulator entirely missing an important local pattern. In the original emulator, this local pattern was captured well by the exact training location which is now removed. This results in diagnostics which

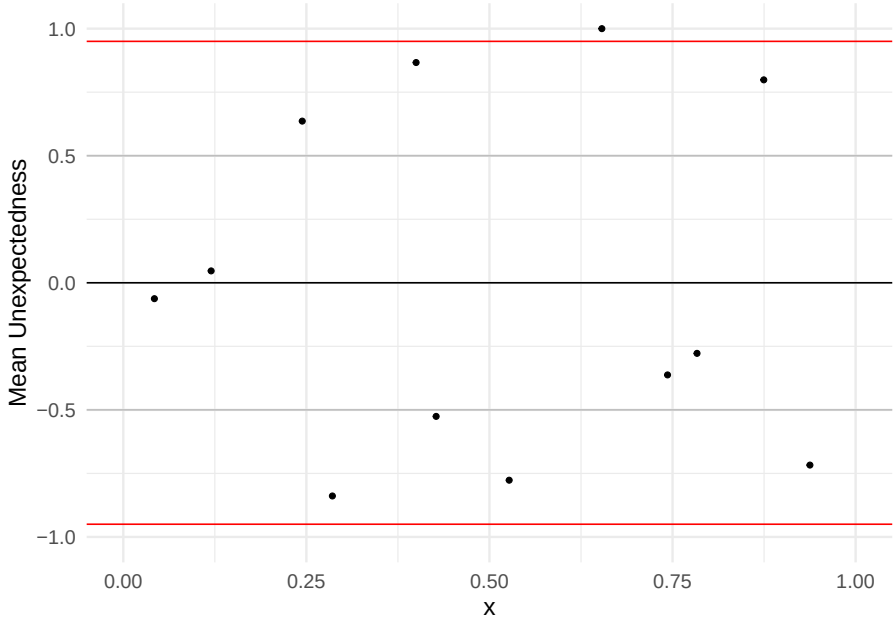


Figure 2.30: Leave-one-out sample mean unexpectedness against x for the emulator in the left of Figure 2.29.

are far more negative than they should really be, because the emulator we want to validate and the emulators we actually validate are not the same.

This issue is reduced when the number of unique training locations is high, as the removal of any individual one should not greatly affect the emulator. In high dimensions, however, it can be difficult to fill the input space enough to avoid this problem. Similarly, because leave-one-out diagnostics do not check any new regions of input space; if a local pattern is not captured by the original emulator (because no training points are nearby this local pattern), leave-one-out will not detect this (because none of the leave-one-out points will be nearby this region). A distinct validation set has a slightly greater ability to detect these problems (although its ability to identify highly local issues would still be weak).

As an important note, the normality diagnostics from the previous section (Section 2.2.3) could use the training data, rather than (or as well as) a dedicated validation data set, or leave-one-out validation. This is because the property being assessed (normality) is not dependent on the parameter estimates or interpolative capabilities of the emulator, and is instead an intrinsic assumption.

Overall, leave-one-out validation will often be a suitable alternative to a completely new validation data set, and is certainly an efficient use of data if the simulation budget is small, and the emulators are also (relatively) quick to fit. However, a

distinct validation data set remains a preferable option when practical.

2.6 Discussion

In this chapter we have discussed how stochastic emulators can be difficult to validate. Standard techniques are unable to identify key problems (at least without an abundance of data) and are unable to diagnose the *reasons* for a poorly fitting stochastic emulator. Our proposed solution involves using replicated validation simulator runs to individually assess the component assumptions and estimates of a stochastic emulator, providing greater insights into the performance of an emulator.

Previously developed emulator diagnostics can be adapted to apply to sample means; providing information regarding the emulator's mean predictions. We also developed additional diagnostics for the sample variance and the normality assumption. The developed validation framework could also be applied to other quantities of interest, which could be useful for other, non-normal, emulators.

It remains future work to develop more complex diagnostics that take the correlation structures into consideration, as with the Cholesky errors or Mahalanobis distance from the Bastos and O'Hagan (2009) deterministic diagnostics. Overall, the presence of independent noise reduces the importance and practicality of the correlation structures more generally; with the developed diagnostics proving to be reasonably sufficient.

A benefit of separately checking the mean and variance functions is that, if we are more interested in the mean than the variance, then we can prioritise the importance of the mean diagnostics (see Chapter 4). Similarly for the variance. On the other hand, one has to be very careful when checking components independently if one is only interested in overall emulator performance. If an emulator poorly predicts the mean at a certain point and poorly estimates the variance at the same point, then these issues can exacerbate each other. In Section 2.1 these issues masked each other, leading to reasonable overall predictions. It is, however, entirely possible that these issues could instead cause the overall predictions to be worse than what is seen in the independent diagnostics. In general, overestimated intrinsic variances will mask problems in the mean, and underestimated intrinsic variances will exacerbate problems in the mean. As such, additional care should be taken with underestimated intrinsic variance estimates, and these stochastic diagnostics should be used with,

rather than instead of, standard emulator diagnostics. In a similar vein, if one is less interested in individual component assumptions and estimates of an emulator and only cares about overall predictions, then the value in these new diagnostics is diminished.

It is also important to note that whilst the mean and variance diagnostics can be checked independently of each other, and the normality assumption can be checked independently of the mean and variance performance, it is not true that the mean and variance can be checked independently of the normality assumption. The mean and variance diagnostics both rely on the normality assumption in generating their reference distributions, and are therefore not valid if the normality assumption does not hold. For example, if the simulator is not normal and is instead very skewed, it would be possible to observe a very unexpected sample mean value even if the emulator's mean is actually reasonable. This means that it is important to check the normality assumption first (either via the independent normality checks, or via a conditional normality check such as a QQ plot) before making any conclusions about the mean and variance. Additionally, it can be useful to consider that the diagnostics check only the emulator's predictions, not the assumptions made for that emulator (with the normality diagnostics being an exception). A mean diagnostic which suggests the mean predictions are acceptable, does not necessarily imply anything about the assumptions used to obtain those predictions. For example, even if the 'correct' model structure was chosen for the variance process (and enough simulations were obtained), it could be possible to obtain overestimated intrinsic variances if a sufficiently poor model structure for the mean process was chosen. In this case, the variance diagnostics might discover an issue (because there is an issue), but increased modelling attention should be given to the mean process model, not the variance process model. As such, the diagnostics only point out issues with the component predictions, not whether poor modelling choices were made for that component.

An issue with these diagnostics (and any practical diagnostic method) is that it can still be possible to validate an invalid emulator, or invalidate a valid emulator. For example; validation coordinates can, by chance, be chosen in areas where the emulator is indeed valid, but problems may exist in other areas of the input space. Similarly, although a validation coordinate may be correctly chosen in an area where the emulator has poor fit, the validation simulator runs for this coordinate may,

simply by chance, provide sample means and sample variances that agree with the emulator's predictions. This is always a risk when dealing with random processes. With these (and many diagnostics tools), we only ever obtain evidence for (or against) the suitability of a model, we rarely ever know for certain whether a model is in fact accurate. We recommend 10 validation input locations per input dimension to fill space to some degree, hopefully avoiding this problem (following the same recommendation for fitting emulators in (Loeppky et al., 2009)), but this may not always be possible and in higher dimensions even more may be desirable. We also recommend 4 simulations per input location, enabling the use of all the diagnostics discussed here (but only 2 could be used if the simulation budget is especially low, facilitating only the mean and variance diagnostics, or leave-one-out could be performed instead if the training data includes replications). More investigation is required for determining how many unique locations is optimal for diagnostics more generally.

Additionally, it is important to note that no emulator will ever perfectly recreate a simulator. This raises frustrating questions about what the purpose of an emulator is, if all emulators are flawed. Ultimately the point of an emulator is for it to be fit for purpose, and as such, whether an emulator is valid depends on the experimental objectives. As such, hard rules have been (and should be) avoided. Instead, it is recommended to treat these, and other diagnostics, simply as information to advise on the decision to 'validate' or 'invalidate' an emulator; and to provide specific details about how an emulator misrepresents the simulator.

Overall, it is difficult to effectively validate a stochastic emulator, at least without an abundance of data (where proper validation is most important). Nonetheless, the ideas presented here appear to be capable and economical, providing a descriptive set of tools for assessing the fit of a stochastic emulator.

Chapter 3

Making use of Deterministic Approximations[1]

3.1 Introduction

A flaw with stochastic emulators, especially those which impose few assumptions (e.g. those that do not assume constant variance) is that far more data is required to properly estimate the shape of the mean (and the variance). A rule of thumb for deterministic emulators is that at least 10 data points per input dimensions are required to fit an emulator (Loeppky et al., 2009). For stochastic problems, many more than this are needed (Binois et al. (2019) use 500 data points when comparing different methods of choosing data point locations for a 1D toy stochastic simulator).

A need for more data is to be expected for more complex simulators, and stochastic simulators are certainty a more complex class of simulator, but such a high number of required runs can be prohibitive in practice. For example, consider the toy stochastic simulator given in Equation (3.1). This is similar (but not identical) to the toy simulator in Chapter 2.

$$\begin{gathered} y(x) = (1-x)\sin(\pi + 6\pi x) + \log(0.2+x) + (1.2-x)\epsilon; \\ \epsilon \sim N\big(0,1\big). \end{gathered} \tag{3.1}$$

We obtain the plot in the left of Figure 3.1 after evaluating this toy simulator on 50 points sampled from $[0,1]$ using a maximin Latin hypercube (McKay et al., 2000). The plot on the right of Figure 3.1 shows predictions for the mean and the 95%

[1]Much of this chapter's contents have been published in Baker et al. (2020a)

predictive intervals for the simulator output using a heteroscedastic Gaussian process emulator, as well as the true mean and 95% predictive intervals.

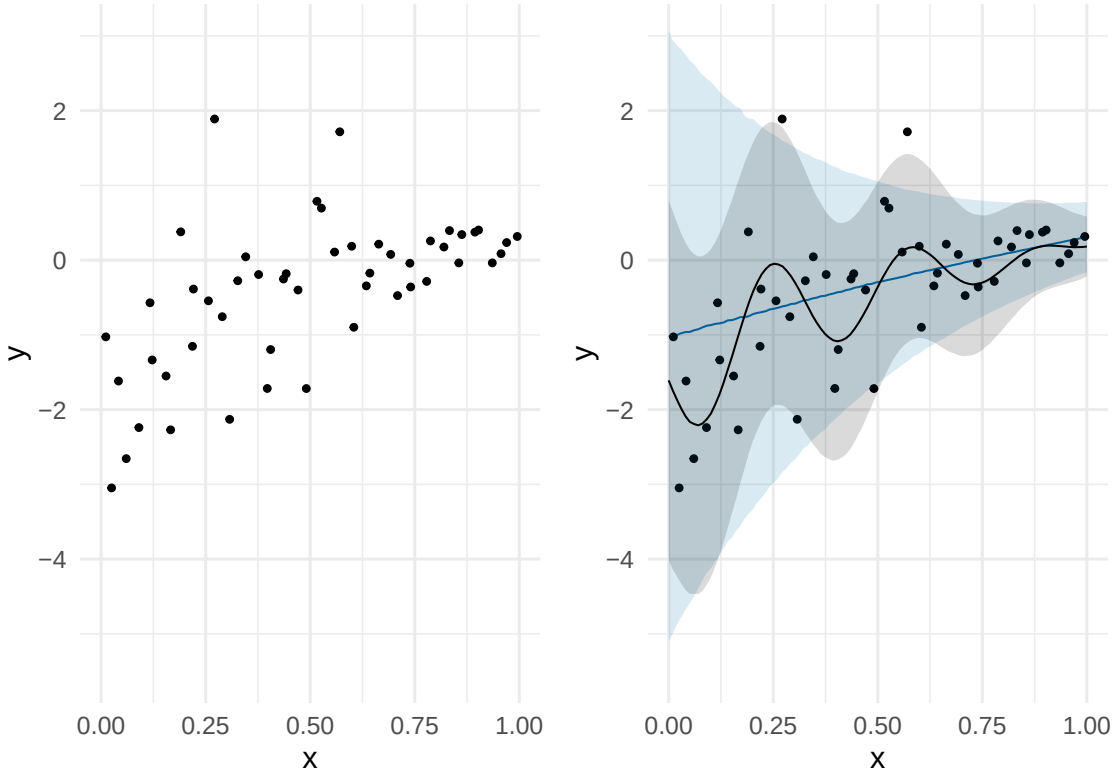


Figure 3.1: 50 Evaluations of the toy simulator from Equation (3.1) (left), and the respective heteroscedastic emulator predictions (right). The true mean and 95% interval are superimposed in black, and the emulator mean and 95% predictive interval are in blue.

The problem with this emulator is similar to that in the previous chapter: the mean is poorly estimated and the variance is overestimated. That this emulator fails is not surprising; the truth is not clear even after visually inspecting the simulator runs (which is only possible with low dimensional problems). Perceived trends can be true, or they can also just be artefacts of the stochasticity. This specific problem, which to some degree is present with any stochastic simulator, is similar to the problem of choosing the degrees of freedom for smoothing splines (Cantoni and Hastie, 2002).

To properly use the emulator as a surrogate for the simulator, this issue should be resolved, but obtaining a sufficient number of data points can be computationally expensive. In situations where additional prior knowledge of the simulator is available, a more descriptive prior mean function $m(\mathbf{x})$ could be used, providing information

that can assist in the prediction of the true mean. However, in practice, sufficient knowledge of the mean function can often be lacking, which is one reason why computer experiments are conducted in the first place.

The solution investigated here is one that leverages deterministic approximations of the stochastic simulator. Deterministic systems are much easier to emulate, and if the deterministic system is informative in some way about the stochastic simulator, then including this information can be worthwhile. This was initially motivated because EnergyPlus is originally deterministic, and so such an approximation already exists. However, the idea can be more generally applicable, as there are many ways a deterministic approximation could exist. Sometimes a deterministic version has intentionally been made; sometimes a different model of the same process exists but is deterministic; sometimes the computer model was once deterministic in its development history and stochasticity was added later on; and sometimes the underlying simulator is actually deterministic but some inputs are taken as random (as in our case). Ultimately, if both the deterministic approximation and the stochastic simulator are supposed to be modelling the same real world process, there should be reason to believe that the deterministic approximation contains relevant information about the stochastic model.

3.2 Model

For our toy model, we obtain a hypothetical deterministic approximation by replacing the random component ϵ with a fixed number (in this case ϵ is replaced with the fixed number 1, 0 has intentionally not been chosen to avoid an overly perfect approximation). The plot on the left of Figure 3.2 shows 12 runs of this toy deterministic approximation simulator (chosen via a maximin Latin hypercube design).

Because these runs are now samples from a deterministic simulator, a deterministic Gaussian process emulator can be used, and the predictions in the right of Figure 3.2 are obtained from such an emulator.

The shape of the emulator produced from this appears visually similar in shape to the true mean of the stochastic simulator from Figure 3.1. This is to be expected, as the true mean of the deterministic emulator is the true mean of the stochastic simulator, offset by $(1.2 - x)$. Visually observing the predictions from this deterministic

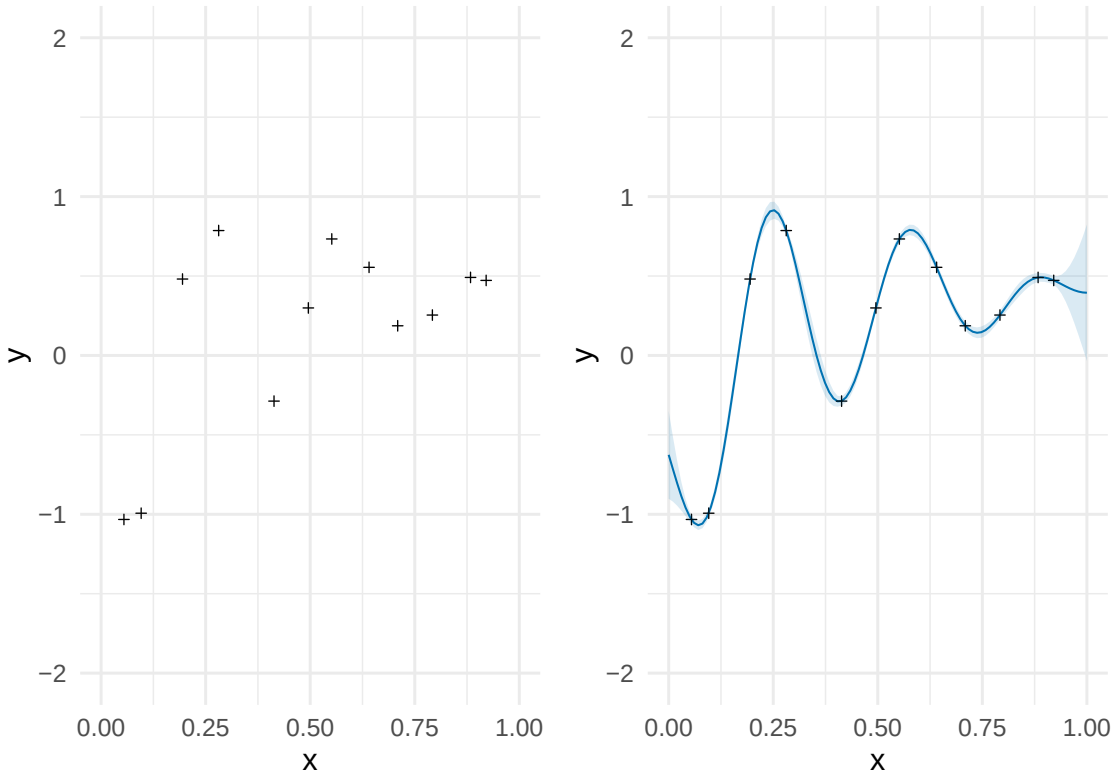


Figure 3.2: 10 Evaluations of the toy deterministic approximation simulator from Equation (3.1), and the respective deterministic emulator predictions (right).

emulator would suggest, even without knowledge of the truth, that the stochastic emulator's mean should not be (approximately) linear as it is in Figure 3.1. And so, we aim to formally incorporate this evidence.

We model the stochastic simulator as the deterministic simulator plus an independent noisy (heteroscedastic) adjustment term. Equations are given by Equation (3.2), with $y_D(\mathbf{x})$ representing the deterministic simulator, and $y_H(\mathbf{x})$ representing the stochastic simulator.

$$\begin{aligned} y_H(\mathbf{x}) = \; & y_D(\mathbf{x}) + \delta(\mathbf{x}); \\ & y_D(\mathbf{x}) \sim GP\big(m_D(\mathbf{x}), K_D(\mathbf{x}, \mathbf{x}')\big), \\ & \delta(\mathbf{x}) \sim GP\big(m_H(\mathbf{x}), K_H(\mathbf{x}, \mathbf{x}') + \sigma^2(\mathbf{x})I\big); \\ & \log(\sigma^2(\mathbf{x})) \sim GP\big(m_\sigma(\mathbf{x}), K_\sigma(\mathbf{x}, \mathbf{x}')\big), \end{aligned} \tag{3.2}$$

where $m_H(\mathbf{x})$, $m_D(\mathbf{x})$ and $m_\sigma(\mathbf{x})$ are prior mean functions, $K_H(\mathbf{x}, \mathbf{x}')$, $K_D(\mathbf{x}, \mathbf{x}')$, and $K_\sigma(\mathbf{x}, \mathbf{x}')$ are covariance functions, I is the identity matrix, and $\sigma^2(\mathbf{x})$ is the heteroscedastic intrinsic simulator variance.

The y_D component models the deterministic approximation and the δ component

models the difference between the stochastic simulator and the deterministic approximation. In this way, the deterministic approximation can inform about the overall shape of the stochastic emulator, but the adjustment term δ allows for differences to be captured. δ also incorporates the intrinsic variance modelled by $\sigma^2(\mathbf{x})$.

Using a sum of Gaussian processes to model a system is a common tool in deterministic emulation for a variety of applications. For calibration purposes, Kennedy and O'Hagan (2001) and Brynjarsdóttir and O'Hagan (2014) use one Gaussian process to model a simulator and another to model the discrepancy between the simulator and real life. To tackle non-stationarity, Ba et al. (2012) use one Gaussian process to model global simulator trends and another to model local deviations from this global pattern. Haaland et al. (2011) use the sum of Gaussian processes to model (prohibitively) large datasets; they use one Gaussian process to model a small subset of the data, and then additional Gaussian processes model the residuals from former Gaussian processes. Kennedy and O'Hagan (2000) then also use one Gaussian process to model a fast simulator and another to model the difference between the fast simulator and the slow simulator (they also allow for more than just two levels of simulator speed). In a way, the sum of Gaussian processes simply treats the prior mean function of one Gaussian process as another Gaussian process.

In this chapter, fitting this model using observed deterministic simulations $\mathbf{y}_D = y_D(X_D)$, and observed stochastic simulations $\mathbf{y}_H = y_H(X_H)$, proceeds as follows: we note that

$$\mathbf{y}_D \sim N\big(m_D(X_D), K_D(X_D, X_D)\big).$$

From here, we know (using the standard Gaussian process prediction equations) that

$$y_D(X_H)|\mathbf{y}_D \sim N\big(\mathcal{M}_D(X_H), \mathcal{V}_D(X_H)\big);$$

$$\mathcal{M}_D(X_H) = m_D(X_H) + K_D(X_H, X_D)K_D(X_D, X_D)^{-1}\big(\mathbf{y}_D - m_D(X_D)\big),$$
$$\mathcal{V}_D(X_H) = K_D(X_H, X_H) - K_D(X_H, X_D)K_D(X_D, X_D)^{-1}K_D(X_D, X_H).$$

And so,

$$\mathbf{y}_H \sim N\big(m_H(X_H) + \mathcal{M}_D(X_H), K_H(X_H, X_H) + \sigma^2(X_H)I + \mathcal{V}_D(X_H)\big)$$

(with $\log(\sigma^2(X_H))$ given a GP prior: $\log(\sigma^2(X_H)) \sim N\big(m_\sigma(X_H), K_\sigma(X_H, X_H)\big)$). These are then input into Stan (Stan Development Team, 2016) and the various

parameters (including the values of $\log(\sigma^2(X_H))$) are estimated at their posterior modes via the optimizing function. A simpler fitting procedure, given in Section 3.6 using Equations (3.9) - (3.11), can also be used.

Although the method described is more general, the examples in this chapter take the mean functions to be linear and the covariance functions as squared exponentials (e.g. for the δ Gaussian process, $m_H(\mathbf{x}) = \beta_{H_0} + \mathbf{x}^T\boldsymbol{\beta}_H$ and $K_H(\mathbf{x}, \mathbf{x}') = \alpha_H^2 \prod_{i=1}^{d} \exp(-\frac{(x_i - x_i')^2}{l_{H_i}})$, where d is the dimension of $\mathbf{x}$, α_H is a standard deviation parameter, and l_{H_i} are length scale parameters). Specific choices for a Gaussian process are often based on personal preference; some authors prefer more complicated mean functions (Vernon et al., 2010), others choose a zero mean function (Binois et al., 2018); and then there are many choices for the covariance function with various justifications (Rasmussen and Williams, 2006). These choices, while important, are outside the scope of this work. The linear mean function coefficients (β_{H_0} and $\boldsymbol{\beta}_H$ for the δ Gaussian process, β_{σ_0} and $\boldsymbol{\beta}_\sigma$ for the $\log(\sigma^2)$ Gaussian process, and β_{D_0} and $\boldsymbol{\beta}_D$ for the y_D Gaussian process) are given $N(0, 10)$ priors; the standard deviation parameters (α_H, α_σ and α_D for the δ, $\log(\sigma^2)$, and y_D Gaussian processes respectively) are given Inverse-Gamma$(2, 1)$ priors; and the length scales (l_{H_i}, l_{σ_i} and l_{D_i}) are given Gamma$(4, 4)$ priors. These choices can also change depending on personal preference; in this case they were chosen to broadly cover the range of reasonable values, while still allowing somewhat unexpected values to be possible.

For the standard heteroscedastic 'HetGP' model used previously, and those used later, parameters were also taken fixed at their posterior modes using via the optimizing function in Stan, using the same priors as used for the model which includes deterministic runs (which we now refer to as DetHetGP).

For predictions, predictive samples can be obtained from the latent log variance Gaussian process and the y_D Gaussian process, before then sampling from the δ Gaussian process (and thus the full model).

For the deterministic Gaussian process, predictions conditional on known deterministic runs $\mathbf{y}_D$ are standard (Rasmussen and Williams, 2006) and are given in

Equation (3.3)

$$
\begin{aligned}
y_D(X^*) \mid \mathbf{y}_D &\sim N\big(\mathcal{M}_D(X^*),\ \mathcal{V}_D(X^*)\big); \qquad (3.3)\\
\mathcal{M}_D(X^*) =& m_D(X^*)+ \\
& K_D(X^*, X_D)K_D(X_D, X_D)^{-1}\big(\mathbf{y}_D - m_D(X_D)\big), \\
\mathcal{V}_D(X^*) =& K_D(X^*, X^*)- \\
& K_D(X^*, X_D)K_D(X_D, X_D)^{-1}K_D(X_D, X^*).
\end{aligned}
$$

Predictions for the intrinsic (log) variance, conditional on (estimated) values at the input points $\sigma^2(X_H)$ are also standard, and given by Equation (3.4)

$$
\begin{aligned}
\log(\sigma^2(X^*)) \mid \log(\sigma^2(X_H)) &\sim N\big(\mathcal{M}_\sigma(X^*),\ \mathcal{V}_\sigma(X^*)\big); \qquad (3.4)\\
\mathcal{M}_\sigma(X^*) =& m_\sigma(X^*)+ \\
& K_\sigma(X^*, X_H)K_\sigma(X_H, X_H)^{-1}\big(\log(\sigma^2(X_H)) - m_\sigma(X_H)\big), \\
\mathcal{V}_\sigma(X^*) =& K_\sigma(X^*, X^*)- \\
& K_\sigma(X^*, X_H)K_\sigma(X_H, X_H)^{-1}K_\sigma(X_H, X^*).
\end{aligned}
$$

Predictions for the $\delta(X^*)$ component are then the standard heteroscedastic predictions (Chapter 1) conditioned on values for $\mathbf{y}_H - y_D(X_H)$ rather than just $\mathbf{y}_H$. This leads to predictions for the heteroscedastic Gaussian process $y_H(X^*)$ having the form in Equation (3.5)

$$
\begin{aligned}
\delta(X^*)\mid \mathbf{y}_H, \sigma^2(X^*), \sigma^2(X_H), y_D(X_H) &\sim N\big(\mathcal{M}_H(X^*),\ \mathcal{V}_H(X^*)\big); \qquad (3.5)\\
\mathcal{M}_H(X^*) =& m_H(X^*)+ \\
& K_H(X^*, X_H)\big(K_H(X_H, X_H) + \sigma^2(X_H)I\big)^{-1}\big(\mathbf{y}_H - y_D(X_H) - m_H(X_H)\big), \\
\mathcal{V}_H(X^*) =& K_H(X^*, X^*) + \sigma^2(X^*)I- \\
& K_H(X^*, X_H)\big(K_H(X_H, X_H) + \sigma^2(X_H)I\big)^{-1}K_H(X_H, X^*).
\end{aligned}
$$

Using Equation (3.3) to obtain predictive samples for $y_D(X_H)$. Predictions for the stochastic simulator output can then be drawn from:

$$
y_H(X^*) \mid \mathbf{y}_H, \sigma^2(X_H), \mathbf{y}_D = y_D(X^*) + \delta(X^*). \qquad (3.6)
$$

Using Equations (3.3) and (3.5) to obtain predictions for $y_D(X^*)$ and $\delta(X^*)$. Importantly, each sample of $y_D(X^*)\mid \mathbf{y}_D$ and $y_D(X_H)|\mathbf{y}_D$ should be from the same draw from the underlying Gaussian process (i.e. $y_D(X^*)\mid \mathbf{y}_D$ and $y_D(X_H)|\mathbf{y}_D$ should not be drawn independently).

To be clear, this means predictions follow the process as given below:

Algorithm 1: Algorithm for obtaining m_n sample predictions from the DetHetGP model.

for $i = 1$ *to* m_n **do**

sample $y_D(\binom{X^*}{X_H})|\ \mathbf{y}_D$ once from Equation (3.3);

sample $\log(\sigma^2(X^*))|\ \log(\sigma^2(X_H))$ once from Equation (3.4);

sample $\delta(X^*)|\ \mathbf{y}_H, \sigma^2(X^*), \sigma^2(X_H), y_D(X_H)$ once from Equation (3.5);

Combine obtained samples of $y_D(X^*)$ and $\delta(X^*)$ using Equation (3.6) to obtain one sample for $y_H(X^*) \mid \mathbf{y}_H, \sigma^2(X_H), \mathbf{y}_D$, and save the result;

end

Alternatively, analytical predictions can be made by taking point estimates for the variance GP predictions, and then using the formulation given in Section 3.6.

To showcase this method, we return to the toy simulator from before, using the previously obtained stochastic and deterministic data points (and standardising according to the stochastic data's mean and standard deviation). Predictions from this model are given in Figure 3.3.

The overall shape of this emulator is much closer to the truth, with the periodic feature now represented in the emulator. This confirms that including deterministic runs can be valuable in building a stochastic emulator. The standard heteroscedastic GP used earlier (and later in this chapter) used the same priors, mean functions, and covariance functions; and predictions were also made via sampling (including the predictive uncertainty in the intrinsic variance predictions).

For this specific example, the deterministic approximation yields outputs that are generally too large compared to the stochastic simulator's mean (not surprising since we set up the problem this way), and thus the same is true for the y_D Gaussian process, the mean predictions of which are represented by the orange, line. This not a problem however, as it is adjusted by the δ component to ensure the emulator's mean and the stochastic simulator's mean match. The complexity of this adjustment Gaussian process δ is related to how good an approximation the deterministic simulator is, and therefore the capability of the overall emulator depends on the accuracy of the deterministic approximation. In this toy example, the deterministic approximation linearly differs from the stochastic simulator's mean, and thus the δ Gaussian process needs to be approximately linear for the full emulator to have good fit.

This "DetHetGP" emulator is evidently a better surrogate for the simulator than

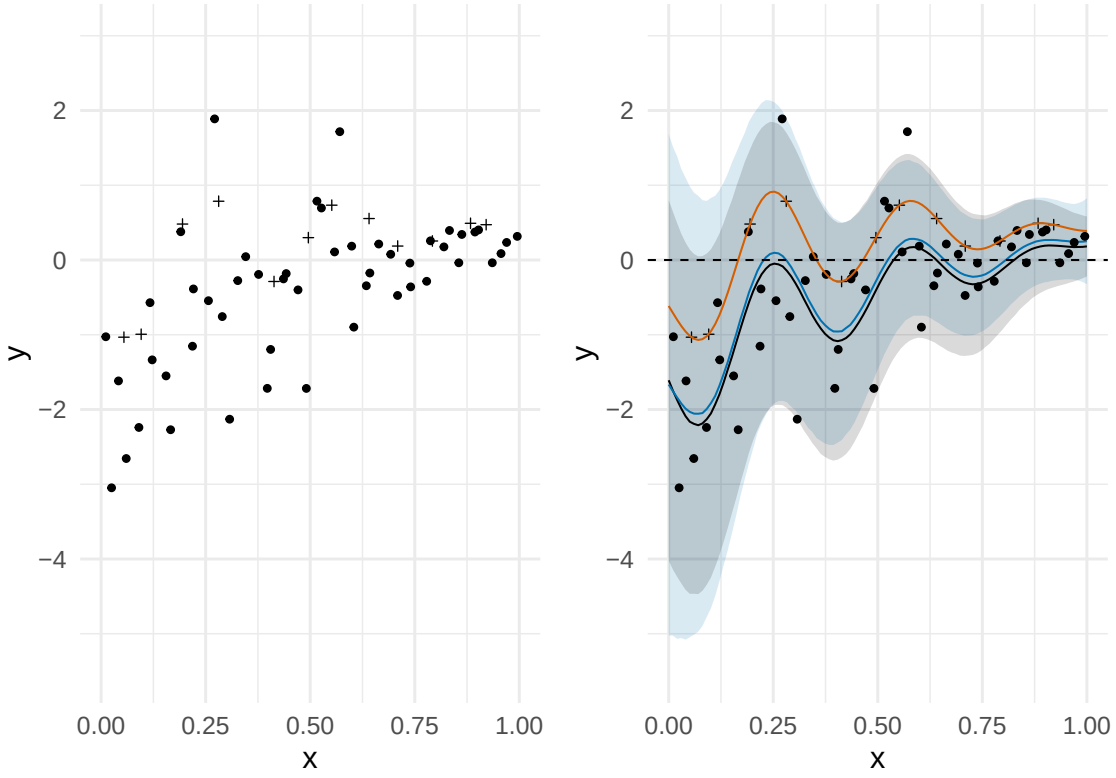


Figure 3.3: Emulator predictions for the toy simulator, using both stochastic and deterministic runs. The black interval and respective line are the true 95% interval and mean, and the blue interval and line are the emulator 95% predictive interval and mean. The stochastic data points are circles, and the deterministic data points are plus symbols. The mean of the y_D component is in orange and interpolates the deterministic data points.

the previous heteroscedastic emulator. On the other hand, it has used an additional twelve simulator runs to be so. To show that utilising deterministic runs was an efficient use of a simulator budget, Figure 3.4 shows the base heteroscedastic emulator fit to the original 50 stochastic data points, plus an additional twelve stochastic data points generated from the same input values as the deterministic runs.

Here, the HetGP emulator remains substantially inferior to the emulator that uses some of the simulator budget to incorporate deterministic runs, failing to capture the periodic component, despite using the same total number of simulator runs.

3.3 Guidelines

The previous section showed that including information from a deterministic approximation can be useful, but this does not always mean that it is always so. This

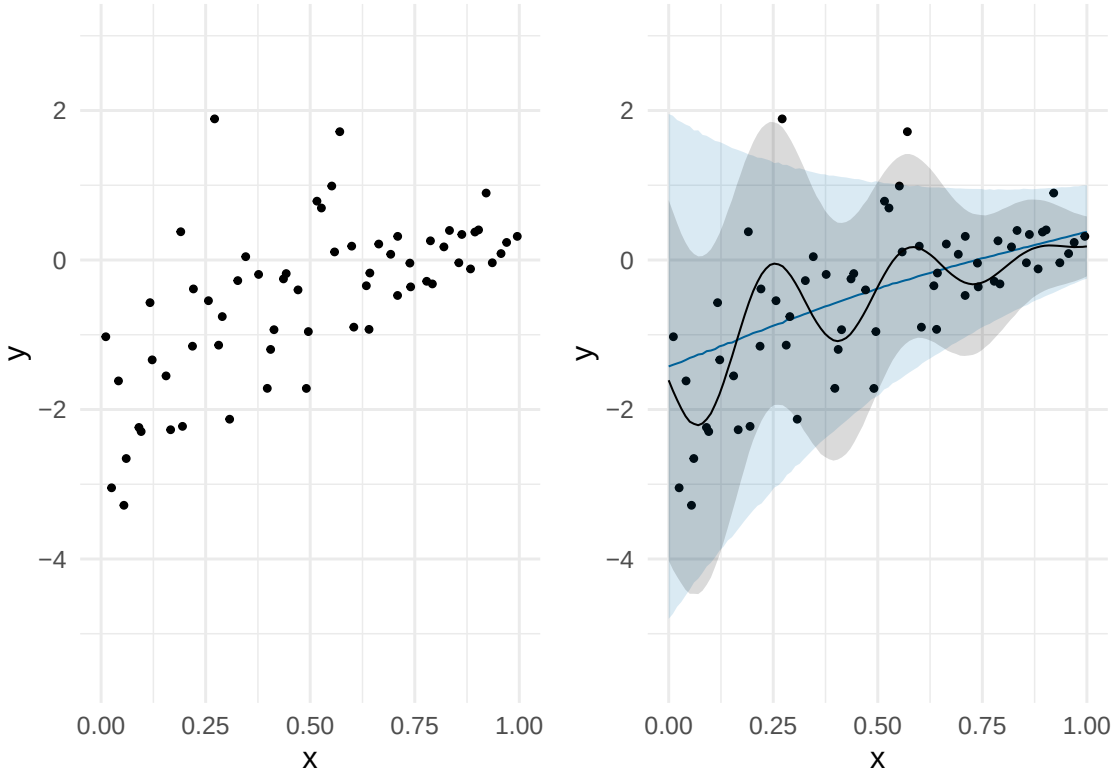


Figure 3.4: 62 Evaluations of the toy simulator from Equation (3.1) (left), and the respective heteroscedastic emulator predictions (right). The more periodic true mean and 95% interval are superimposed in black, and the more linear emulator mean and 95% predictive interval are in blue.

section provides three criteria that must be satisfied for the outlined method to perform well, and advice on how to ensure they are satisfied. Also shown will be examples of what outcomes can occur if a criterion is not met. This section is not intended to downplay the outlined method, but it does serve to provide an intuitive understanding and actionable advice to ensure the method will work in practice.

The three criteria that we identify, in descending order of perceived importance, are:

1. **The deterministic approximation must be informative in some way.**
2. **There must be a sufficient number of deterministic simulation runs.**
3. **There must be a sufficient number of stochastic simulation runs.**

3.3.1 Criteria 1 - Accurate Approximation

The first criterion is fairly straight forward: an assumption is made that the deterministic approximation provides some information about the shape of the stochastic simulator. This criteria demands some degree of expert prior knowledge. In general, we would hope that if the deterministic approximation and the stochastic simulator are both models of the same process, that common attributes should be present in both. It would be concerning if two models of the same process had no similarities.

To show what can occur should this criteria fail, we return to the toy simulator in Section 3.2, but this time use the following deterministic approximation:

$$y(x) = \log(0.1 + 4x). \tag{3.7}$$

This bears no resemblance to the stochastic simulator, and is ultimately just a different simulator, and not an approximation.

Fitting the statistical model, using 50 stochastic points and 12 deterministic points (and standardising the data according to the stochastic points), provides the predictions in Figure 3.5

One can see from this plot that the deterministic points appear in an "r" shaped curve, and so the y_D component tracks this. The resulting emulator uses this as a baseline for what the overall stochastic trend is; the outcome being that the emulator incorrectly believes the mean of the stochastic simulator is also an "r" shaped curve. It is in this way that a deterministic code which does not approximate the stochastic simulator can misinform the emulator into believing something which is untrue. This does not appear to be too problematic in Figure 3.5, but if the standard heteroscedastic Gaussian process emulator *was* able to capture the correct trend, a poor deterministic "approximation" might instead bias it into no longer learning the correct trend.

Our advice here is to only use a deterministic approximation if there is a real reason to expect some similarity between the deterministic approximation and the stochastic simulator, such as with the four example situations suggested in Section 3.2.

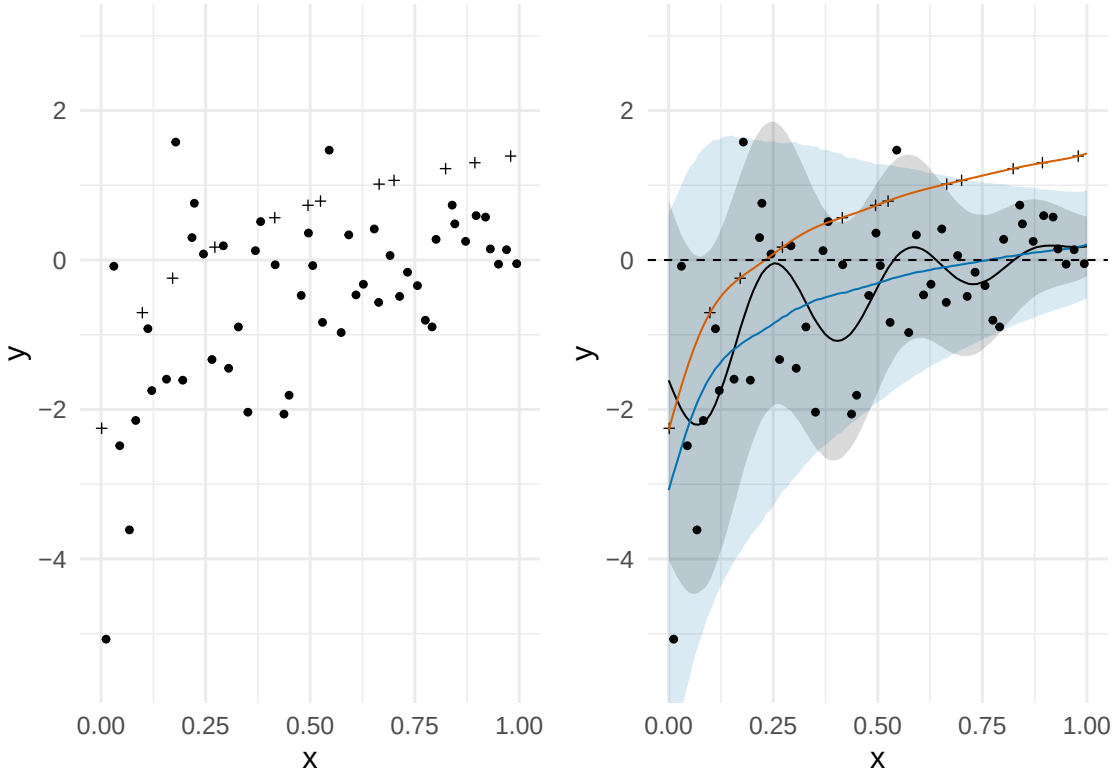


Figure 3.5: Emulator predictions for the toy simulator, using both stochastic and deterministic runs. The deterministic simulator here bears no resemblance to the stochastic simulator. The black interval and respective line are the true 95% interval and mean, and the blue interval and line are the emulator 95% predictive interval and mean. The stochastic data points are circles, and the deterministic data points are plus symbols. The mean of the y_D component is in orange and interpolates the deterministic data points.

3.3.2 Criteria 2 - Enough Deterministic Runs

Failing the second criteria yields similar effects as failing the first criteria. Should there be too few deterministic simulations, the y_D component can fail to capture the information contained within the deterministic approximation, and instead learn a different relationship. This different relationship may not be particularly informative about the shape of the stochastic simulator, but will nonetheless be used to inform predictions of the stochastic simulator.

Fitting the statistical model, using the exact same toy stochastic simulator and deterministic approximation from Section 3.2, but now with only 4 deterministic points (and 58 stochastic points), provides the predictions in Figure 3.6

In this plot, the y_D component obtains a relationship which is not particularly

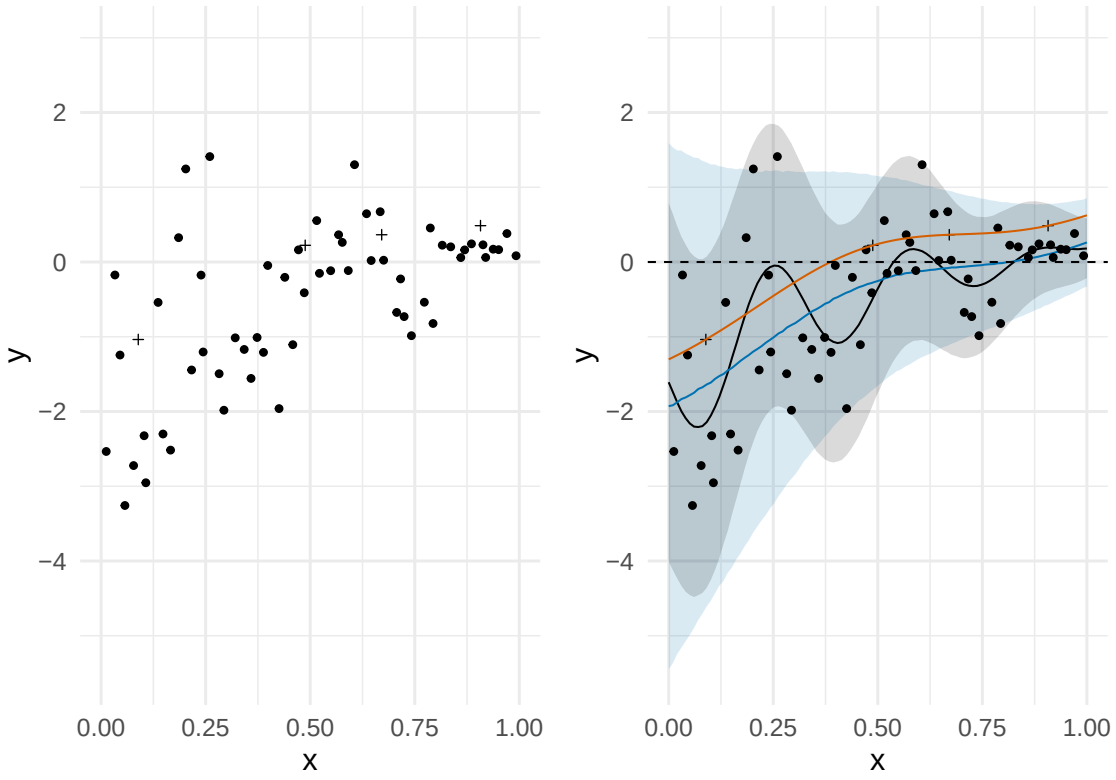


Figure 3.6: Emulator predictions for the toy simulator, using both stochastic and deterministic runs. Only 4 deterministic runs were used. The black interval and respective line are the true 95% interval and mean, and the blue interval and line are the emulator 95% predictive interval and mean. The stochastic data points are circles, and the deterministic data points are plus symbols. The mean of the y_D component is in orange and interpolates the deterministic data points.

similar to the overall mean of the stochastic simulator. The relationship that the y_D component obtains still acts as a baseline for the overall mean, and there is not enough stochastic data to rectify this. This is effectively the same problem as when Criteria 1 fails: there is little difference between having a bad deterministic "approximation", and having a good deterministic approximation but obtaining a bad representation of it.

Our advice here is based on the rule-of-thumb from Loeppky et al. (2009). Because satisfying Criteria 2 is essential, we recommend a more conservative 20 simulation runs per dimension, if possible, to capture the relationship of the deterministic approximation. We recommend this more conservative rule, as the potential missed opportunity from failing to capture the deterministic approximation is large. This rule is followed in Section 3.4. A more thorough practice, would be to fit a separate

standard deterministic emulator to the deterministic simulator; and ensure that enough deterministic data is obtained to pass certain validation checks (such as those outlined in Bastos and O'Hagan (2009), or simpler leave-one-out cross validation checks (see Chapter 2 and Williamson et al. (2017) for examples)). After this, one can feel confident that enough deterministic data has been obtained.

3.3.3 Criteria 3 - Enough Stochastic Runs

The final criteria deals with the opposite problem to Criteria 2. Stochastic runs are essential to modify the deterministic approximation such that it agrees with the stochastic simulator. Stochastic runs are also needed to estimate the variance of the stochastic simulator.

Figure 3.7 presents the predictions obtained from fitting the emulator to the toy simulator and deterministic approximation when only 6 runs were stochastic (and 56 were deterministic).

This plot shows the emulator failing to adjust the location and shape of the deterministic approximation correctly - it has been decreased too little for small values of x and too much for large values of x. Additionally, the variance is estimated poorly - especially for large values of x where it is estimated far too small. This serves as an extreme example, where such a huge percentage of the simulation budget is assigned as deterministic points, but it clearly establishes that having a sufficiently large number of stochastic runs is still important.

Our advice here is that, after the required number of deterministic runs for Criteria 2 to be satisfied are obtained, the remainder of the simulation budget should be assigned as stochastic points, and no more assigned as deterministic points.

3.3.4 Large Simulation Budgets

Including deterministic runs was motivated by low simulation budget scenarios; where not enough data can be obtained to capture the true relationship. If the simulation budget is large enough, it is not unreasonable to assume that including deterministic runs will no longer be optimal.

Deterministic runs primarily provide information about the shape of the mean (although better capturing the mean can often aid in capturing the variance). Since Criteria 2 and 3 directly act against each other, it is not impossible for deterministic

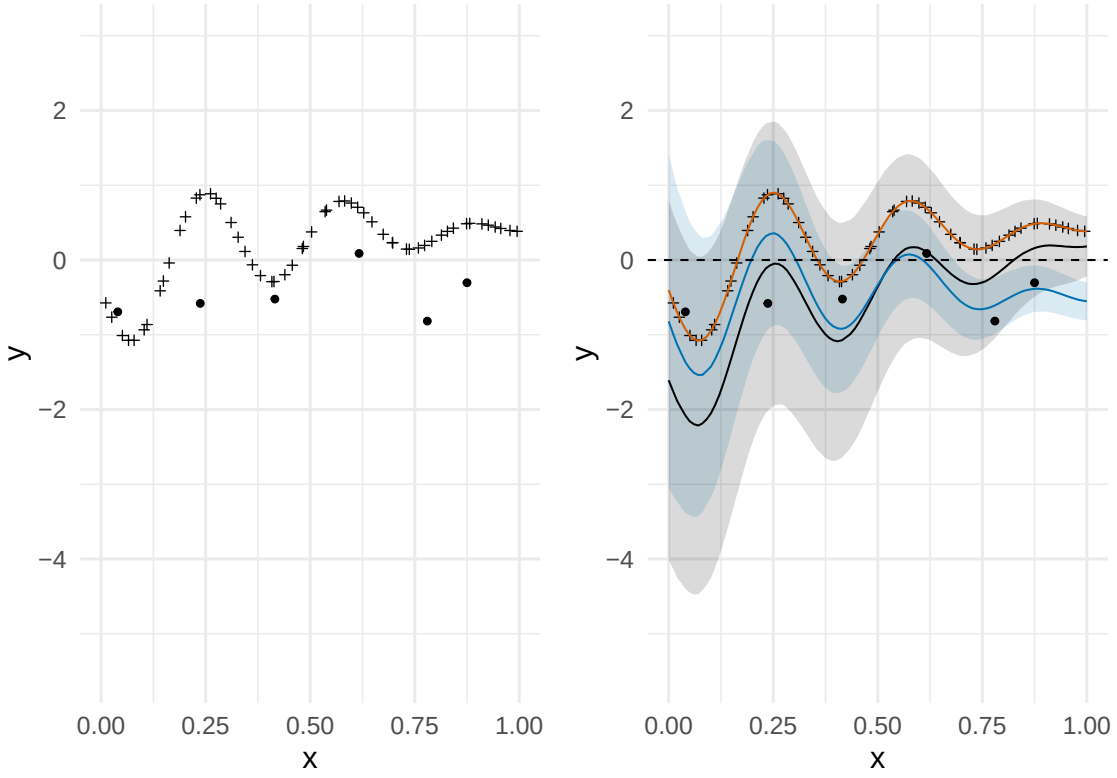


Figure 3.7: Emulator predictions for the toy simulator, using both stochastic and deterministic runs. Only 6 stochastic runs were used. The black interval and respective line are the true 95% interval and mean, and the blue interval and line are the emulator 95% predictive interval and mean. The stochastic data points are circles, and the deterministic data points are plus symbols. The mean of the y_D component is in orange and interpolates the deterministic data points.

runs to worsen the quality of a stochastic emulator, and as the simulation budget increases, this possibility becomes more likely. Initially this can arise in the form of a trade-off, where the deterministic approximation still contains additional information about the mean and so including deterministic runs can improve the estimation of the mean, but the resulting decrease in the number of stochastic runs can worsen the estimation of the intrinsic variance. The examples in the next section provide some illustration of this trade-off in practice. At what point this trade-off begins, and how strong this trade-off is, will depend on the accuracy of the deterministic approximation, the complexity of the mean, the complexity of the intrinsic variance, and the magnitude of the intrinsic variance. These can all be difficult to know about a-priori (although some intuition may be present), which makes the outlined method best suited for situations where the simulation budget is presumed to be overly

limited, or situations where the mean is of greater interest.

Should the simulation budget be very large (or conversely should the simulator be overly simplistic), the information provided by the deterministic approximation can become obsolete because the stochastic simulations alone are enough to learn the various relationships. In these situations, running any deterministic points can be wasteful, as improvements would be made by instead increasing the number of stochastic simulations. For an example of this, see Section 3.4.2. This is not a particularly surprising outcome, especially since the outlined method was largely motivated by limited simulation budget settings. In large simulation budget settings, whether or not to include deterministic simulations should not be too troubling, as both choices should provide capable emulators; with a marginal improvement distinguishing the two.

3.4 Examples

In this section, the method will be applied to two examples that are more realistic.

To compare the performance of the standard heteroscedastic Gaussian process (HetGP) and the developed method (DetHetGP), we will use out-of-sample simulation runs to assess the predictions of a statistical model. Two metrics will be used: the root mean squared error (RMSE), which assesses the mean of the emulator; and the "score" which assesses predictions according to both their mean and variance. The RMSE is the square root of the mean squared difference between the observed out-of-sample simulator runs and the predictive mean of the emulator; and the score is from Equation (27) in Gneiting and Raftery (2007). Both of these were used in Binois et al. (2018) to compare stochastic emulator predictions. Smaller values for the RMSE indicate an improved mean function, and larger values for the score indicate better overall predictions.

All simulation data sets are obtained with input points chosen by a maximin Latin hypercube design, unless otherwise stated.

3.4.1 Susceptible-Infected-Recovered Simulator

The first simulator we investigate is a basic Susceptible-Infected-Recovered (SIR) model using the Individual Contact Model (ICM) from the EpiModel package (Jenness

et al., 2018). This stochastically models a population where individuals can be: susceptible to some disease, currently infected with the disease, or recovered from (and now immune to) the disease. This is a fast simulator, but it serves as a more authentic example than the toy simulator in the previous sections. The output we consider is the number of infected people after 300 time steps. The two parameters we vary and use as our inputs for the emulators are: the probability of infection which will be allowed to vary between 0.5 and 1, and the recovery rate which will vary between 0 and 0.01 (for the emulators, both are scaled to be between 0 and 1). Other parameters exist in the model, such as the rate of interactions between individuals, the initial number of infected, and the total population (taken fixed as 0.01, 5, 1000 respectively). The package also includes a Deterministic Compartmental Model (DCM), which is a different simulator, but takes the same inputs and provides the same outputs as the stochastic simulator, and this will be our deterministic approximation. Note that this is the case where a deterministic approximation is available because an alternative deterministic model of the same real world process exists.

HetGP is fit using 120 stochastic simulations, and DetHetGP is fit using 80 stochastic simulations and 40 deterministic simulations (i.e. 20 deterministic points per input dimension, with the rest of the budget assigned as stochastic runs). Both data sets are standardised according to the stochastic data mean and standard deviation. To compare the two emulators, we obtain 200 more simulation runs, and then calculate the RMSE and score. Because this simulator is relatively cheap, we can repeat this entire procedure 100 times; smoothing out the variation from different Latin hypercube realisations and different simulation runs. We can then present summary statistics of the RMSE and score.

Table 3.1 gives these values.

Here we can see that, numerically, the score is marginally improved for DetHetGP and the $RMSE$ is more substantially improved (in terms of proportional improvements), with the upper quartile for the DetHetGP RMSE smaller than the median for HetGP. Additionally, not only does DetHetGP appear to be the preferred emulator here, but the deterministic approximation is much faster than the stochastic version. The stochastic simulator takes approximately 0.18 seconds to run, whereas the deterministic approximation takes approximately 0.05 seconds to run. It is not uncommon for a deterministic approximation to be cheaper, such simulators are

		Lower Quartile	Median	Upper Quartile
RMSE	HetGP	10.65	11.76	12.90
	DetHetGP	9.81	10.64	11.34
Score	HetGP	-1034.3	-1012.2	-993.7
	DetHetGP	-1036.5	-1009.7	-986.7

Table 3.1: The summary statistics of the score for both HetGP and DetHetGP from the simulation experiment conducted on the SIR simulator.

likely to be simpler, and thus faster. This implies that more simulation runs could be afforded if more deterministic runs were chosen; in this case, one stochastic run costs more than 3 deterministic runs. To provide more conservative results, we ignore this advantage in these comparisons, despite the potential improvement it provides DetHetGP.

Because the score is only marginally improved (and potentially within the margin of natural variability from only repeating the comparison 100 times, -1009.7 for DetHetGP and -1012.2 for HetGP are not particularly different when the interquartile ranges for the scores were 49.8 and 40.6), but the root mean squared error is substantially improved, this example is potentially a case where the trade-off between improved mean and improved variance occurs.

To visually show the improvements DetHetGP yields, Figure 3.8 shows predictions from both emulators (for one fit) with the infection rate x_1 kept constant at 1, and only the recovery rate is varied. Superimposed on this plot are 100 out of sample simulator runs where the infection rate was also kept fixed at 1.

The mean predictions from HetGP are approximately linear, missing the sharper increase for lower values of the recovery rate, and instead a larger variance is predicted for these values - large observed simulator runs were probably interpreted by the emulator as being because of a larger variance rather than because of a larger mean. DetHetGP on the other hand, does estimate the sharp increase in the mean function, and does not feature an overly large variance estimate for low values of the recovery rate. Other cross-sections of the emulator's prediction surface could have been presented, and a similar pattern exists for when the recovery rate is fixed at 0 and the infection rate is allowed to vary. These plots show how DetHetGP can provide a significant advantage over HetGP.

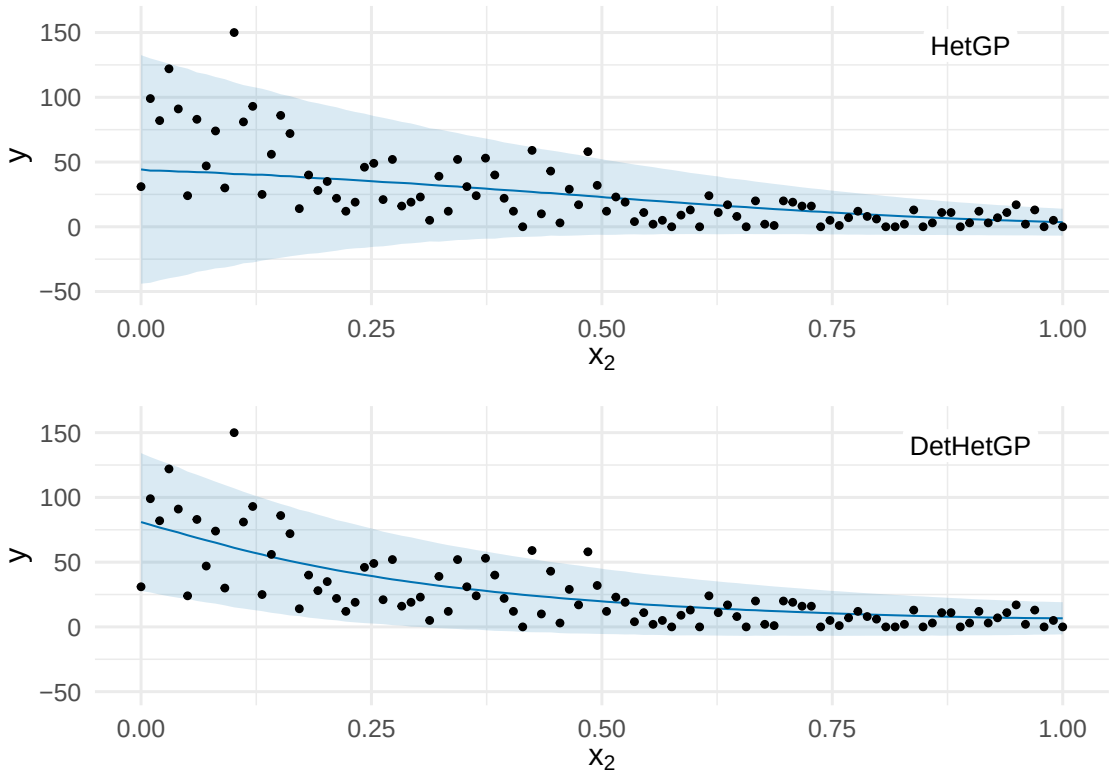


Figure 3.8: Emulator predictions for both HetGP (top) and DetHetGP (bottom). The predictions are for simulator runs where the infection rate is constant at 1, and only the recovery rate varies. Also superimposed are 100 additional simulator runs where the infection rate is kept constant at 1.

To better present the differences between the HetGP predictions and the DetHetGP predictions for this example, 2D colour plots are given in Figure 3.9. The top row presents the mean predictions and the bottom row presents the predictive standard deviations.

The HetGP plots show that the mean is relatively constant, but the standard deviation spikes in the bottom right corner. For DetHetGP, this relationship is flipped; the standard deviation is mostly constant, and the mean spikes in the bottom right corner. Given the numerical and visual evidence from previously, we can determine that DetHetGP is more accurate, and that HetGP has discovered the wrong relationship. Whether this specific difference is important will depend on the application of the emulator, but it seems clear that DetHetGP is better learning the underlying processes here. Note that this is an example of non-stationarity - where different regions of space have different correlation structures. In most of the space very little occurs, but in the bottom right there is a large degree of change; perhaps

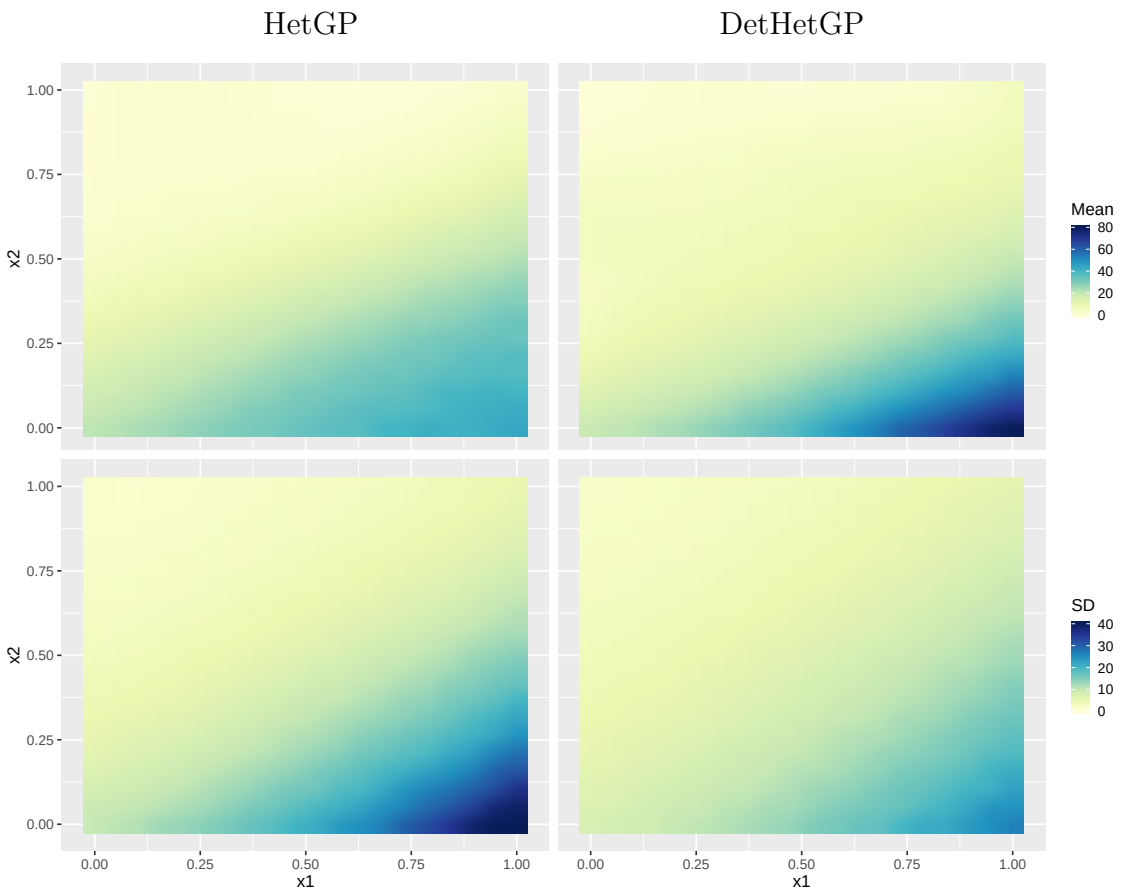


Figure 3.9: Emulator predictions for both HetGP (left) and DetHetGP (right) for the SIR model. Top are mean predictions and bottom are predictive standard deviations.

the reason for the failings of HetGP here. Non-stationarity is a well researched problem (Gramacy and Lee, 2008; Ba et al., 2012; Volodina and Williamson, 2020), and it would be interesting to see if properly accounting for non-stationarity would provide improved results (in either the y_D or δ component of DetHetGP). This example (and many of the other examples in this thesis) shows more generally how 'variability' can be exchanged between the mean model and intrinsic variance model in a stochastic emulator, which is something DetHetGP can protect against.

3.4.2 EnergyPlus

For the second example, we return to EnergyPlus. The specific building we model here is the hospital as in Chapter 2, and the input variables considered are: wall concrete thickness, wall insulation thickness, roof insulation thickness, floor concrete thickness, and window size (as a percentage of the total wall height). These are varied from 0m to 0.7m, 0m to 0.5m, 0 to 0.5m, 0.1 to 0.7m and 0.25 to 0.75% of the wall size. Weather generation is the same as in Chapter 2.

200 data points are used to fit the two emulators, with each run taking roughly 1 minute. For DetHetGP 100 of the data points will be deterministic runs (20 per dimension), and 100 will be stochastic. For HetGP the same stochastic runs will be used, as well as an additional 100 stochastic points with input values the same as those for the deterministic points. The data sets are also standardised according to the sample mean and standard deviation of the stochastic data points.

This simulator is more expensive than the SIR simulator, and so only one set of 500 out-of-sample simulation runs are generated. For these, DetHetGP receives a score of -5894 and an $RMSE$ of 212, whereas HetGP receives a score of -5968 and an $RMSE$ of 261. DetHetGP performs better than HetGP on the building model in this case, although it is hard to assess the magnitude of this numerical improvement without a scale or repeated experiments. Additionally, 500 out-of-sample simulation runs is actually quite sparse in 5D, and so does not tell the complete story of the two methods performances.

To further investigate this difference, Figure 3.10 shows the emulator predictive distribution for wall insulation thickness, keeping all other inputs fixed (at 0.5). 100 additional out-of-sample simulator runs are also shown, where all other input points are also fixed at 0.5.

The predictions from DetHetGP are characterised by a decrease in energy usage for very low values of wall insulation, after which improvements in energy efficiency seem to stabilise. This is consistent with our experience of the building model. HetGP does not yield this characteristic, instead it is more defined by a decrease in the variability of the energy usage as wall insulation thickness increases. If we were to assume that the trend discovered by DetHetGP is more accurate, it is likely that extremely large observed values of energy usage were instead assumed by HetGP to be the result of an increased variance, rather than a sharp increase in the mean. This is a similar result as found with the SIR example. The superimposed out-of-sample data points agree more with the predictions from DetHetGP, suggesting that this emulator is better. The data suggests the sharp increase for low values of wall insulation thickness, predicted by DetHetGP but not by HetGP, is correct and possibly even sharper than predicted by DetHetGP.

DetHetGP does not appear perfect from these plots however, as around $x_2 = 0.75$ there is a small bump in the mean. This bump does not seem sensible from the data, nor from prior understanding. Perhaps the deterministic approximation was

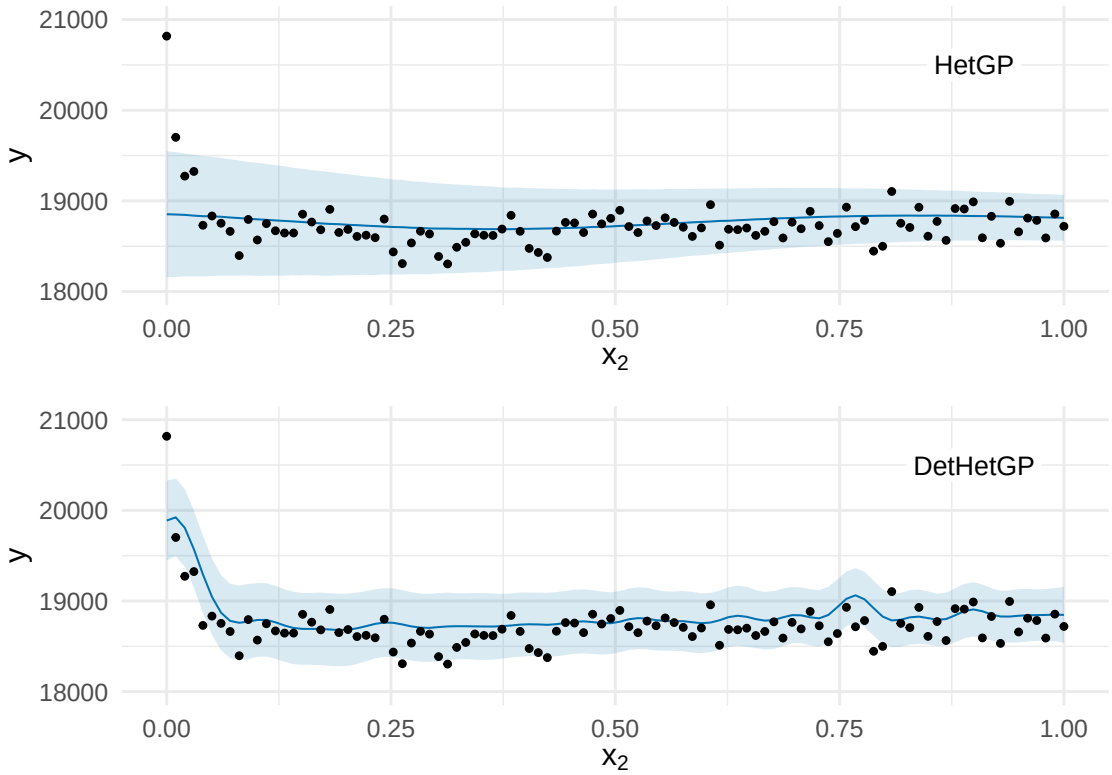


Figure 3.10: Emulator predictions for the second input of the building model (wall insulation thickness), when all other inputs are kept fixed at 0.5. HetGP predictions are plotted above, and DetHetGP predictions are below. Also superimposed are 100 additional simulator runs where the remaining 4 inputs were kept constant at 0.5.

modelled poorly due to a lack of deterministic data, or perhaps a limited amount of stochastic data caused this error (the following example suggests the former). Although not definitive, the evidence in general suggests that DetHetGP performs better than HetGP for this example. Including deterministic runs leads to a more accurately shaped mean function, and because the two emulators disagree with the mean function shape, it is probable that HetGP has estimated the mean function poorly.

EnergyPlus with a larger simulation budget

In the previous example, DetHetGP appears better than HetGP because it is able to capture (to some degree) a sharp increase in the mean for low values of x_2. Aside from this local feature however, EnergyPlus has a reasonably simple mean, which limits the value of DetHetGP. With a slightly larger simulation budget, HetGP can produce equivalent (or even improved) results to DetHetGP (as discussed more

generally in Section 3.3.4).

To see this, consider the previous example, but this time with an added 200 stochastic simulations (chosen via a maximin Latin hypercube). This results in a total simulation budget of 400; for HetGP this is 400 stochastic simulations, for DetHetGP this is 300 stochastic and 100 deterministic simulations. With the same scoring dataset as before, this results in a DetHetGP score of -5837, a DetHetGP $RMSE$ of 206, a HetGP score of -5812, and a HetGP $RMSE$ of 203. Here HetGP now seems to be better than DetHetGP overall, and also has a slightly better mean. This difference is minimal, and less of an advantage compared to the disadvantage in the smaller simulation budget example, but it is still present.

Figure 3.11 presents the same cross-section plots as before but with the larger training data sets.

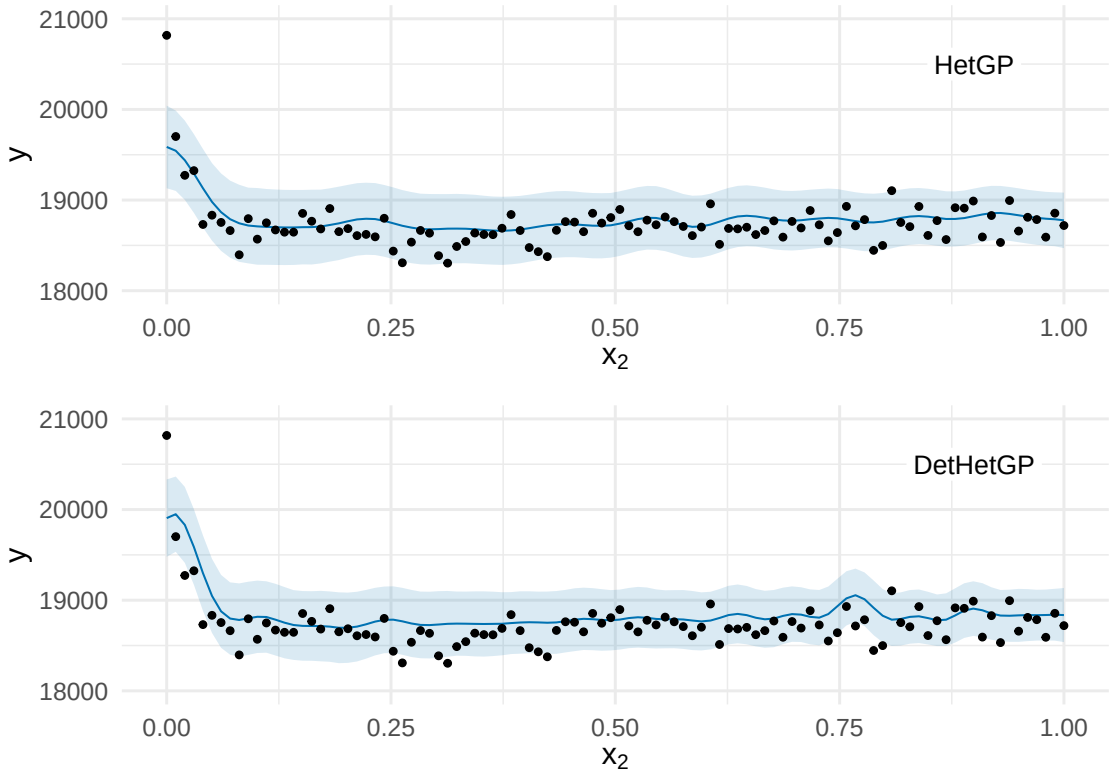


Figure 3.11: Emulator predictions for the second input of the building model (wall insulation thickness), when all other inputs are kept fixed at 0.5, using the larger simulation budget. HetGP predictions are plotted above, and DetHetGP predictions are below. Also superimposed are 100 additional simulator runs where the remaining 4 inputs were kept constant at 0.5.

Here we can see that the HetGP predictions are able to capture the uptick at

small x_2 values (at least as well as DetHetGP can), which was the main advantage DetHetGP had in the previous example. Without this mean flaw, the HetGP emulator now seems to be the preferred choice as the improved score values (and similar $RMSE$ values) suggest that the variance is better captured by only using stochastic simulations, and the mean is no worse off. Additionally, the slight bump in the mean around $x_2 = 0.75$ for DetHetGP is still present (which is not present for HetGP); the additional stochastic simulations have not been able to flatten it out. This example reaffirms the idea that DetHetGP is mainly valuable in low simulation budget examples. Because the building model's mean is relatively simple, what counts as a *large* simulation budget is actually quite reasonable, and EnergyPlus is also quite a fast simulator, which makes DetHetGP less valuable here.

3.5 ρ Parameter

One can also include a parameter, ρ, in the DetHetGP model which allows the y_D component to be turned off if need be (or its importance de-weighted). In other words: $y_H(\mathbf{x}) = \rho y_D(\mathbf{x}) + \delta(\mathbf{x})$. The motivation for including such a parameter is to mitigate the negative effects of a poor deterministic approximation (i.e. when Criteria 1 fails). In this section, estimation for the ρ parameter is done via maximum a posteriori estimation in Stan, along with all other parameters.

Using the same toy deterministic approximation as before (Section 3.2), but this time with more (80) stochastic points, and a stochastic simulator with a simpler mean (in Equation (3.8)) we obtain the predictions in Figure 3.12. On the left are predictions without the ρ parameter, and on the right are predictions including the ρ parameter. In these examples, ρ is given a $U(0, 1)$ prior.

$$\begin{gathered} y(x) = (1-x)\sin(\pi + 2\pi x) + \log(0.2 + x) + (1.2 - x)\epsilon; \\ \epsilon \sim N\big(0, 1\big). \end{gathered} \tag{3.8}$$

With a simpler mean, and more stochastic points, a deterministic approximation is not necessary to capture the overall structure of the simulator. However, because this deterministic approximation is deeply flawed (it is overly complex), the resulting DetHetGP predictions are also flawed (left in Figure 3.12); predicting a mean that is far more complex than the truth. This is another example of what can occur when Criteria 1 fails and there aren't enough stochastic runs to mitigate this. Including the

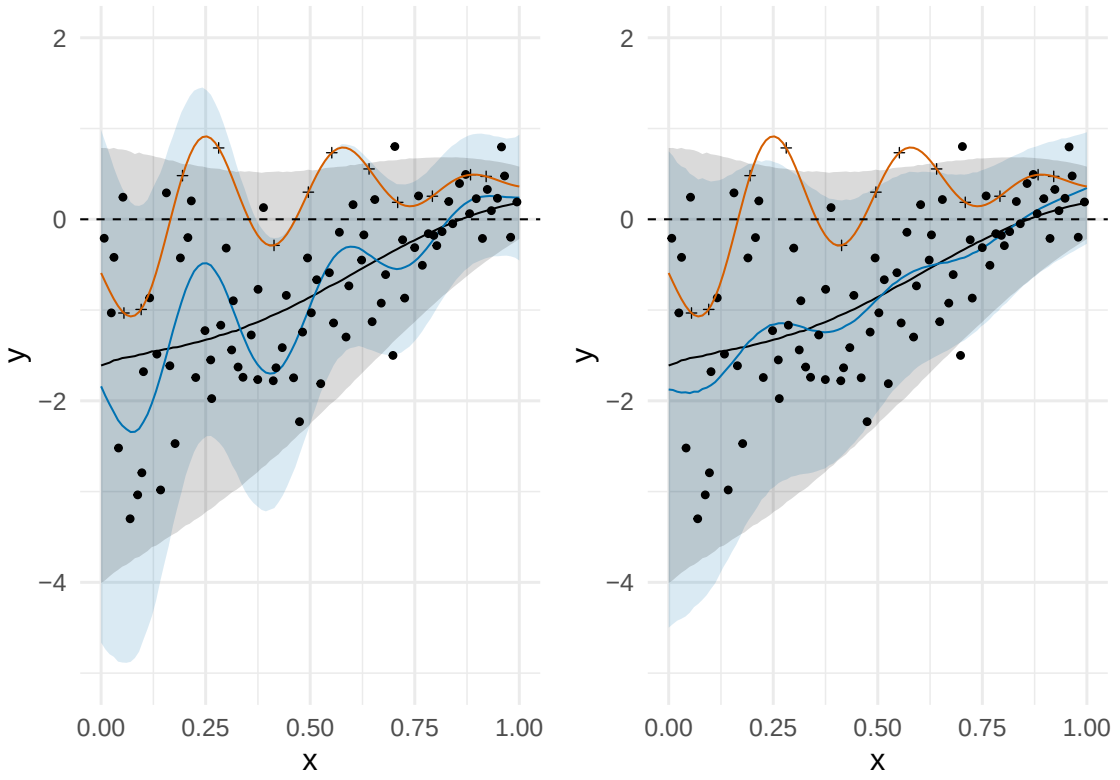


Figure 3.12: Emulator predictions for the simplified-mean toy simulator, using both stochastic and deterministic runs. The left plot does not include a ρ parameter, and the right plot does.

ρ parameter can de-weight the impact of the deterministic approximation, providing predictions which are again valid (right in Figure 3.12). These predictions still aren't completely free from the influence of the poor deterministic approximation ($\rho = 0.249$), but they are substantially better. In this way, including the ρ parameter can be a useful modification if the accuracy of the deterministic approximation is in doubt.

On the other hand, such a parameter can decrease the usefulness of a deterministic approximation, even when it is reasonably accurate. Returning to the example used to demonstrate Criteria 3 (where too few stochastic points were used, Section 3.3.3), we can see this in practice (Figure 3.13)

In this example, neither emulator is particularly accurate, but the one without the ρ parameter at least leverages the excessive number of deterministic runs, obtaining a mean that is somewhat similar to the truth. The emulator which includes a ρ parameter does not have enough confidence to make any use of the deterministic approximation ($\rho = 0.005$), and so the resulting predictions are worse. It is therefore

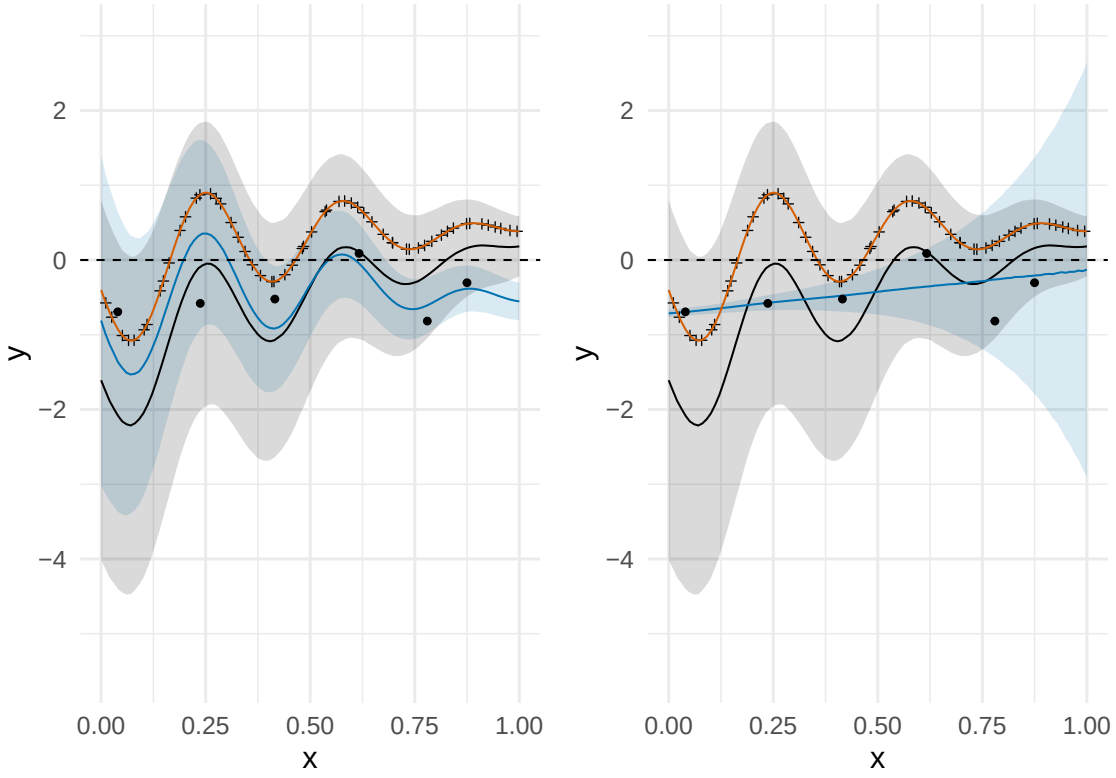


Figure 3.13: Emulator predictions for the toy simulator, using both stochastic and deterministic runs, with too few stochastic runs. The left plot does not include a ρ parameter, and the right plot does.

important that, if a ρ parameter is to be included, that a good prior is provided for it. Even without this parameter, DetHetGP does have the ability to adjust the overall mean predictions away from the deterministic approximation, and so a ρ parameter will mostly be useful in low data scenarios where including the deterministic approximation is a questionable strategy.

3.6 Analytical Formulation

The formulation in Section 3.2 requires sampling to enable prediction because the final predictions are conditional on the predictions from the y_D component. It also requires a prediction step in the fitting stage, which can make fitting the model slow. It is possible to re-write the model to enable analytical predictions (if predictions for the $\log(\sigma^2)$ process are also taken fixed at their expectation) and to provide a simpler formulation for fitting. This formulation is outlined below.

Taking $\mathbf{y}_{full} = (\mathbf{y}_D, \mathbf{y})^T$, and using the generative model as in Equation (3.2),

then:

$$\mathbf{y}_{full} \sim N\big(Mean(\mathbf{y}_{full}), Cov(\mathbf{y}_{full})\big); \tag{3.9}$$

with

$$Mean(\mathbf{y}_{full}) = \begin{pmatrix} Mean(\mathbf{y}_D) \\ Mean(\mathbf{y}) \end{pmatrix} = \begin{pmatrix} m_D(X_D) \\ m_D(X_D) + m_H(X_H) \end{pmatrix}, \tag{3.10}$$

and

$$\begin{aligned} \mathrm{Cov}(\mathbf{y}_{full}) &= \begin{pmatrix} \mathrm{Cov}(\mathbf{y}_D) & \mathrm{Cov}(\mathbf{y}_D, \mathbf{y}) \\ \mathrm{Cov}(\mathbf{y}, \mathbf{y}_D) & \mathrm{Cov}(\mathbf{y}) \end{pmatrix} \\ &= \begin{pmatrix} K_D(X_D, X_D) & K_D(X_D, X_H) \\ K_D(X_H, X_D) & K_D(X_H, X_H) + K_H(X_H, X_H) + \sigma^2(X_H)I \end{pmatrix}. \end{aligned} \tag{3.11}$$

These equations can be used to fit the model (whether by maximum a posteriori estimates as done here, maximum likelihood estimation, MCMC, or otherwise). Predictions, conditional on σ^2, are then:

$$\begin{aligned} y_H(X*) \mid \mathbf{y}_{full}, \sigma^2(X^*), \sigma^2(X_H) &\sim N\big(\mathcal{M}(X^*), \mathcal{V}(X^*)\big); \\ \mathcal{M}(X^*) &= m(X^*) + K(X^*, X_H)\big(Cov(\mathbf{y}_{full})\big)^{-1}\big(\mathbf{y}_{full} - m(X_H)\big), \\ \mathcal{V}(X^*) &= K(X^*, X^*) - K(X^*, X_H)\big(Cov(\mathbf{y}_{full})\big)^{-1}K(X_H, X^*), \end{aligned} \tag{3.12}$$

where

$$\begin{aligned} &m(X^*) = m_D(X^*) + m_H(X^*), \\ &K(X^*, X^*) = K_D(X^*, X^*) + K_H(X^*, X^*) + \sigma^2(X^*)I, \\ &m(X_H) = \big(m_D(X_D), m_D(X_H) + m_H(X_H)\big)^T, \\ &K(X^*, X_H) = \big(K_D(X^*, X_D), K_D(X^*, X_H) + K_H(X^*, X_H)\big), \\ &K(X_H, X^*) = K(X^*, X_H)^T. \end{aligned}$$

This is a similar formulation as used to link multiple models of varying fidelity (Kennedy and O'Hagan, 2000; Le Gratiet, 2013; Kennedy et al., 2020). Predictions using the sampling formulation and the analytical formulation are provided below. The analytical formulation can be much faster (no longer needing expensive Monte Carlo sampling for every prediction point[2], and also not needing a prediction step in

[2]By taking the $\log(\sigma^2)$ predictions fixed at their expectation, these predictions are only an approximation to the full predictions. Since point estimates are already used for $\log(\sigma^2(X_H))$, a full quantification of uncertainty for the $\log(\sigma^2)$ process is lost regardless. To avoid this approximation these equations could still be used, coupled with sampling from the $\log(\sigma^2(X^*))$ process, which should still be faster because it avoids the intermediate y_D sampling step required in Equation (3.3).

the fitting stage). This can be especially important for alternative objectives and extensions, where a fast, analytical, result can sometimes be essential (for example, closed form solutions facilitate Bayesian optimisation with a standard Gaussian process (Frazier, 2018)).

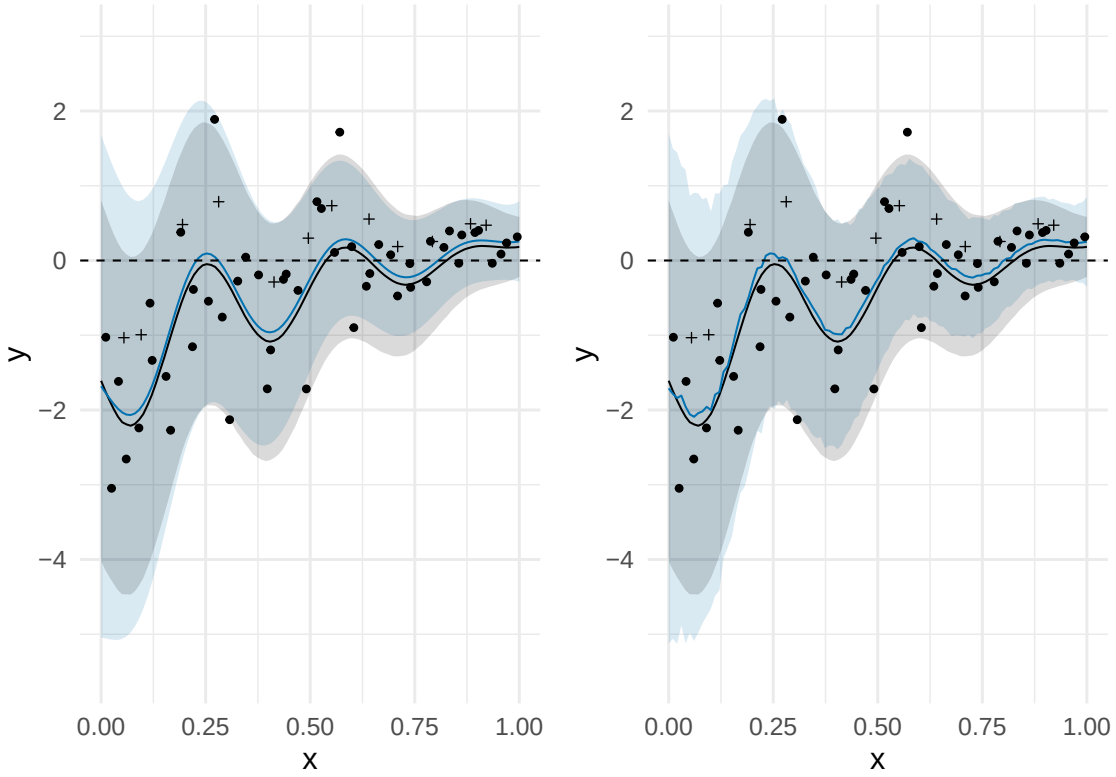


Figure 3.14: Emulator predictions for the toy simulator, using both stochastic and deterministic runs. The black interval and respective line are the true 95% interval and mean, and the blue interval and line are the emulator 95% predictive interval and mean. The stochastic data points are circles, and the deterministic data points are plus symbols. Left uses the analytical prediction formula, right uses sampling (with only 1000 samples).

We can see from the plots in Figure 3.14 that both provide the nearly identical results, but the one on the right provides noisier predictions due to residual sampling variation caused by not using enough Monte Carlo samples. In situations where enough samples can be drawn (for example, problems with small data sets), sampling is less of an issue. In general, however, the analytical formulation is better, and the analytical formulation should have been used throughout and in Baker et al. (2020a).

3.7 Discussion

In this chapter we have presented a method for including deterministic runs in the emulation of stochastic simulation experiments. By utilising a deterministic approximation of the stochastic simulator, a less noisy view into the general shape of the mean function can be learned. Including deterministic runs can produce better emulators, especially when the mean function is complex and sufficient prior knowledge of the mean function is lacking, and the simulation budget is insufficiently large.

A heteroscedastic Gaussian process is not the only method for modelling a stochastic simulator with a Gaussian process. The variance can instead be modelled with a simple parametric form, or taken as constant fixed variance, as in basic Gaussian process regression (Rasmussen and Williams, 2006). It might be interesting to see how useful DetHetGP is in these alternative scenarios. Similarly, other methods apart from standard Gaussian processes exist for flexibly modelling stochastic systems, and so it would be interesting to see if deterministic approximations can be incorporated into other such methods.

Phase-shifts and other more complex relationships between the deterministic approximation and the stochastic simulator are also unlikely to be properly captured by the described model formulation. Deep Gaussian processes have been investigated for dealing with this problem for multiple levels of code (Cutajar et al., 2019), and such a strategy could also be useful for incorporating deterministic approximations.

An interesting design question has been posed regarding how many deterministic points should be included: too few and the deterministic approximation will not be modelled well; too many and not enough stochastic data points will be generated, yielding a poor estimate of the variance and potentially the overall mean. The advice given in Section 3.3 provides a good starting point for this. Additionally, replicated simulations also provide an insight into the mean at any given point (and the variance). Deterministic simulations provide a perfect insight into an imperfect representation of the stochastic simulator, whereas replications provide an imperfect insight into the simulator itself. Which is more valuable likely depends on the degree of noisiness of the simulator and the accuracy of any given deterministic approximation, among other features. Using both deterministic simulations and replicated simulations together may provide an overall better solution, with the deterministic simulations

providing a good base emulator, and replications allowing improved adjustment and variance estimation. A thorough investigation into replicates and deterministic approximations remains future work.

In general however, whilst DetHetGP seems capable, interesting, and potentially worthy of further study; it is not particularly valuable for EnergyPlus. EnergyPlus is not particularly slow to run, which means that a relatively large simulation budget is often feasible. Additionally, EnergyPlus does not seem to contain particularly complex relationships (except the non-stationarity for very low values of wall insulation). Together, this makes a DetHetGP unnecessary in many cases for building simulation, and so the rest of this thesis returns to the building simulator and does not leverage DetHetGP. However, in other applications (such as epidemiology) where the computational cost of running a simulation is larger, we think that DetHetGP, and the idea more generally, could be extremely valuable.

Chapter 4

Finding Acceptable Building Designs[1]

4.1 Introduction

In previous chapters the building model served more as an illustrative example for different stochastic emulation problems. In this chapter, the building model problem takes centre stage, as we investigate a case study example where stochastic emulation techniques can be practically applied. Whilst new methodology is developed, the focus in this chapter is primarily on the application.

Our goal is to find modifications to an existing building design which will provide satisfactory overheating risks *and* energy demands; even after the expected increase in temperature by the end of the 21st century. Due to the non-trivial time required to run EnergyPlus, and the stochastic nature of (our version of) EnergyPlus, emulators are essential for comprehensive analysis.

In this chapter we aim particularly for this notion of 'satisfactory' rather than 'optimal'. The main reasoning for this is that, in practice, there can often be additional criteria beyond numerical summaries which can be important when it comes to building design (for example, a building's appearance). Additionally, it would be far easier, and more sensible, for potential regulations to require specific threshold standards rather than some relative notion of 'most-improved'.

We refer to our goal as level set estimation. The 'level set' typically refers to the set of inputs where the output is exactly at some threshold, but we use the

[1]Much of this chapter's contents have been published in Baker et al. (2021)

term more liberally to refer to when the output is less than (or greater than) some threshold. Finding the former typically reveals the latter (and vice-versa), and so this abuse of definition is reasonably innocent. A more accurate term for our goal would be "sublevel" set estimation (or "superlevel" set estimation if we targeted values larger than some threshold) or "excursion" set estimation.

To efficiently achieve our goal, we modify the history matching procedure (Craig et al., 1997; Vernon et al., 2014; Andrianakis et al., 2015). In general, history matching provides straightforward general implementation, easy utilisation even when the input and output dimension is high, a robustness to low simulation budgets, and the ability to identify when no such 'real' input exists. History matching techniques have also already been extended to the problem of optimisation (Lawson et al., 2016), and so it is a natural step to extend them to level set estimation as well.

4.2 Building Model

For this case study, we use the building outlined in Figure 4.1. However, even though this chapter gives more attention to a specific building than previous examples, it is still an illustrative example, and the outlined procedure should be useful for other buildings in practice.

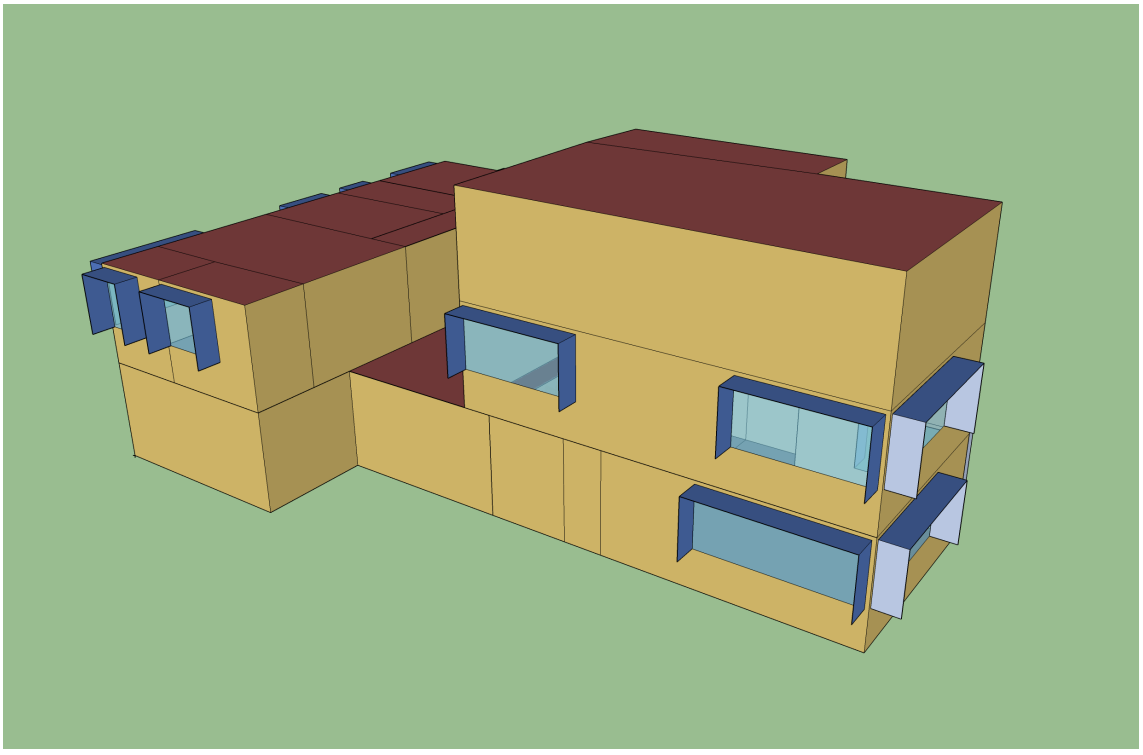


Figure 4.1: The geometry of the modelled building.

Our building is fitted with an "ideal loads" heating system, and no air conditioning. No air conditioning may seem like an odd setup, given the intended objective, but this choice is more representative of the UK's building stock, where it has been reported that only 0.5% of residential buildings have air conditioning (BBC, 2013). Air conditioning could have been included, with a user-input capacity, without any major methodological modifications. A more realistic heating system could also have been used.

For different building designs, different attributes can be adjusted. For this case study we assume the following attributes are adjustable: the wall insulation thickness (considering values from 0m to 0.5m), the roof insulation thickness (from 0m to 0.5m), the ground insulation (from 0m to 0.1m), the size of the windows (from 20% to 100% of the wall size, ensuring at least a 0.1m windowless border), the length of window overhangs (from 0% to 100% the height of the windows), the amount the windows can be opened by occupants (from 0% to 100% of the windows size), the emissivity of the roof (from 0.4 to 1), and whether or not the windows are double or triple glazed. From now on, these are referred to as $x_1, \ldots, x_8$ and their input ranges are rescaled to be between 0 and 1. Other inputs could have been considered (for example, the air conditioning capacity) - the specific choices in practice are down to what construction options are available.

For this example we want to use a more realistic weather generator than in Chapters 2 and 3. Additionally, our interest lies in 'future-proofing' the building, i.e. making sure it performs well towards the end of the 21st century. With these desires in mind, our choice of weather generator is the UKCP09 weather generator, which can output possible samples of weather for the year 2080, which can then be input into EnergyPlus (Eames et al., 2011). UKCP09 is relatively old, and therefore its internal climate predictions are fairly out of date. The more recent UKCP18 projections have since been developed (Lowe et al., 2018), although at the time of writing no weather generator has been produced for it. Other weather generators can be used here, with the specific conclusions differing slightly depending on the form of the chosen weather generator.

EnergyPlus directly outputs the yearly heating energy usage, which can then be used to improve the building design. For the other objective (overheating), EnergyPlus outputs the indoor temperature throughout the year. The indoor temperature can then be used to classify whether or not the building overheated

during the simulation. We follow the definition of overheating as provided by the Chartered Institution of Building Services Engineers (CIBSE, TM52 (2013)). Other classifications or metrics for overheating could have been considered instead.

For their definition, the weighted running mean T_{rm} is first calculated for every day as $T_{rm} = 0.2T_{od-1} + 0.8T_{rm-1}$, where T_{od-1} is the outdoor temperature from the previous day and T_{rm-1} is the weighted running mean from the previous day. The maximum acceptable indoor temperature T_{max} is then defined as $0.33T_{rm} + 21.8°C$. This maximum acceptable temperature is then used to assess the relative temperature inside the building; ΔT is an hourly time series of the difference between the "operative indoor temperature" T_{op} and the maximum comfort temperature T_{max}. ΔT then represents a value for how hot the building is at any one hour. ΔT can be calculated from the outputs from EnergyPlus, as EnergyPlus outputs the operative temperature and the outdoor temperature is a required input. Three criteria are then constructed from ΔT to quantify the different ways a building can overheat. Criteria 1 checks how often the building is too hot. This criteria is broken if more than 3% of occupied hours between May and September have a ΔT value over 1 degree. Criteria 2 is a combination of the length of time a building is too hot, and how hot it is. This criteria is broken if, for any given day, ΔT sums to be greater than or equal to 6 (only counting occupied hours). Criteria 3 checks how hot a building gets at its peak. This criteria is broken if, for any occupied hour, ΔT is greater than 4. A building is then considered to overheat if 2 or more of the criteria are broken.

Because our modified EnergyPlus is stochastic, a building may not overheat, or have low heating costs, only because the weather was favourable in that simulation. As such, we are interested in the *probability* of a given building design overheating and the *average* heating energy usage of a given building design.

In this work, we decide to aim for a less than 1% probability of overheating. This represents a 'sufficiently small' value, but a different value could have been chosen. For the average heating energy usage, we aim for less than 15kWh/m^2, which is the requirement set by the Passivhaus standard (Schnieders and Hermelink, 2006), but a different threshold could also have been chosen.

To summarise, our goal is to find the set $\mathcal{L}$, such that:

$$\mathcal{L} = \{(x_1, \ldots, x_8) : P\big(y_{oh}(x_1, \ldots, x_8)\big) < 0.01, E\big(y_{eu}(x_1, \ldots, x_8)\big) < 15\}, \quad (4.1)$$

where $(x_1, \ldots, x_8)$ are the different design choices, y_{oh} is the binary output that classifies whether the building overheats and y_{eu} is the building's energy usage.

4.3 Emulators

Two emulators are needed for this problem, one for energy usage and one for overheating. Energy usage is a continuous variable and thus can be modelled in a similar fashion as in the previous chapters. Overheating is binary, and therefore needs to be modelled differently.

Additionally, the existence of the binary input variable x_8 is a mild complication. This is non-standard in the emulation community, with a standard Gaussian process formulation requiring all inputs be continuous. The window glazing variable was selected partially to show that binary input variables can still be included, as binary variables are likely to be common as potentially adjustable attributes in a building design. In this work, we use the mechanism outlined in Qian et al. (2008) that allows non-continuous variables to be included in the covariance structure of a Gaussian process. A simpler (but less efficient) alternative would have been to fit independent emulators for each of the different window glazing options.

We first model the overheating risk using a Gaussian process classifier (Rasmussen and Williams, 2006). This models overheating as a Bernoulli variable with a latent Gaussian process for the logit probability of overheating ($\text{logit}(p)$). Taking the continuous inputs as $\mathbf{x}_c = x_1, \ldots, x_7$, and the binary input(s) as $\mathbf{x}_b = x_8$, we have the following overheating risk emulator:

$$\begin{aligned} y_{oh}(\mathbf{x}_c, \mathbf{x}_b) &\sim \text{Bernoulli}\big(p(\mathbf{x}_c, \mathbf{x}_b)\big); \\ \text{logit}(p(\mathbf{x}_c, \mathbf{x}_b)) &\sim GP\big(m_{oh}(\mathbf{x}_c, \mathbf{x}_b), K_{oh}(\mathbf{x}_c, \mathbf{x}_b, \mathbf{x}'_c, \mathbf{x}'_b)\big). \end{aligned} \tag{4.2}$$

We use a zero-mean function for $m_{oh}(\mathbf{x}_c, \mathbf{x}_b)$, letting the covariance function $K_{oh}(\mathbf{x}_c, \mathbf{x}_b, \mathbf{x}'_c, \mathbf{x}'_b)$ do all of the work. The covariance function is taken as:

$$K_{oh}(\mathbf{x}_c, \mathbf{x}_b, \mathbf{x}'_c, \mathbf{x}'_b) = \alpha_{oh}^2 exp\left(- \sum_{x \in \mathbf{x}_c} \frac{(x_i - x'_i)^2}{l_{oh_i}^2} - \sum_{x \in \mathbf{x}_b} \phi_{oh_i} \mathbb{1}(x_i \neq x'_i) \right), \tag{4.3}$$

where α_{oh}^2 is the overall variance of the process; the left sum controls the correlation between two data points if only the continuous variables vary, with the lengthscales l_{oh_i} modelling the smoothness of the relationship in the i^{th} continuous dimension (this is the squared exponential correlation function); and the right sum controls

the correlation between two data points if only the binary variables vary ($\mathbb{1}$ is the indicator function), with ϕ_{oh_i} modelling this correlation for the i^{th} binary variable. In our case, we only have one binary variable, so the right sum can be replaced with a single term, but the full summation is provided here for generality.

For hyperparameter priors, the overall variance is given a Half-Normal$(0, 1)$ prior, the lengthscales are given Inverse-Gamma$(5, 5)$ priors and the binary correlation parameter is also given a Half-Normal$(0, 1)$ prior. These priors were chosen to broadly cover the range of reasonable values, with the overall variance and lengthscale priors following those advised by the Stan user guide (Stan Development Team, 2016).

Fitting this model is done via variational inference using GPflow (Matthews et al., 2017), providing uncertainty distributions for the latent probability estimates.

The second emulator is the one for energy usage. We model this with a heteroscedastic Gaussian process as in previous chapters. Here, point estimates are used for the hyperparameters, and the predictive intrinsic variance is taken fixed at its expectation. Using the same notation as Equation (4.2) we have the following energy usage emulator:

$$\begin{aligned} y_{eu}(\mathbf{x}_c, \mathbf{x}_b) &\sim GP\big(m_{eu}(\mathbf{x}_c, \mathbf{x}_b), K_{eu}(\mathbf{x}_c, \mathbf{x}_b, \mathbf{x}_c', \mathbf{x}_b') + \sigma^2(\mathbf{x}_c, \mathbf{x}_b)\big); \\ \log(\sigma^2(\mathbf{x}_c, \mathbf{x}_b)) &\sim GP\big(m_\sigma(\mathbf{x}_c, \mathbf{x}_b), K_\sigma(\mathbf{x}_c, \mathbf{x}_b, \mathbf{x}_c', \mathbf{x}_b')\big), \end{aligned} \tag{4.4}$$

where σ^2 represents the intrinsic variability of the energy usage output. The mean functions ($m_{eu}(\mathbf{x}_c, \mathbf{x}_b)$ and $m_\sigma(\mathbf{x}_c, \mathbf{x}_b)$) and covariance functions ($K_{eu}(\mathbf{x}_c, \mathbf{x}_b, \mathbf{x}_c', \mathbf{x}_b')$ and $K_\sigma(\mathbf{x}_c, \mathbf{x}_b, \mathbf{x}_c', \mathbf{x}_b')$) are given the same structure and priors as the overheating emulator. This model is fit via maximum a posteriori estimation using Stan (Stan Development Team, 2016), but the Gaussian process structure ensures that a full probability distribution is still obtained for the mean (which is our quantity of interest)[2]. The hetGP R package could have been used here (Binois and Gramacy, 2017), as technically the non-standard covariance function used in Equation (4.3) can be represented as a standard squared exponential correlation function, as the $\mathbb{1}(x_i \neq x_i')$ term in the binary component can be represented as $(x_i - x_i')^2$, which is exactly the numerator of the squared exponential correlation function (and so the binary correlation parameter ϕ would be equal to $1/l^2$).

[2]GPflow has a capable implementation of variational inference for GP classifiers, whilst Stan does not. On the other hand, implementing hetGP in Stan was easier than in GPflow. This is the reason why different inference methods were used for each emulator, but in both emulators a full probability distribution is obtained for the quantity of interest.

Together, these two emulators provide a way of predicting what the overheating risk and the average energy usage are for any values of $(\mathbf{x}_c, \mathbf{x}_b)$. These predictions will have an uncertainty distribution around them, which can be easily obtained from the Gaussian process predictive equations. The accuracy and precision of these predictions will depend on the total number of simulations used to fit these models.

At this stage, we could use these emulators directly to estimate the level set. However, if the level set makes up a relatively small part of the input space, the precision of the emulator predictions in the region of interest will be low. This is because the number of nearby simulations will be low, as the number of simulations in the region of interest will be quite sparse. This is where a history matching strategy can be useful.

The next section will outline the history matching inspired level set estimation methodology - detailing how one can better estimate the level set of a simulator. The section after that will then apply said methodology to the above emulators - finding suitable buildings with regard to overheating risk and average energy usage. One key benefit of the proposed methodology, and history matching more generally, is that it is easily generalisable to many types of emulator - as long as a mean and variance of a prediction can be obtained, then the proposed methodology can be directly used.

4.4 History Matching Level Set Estimation

We adapt the history matching procedure to apply to level set estimation, rather than calibration. More information about standard history matching can be found in Craig et al. (1997); Vernon et al. (2014); and Andrianakis et al. (2015). The general idea here is to build initial emulators for our quantities of interest, and use these to rule-out buildings which almost certainly do not meet the desired criteria. The emulators can then be improved by obtaining more simulations for building designs which have not been ruled-out, providing a better estimate for what the level set is. This process can be repeated, each time improving the results. Each set of simulations and analysis is typically referred to as a 'wave'.

Taking y as the quantity of interest (overheating risk or average energy usage), $\mathbf{x}$ a set of inputs $((x_1, \ldots, x_8))$ and T the threshold (0.01 or 15), we define the

'implausibility' as follows:

$$I(\mathbf{x}) = \frac{E(y(\mathbf{x})) - T}{\sqrt{V(y(\mathbf{x}))}}, \tag{4.5}$$

where $E(y(\mathbf{x}))$ is the expectation of the emulator, and $V(y(\mathbf{x}))$ is its variance. In standard history matching, the $V(y(\mathbf{x}))$ term is usually supplemented with an added V_{obs} term to represent the observational error (which is arguably not present around the threshold T) and an added V_{MD} to represent the simulator error (which was not included here, but discussed more in Section 4.8). This implausibility can be positive or negative, which is a key difference to standard history matching. Large, positive, values of $I(\mathbf{x})$ suggest the input setting is not in the level set, as the expectation is much larger than the threshold T. Large, negative, values suggest that it is in the level set, as the expectation is much smaller than T.

A value of 3 or greater for $I(\mathbf{x})$ is taken as the threshold for a value being 'implausible'. Three is the value often used in standard history matching, based on the Pukelsheim's three sigma rule (Pukelsheim, 1994). Any value of $\mathbf{x}$ with $I(\mathbf{x}) > 3$ is ruled-out, and no longer needs to be considered. The set of $\mathbf{x}$ values that are not ruled-out yet is often referred to as the 'NROY' space (the Not Ruled-Out Yet space). Similarly, we 'rule-in' any value of $\mathbf{x}$ where $I(\mathbf{x}) < -3$; these are so likely part of the level set that they are not worth wasting further simulations on, and thus also no longer need to be considered (but should be remembered). The set of $\mathbf{x}$ values which are not ruled-in yet will here on be referred to as the 'NRIY' space (the Not Ruled-In Yet space), which is not present in standard history matching.

For clarification, consider the image in Figure 4.2.

This illustration demonstrates how 4 distinct regions of space emerge from using the implausibility metric from Equation (4.5). The central line, going from the top left corner to the bottom right corner, represents the set of inputs where the output exactly equals T. The black, top right, region represents the set of inputs where the output is much larger than T; these are almost certainly not in the level set, and thus are ruled-out. The red, bottom left, region represents the set of inputs where the output is much smaller than T; these are therefore almost certainly in the level set, to the extent that they become less interesting, and thus are ruled-in. The uncoloured middle regions are the regions of greater interest. The upper uncoloured region, NROY, represents the set of inputs where the implausibility is greater than 0, and thus are not believed to be in the level set; but the implausibility is not large

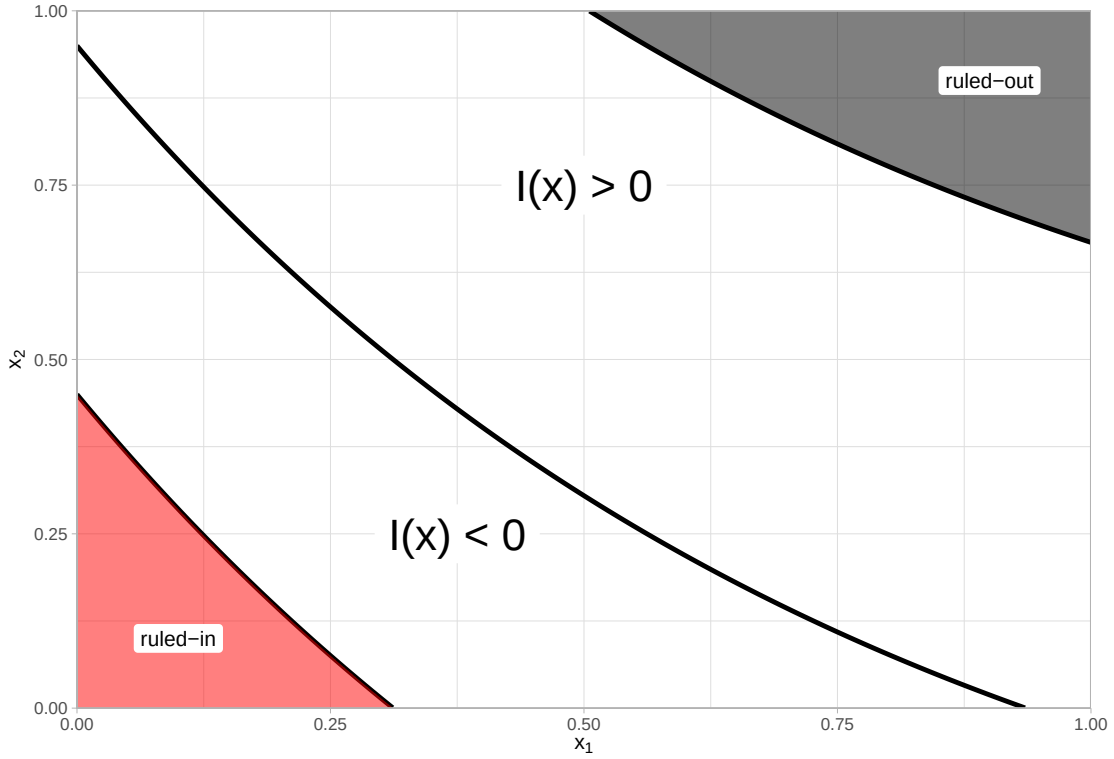


Figure 4.2: An illustration of the 4 regions that emerge from the history matching level set estimation technique.

enough to know for certain. On the other hand, the lower uncoloured region, NRIY, represents the set of inputs where the implausibility is smaller than 0, and thus are believed to be in the level set; but the implausibility is not small enough to know this for certain either. Note that buildings can move between NROY and NRIY between waves.

With this, we then have a set of $\mathbf{x}$ values which are candidates for future simulations (any values where $-3 < I(\mathbf{x}) < 3$). Running simulations for some of these buildings which are both NROY and NRIY (from here on referred to as the NROY/NRIY space) and refitting the emulator will improve the emulator in this space. This space is the region closest to the boundary, and additional simulations here should help to distinguish which side of the boundary specific buildings are. This process can then be repeated several times, until the space of NROY/NRIY is acceptably small, it does not appear to change, or the simulation budget is exhausted.

If at any point, all choices of $\mathbf{x}$ are ruled-out (i.e. all values of $I(\mathbf{x})$ are greater than three), then this implies that no values of $\mathbf{x}$ are in the level set.

If more than one output is being emulated (as in our problem, where we have two

quantities of interest), one can take the overall implausibility to be the maximum of the individual implausibilities - if it is implausible that a specific building meets one of the individual criteria, then that building is considered implausible overall.

In the final wave, when a final decision must be made (or a set of final candidate values must be presented to a practitioner), it is not reasonable to allow the choice of any 'non-implausible' values if they still have fairly large implausibility, but not quite as large as three. Such buildings are still predicted by the emulators to fail the criteria, but not with enough certainty to rule them out. Therefore, in the final wave, we can impose stricter requirements, such as any input values where the implausibility is less than 0 (which we call the 'tenable' set), or inputs where the implausibility is less than -1. A more desirable choice would be to only consider values with an implausibility less than - 3 (i.e. those ruled-in), with our simulation budget however we find this to be too strong a requirement as almost none of our candidate buildings end up ruled-in.

This methodology is easy to implement, and is conceptually understandable - we rule-out buildings that are obviously not in the level set, rule-in those that are obviously in the level set, and all others can be investigated further.

In the next section, we shall apply this methodology to the two building criteria described previously.

4.5 Results

We start by considering (but not simulating) a set of 1000000 candidate buildings (these are chosen by combining two random Latin hypercubes of size 500000 (McKay et al., 2000), one for each value of the binary input variable). This reduces the infinite number of possible buildings down to a finite (but very large) number to consider. Our goal is to then reduce this huge number of candidate buildings to a more manageable subset of 'future-proofed' buildings. For clarity, the different inputs and their initial considered ranges are presented again in Table 4.1

Initially, in the first wave, we fit the two emulators using an initial set of 500 simulations: 250 unique $\mathbf{x}$ input points chosen by a sliced Latin hypercube design (Ba et al., 2015), each replicated once. We then calculate $I_{oh}(\mathbf{x})$ and $I_{eu}(\mathbf{x})$ (that is, the implausibilities for the overheating risk and the energy usage) for each of the initialised candidate buildings.

Variable	Variable name	Range
x1	Wall Insulation Thickness	0m - 0.5m
x2	Roof Insulation Thickness	0m - 0.5m
x3	Ground Insulation Thickness	0m - 0.1m
x4	Window Size	20% - 100%
x5	Window Overhang Length	0% - 100%
x6	Window Opening Amount	0% - 100%
x7	Roof Emissivity	0.4 - 1
x8	Glazing Type	Double or Triple

Table 4.1: Table showing the inputs and their ranges considered for the building model.

As a reminder, we are interested in the values of the overheating *risk* and the *average* energy usage, not the raw outputs of the simulator. Therefore, in calculating $I_{oh}(\mathbf{x})$ via Equation (4.5), $y(\mathbf{x})$ is replaced with the logit overheating risk and $y(\mathbf{x})$ is replaced with the mean energy usage in the calculation of $I_{eu}(\mathbf{x})$. Because the logit overheating risk is used, rather than the overheating risk itself, we also modify the value of L, the target threshold, to be logit(0.01) rather than just 0.01. The logit overheating risk is the original output of the latent Gaussian process emulator, and is also unbounded. Using the overheating risk itself could also be done, although such a quantity is bounded between 0 and 1.

With these two sets of implausibilities, $I_{oh}(\mathbf{x})$ and $I_{eu}(\mathbf{x})$, we take the overall implausibility as $I(\mathbf{x}) = max(I_{oh}(\mathbf{x}), I_{eu}(\mathbf{x}))$. Any candidate building $\mathbf{x}$ where the overall implausibility is greater than 3 can then be ruled-out (and any less than -3 can be ruled-in).

4.5.1 Wave 1

In wave 1, 14.47% of the space was found with $I(\mathbf{x}) < 3$ (i.e. 85.53% of all the initial candidate buildings were immediately ruled-out / only 14.47% were left not ruled-out) and 0.10% of the space was found with $I(\mathbf{x}) < 0$ (i.e. a very small 0.10% of the initial candidate buildings were found to be tenably future-proof). This means that, without further simulation, almost no building designs are believed to satisfy both criteria: 0.10% of all buildings are estimated to satisfy both criteria using the

emulators' mean predictions. This is a very small number, providing very little choice for a practitioner (and as we will see later, the confidence in these designs satisfying both criteria may be quite low).

To visualise the types of buildings which are currently most future-proof, we make use of standard (in the history matching literature) minimum implausibility and optical depth plots (Andrianakis et al., 2017). In these, for every combination of two input variables, a 2D grid is made. Every candidate building is then sorted into the relevant grid cell for the 2D combination of input variables. Minimum implausibility plots present the minimum implausibility of any building within each grid cell, and the optical depth plots plot the proportion of buildings that are not ruled-out within each grid cell. Small minimum implausibilities or large optical depths represent good building design choices. These plots then provide information about the shape of the ruled-out space and the implausibility in this 2D projection. This can then be repeated for all 2D input variable combinations. These plots provide insight into what types of buildings are acceptable, and key relationships can be identified. We also present the 1D histograms showing the proportion of buildings which are not ruled-out for each input. These make it more clear what the individual impact of each input is, but it is important to remember that the overall shape of NROY/NRIY is a complicated 8D surface with potentially many complex interactions.

These plots are initially reasonably overwhelming, but they do contain a great deal of useful information. Along the diagonals we can see the individual impact of each input. We can observe that small values for x_3 and large values of x_4 typically result in poor buildings. Large values for x_1, small values for x_2, small values for x_5, large values for x_7, and $x_8 = 0$ are also poor choices.

On the off diagonals we can see key 2D relationships. For example, the 2 plots corresponding to the $x_4 : x_5$ interaction show how the negative effect of choosing a larger value for x_4 can be mitigated by choosing a larger value for x_5. This is not surprising given that x_4 corresponds to the size of the windows, and x_5 corresponds to the size of the windows overhang (intuitively, larger windows are likely to increase the chance of overheating, and larger overhangs will decrease this chance). Other relationships are also visible. For example, it seems that small values of x_3 and small values x_5 are particularly bad, but these don't appear to interact with each other - a large value for x_5 does not appear to allow for a small value of x_3.

Another key observation from these wave 1 plots, is that the minimum implausi-

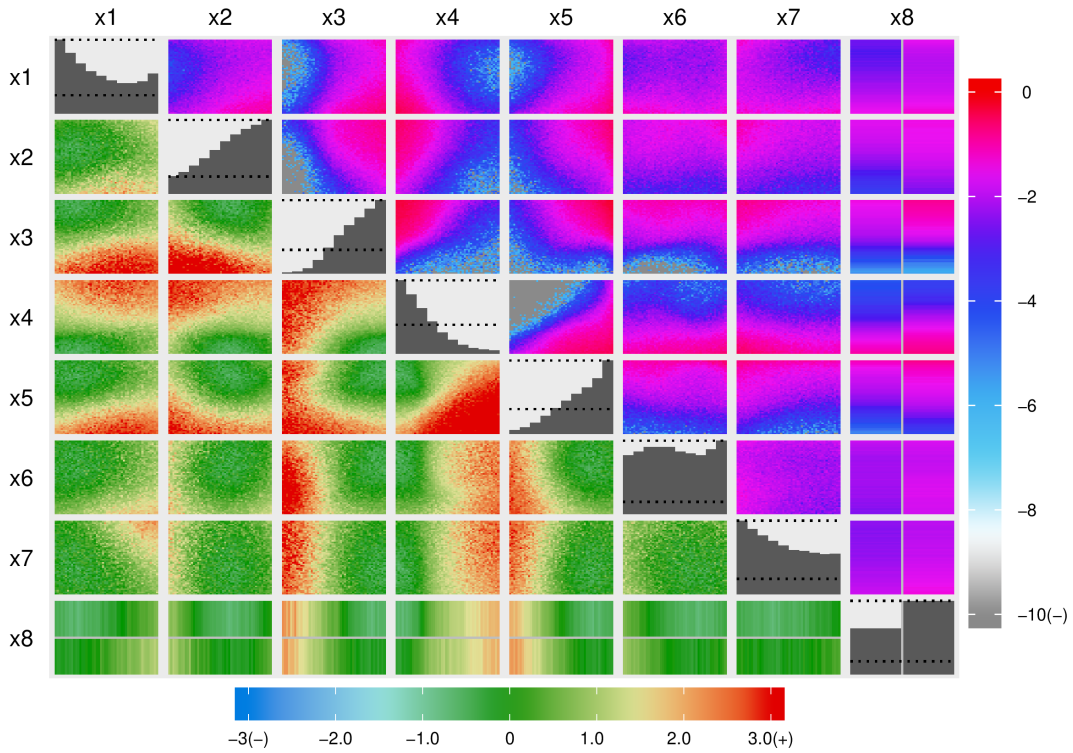


Figure 4.3: Minimum implausibility plots (below and left of diagonal) and optical depth plots (above and right of diagonal) for Wave 1. For the minimum implausibility, the scale is capped above by 3 (above this all buildings are ruled-out), and below by -3 (below this all buildings are ruled-in). For the optical depth, the scale is on the log scale, and goes from 0 (i.e. all buildings are non-implausible) down to -10 or lower (i.e. less than 0.0045% of buildings are non-implausible). The diagonal histograms show the *relative* proportion of buildings which are not ruled-out; also shown are two horizontal lines indicating how much the histograms must be scaled down to instead present the *absolute* proportion of buildings which are not ruled-out.

bility is never blue (i.e. the minimum implausibility is never much smaller than 0). We already knew this, but later waves will show a stark contrast to this initial plot.

4.5.2 Wave 2

With a large degree of emulator uncertainty (14.47% of designs still NROY/NRIY), it is reasonable to believe that, with more simulations, we could discover more viable designs.

For the next wave's simulation data, a random selection of 250 buildings are chosen from the larger candidate set, only considering those which are still NROY/NRIY.

Each of these chosen buildings is then simulated twice. This simulated data set, along with any of the previously simulated data which is still not ruled-out, can be used to re-fit the emulators. By homing in on more relevant building designs, we can greatly improve the accuracy of the emulators in the regions of input space which are most interesting. The newer emulators can then be used to further rule-out (or rule-in) candidate buildings. A helpful computational attribute here is that once a building is ruled-out (or ruled-in), it no longer needs to be checked - it has already been ruled-out (or ruled-in), its final implausibility value is the last one it was assigned.

For Wave 2, the not ruled-out space was shrunk further down to 7.44% of the total space and 1.49% of the space was tenable. Although not as large a shrinking of input space as the first wave, wherein clearly terrible buildings could be ruled-out; this wave still rules-out a large number of buildings.

Visual results for this wave are shown in Figure 4.4

As mentioned earlier, the most striking difference here is that a large part of the input space is now light blue for the minimum implausibility plots. This shows how, at this stage, many buildings are now best estimated as being tenably future-proofed. With additional simulation in the region of interest, we begin to discover viable building designs. The second most obvious observation is that the ruled-out space has grown - with the red regions in the minimum implausibility plots having spread, and the grey regions in the optical depth plots also having grown.

4.5.3 Wave 3

Repeating again for a third wave, exactly as before: 6.32% is not ruled-out and 1.99% of all buildings are tenably future-proof. This is a minor change compared to the previous improvement, and so we stop here. Eventually, the results would converge to the point where all buildings are either ruled-out or ruled-in. However, because every simulation is costly, this is not practical. At this point, it appears that additional simulations are providing diminishing returns, and so additional simulation is more difficult to defend.

Figure 4.5 presents the visual results for wave 3.

Whilst the differences here are smaller, there are observable differences between the wave 3 and wave 2 results - the blue regions are larger and darker, the red regions are larger, and the boundary between them is more stark. With additional waves,

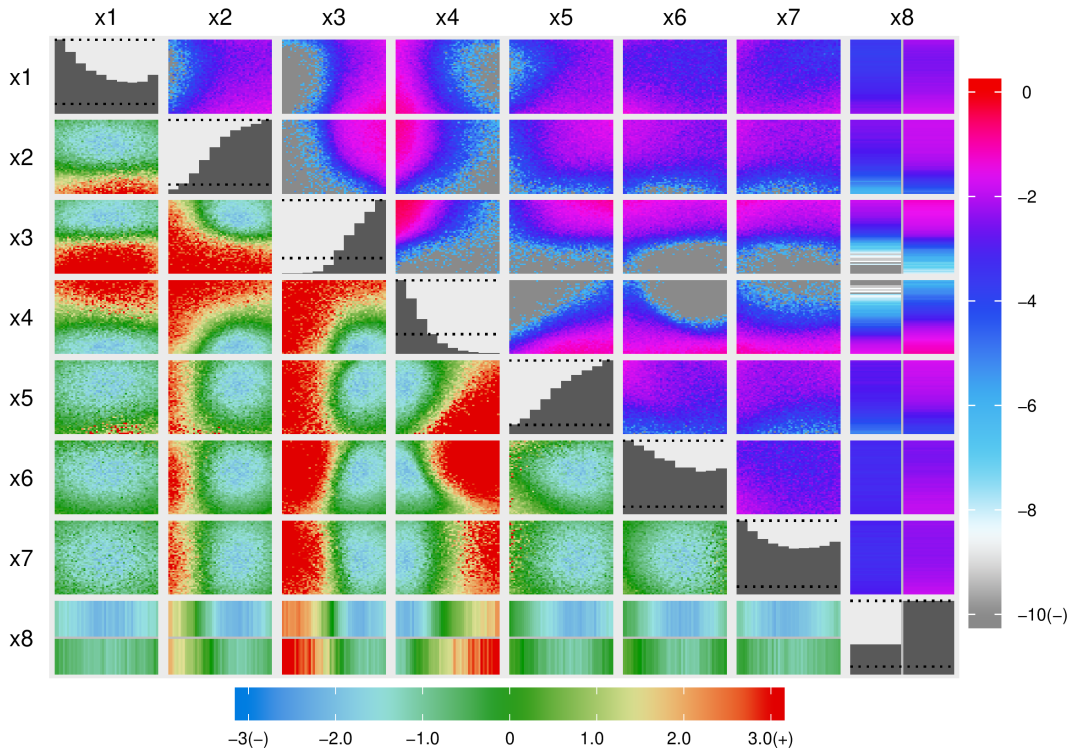


Figure 4.4: Minimum implausibility plots (below and left of diagonal) and optical depth plots (above and right of diagonal) for Wave 2. For the minimum implausibility, the scale is capped above by 3 (above this all buildings are ruled-out), and below by -3 (below this all buildings are ruled-in). For the optical depth, the scale is on the log scale, and goes from 0 (i.e. all buildings are non-implausible) down to -10 or lower (i.e. less than 0.0045% of buildings are non-implausible). The diagonal histograms show the *relative* proportion of buildings which are not ruled-out; also shown are two horizontal lines indicating how much the histograms must be scaled down to instead present the *absolute* proportion of buildings which are not ruled-out.

we could expect this pattern to continue.

From this point, we leave a final choice of building design down to a practitioner because secondary (or in this case, tertiary) criteria often exist. Preferably, the practitioner would choose only from ruled-in designs, but this can be an overly strict criteria and barely any ruled-in buildings have been found (0.0019% of buildings have $I(\mathbf{x}) < -3$) by the end of the third wave). Additional waves with further simulation would rectify this, but such a strict requirement is not always necessary, nor would it always be feasible.

A bare-minimum requirement for selecting a final building might involve consid-

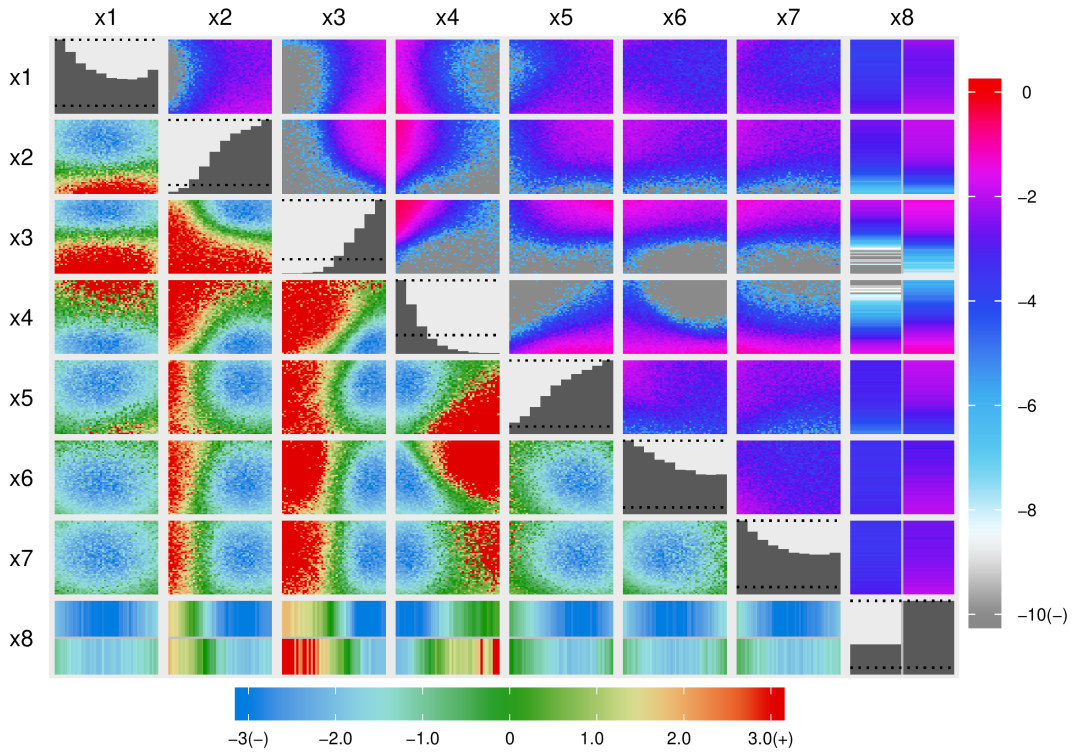


Figure 4.5: Minimum implausibility plots (below and left of diagonal) and optical depth plots (above and right of diagonal) for Wave 3. For the minimum implausibility, the scale is capped above by 3 (above this all buildings are ruled-out), and below by -3 (below this all buildings are ruled-in). For the optical depth, the scale is on the log scale, and goes from 0 (i.e. all buildings are non-implausible) down to -10 or lower (i.e. less than 0.0045% of buildings are non-implausible). The diagonal histograms show the *relative* proportion of buildings which are not ruled-out. Also shown are two horizontal lines indicating how much the histograms should be scaled down to instead present the *absolute* proportion of buildings which are not ruled-out.

ering any buildings with $I(\mathbf{x}) < 0$. These are all the candidate buildings where the emulator mean predictions (i.e. the best guesses) suggest each requirement is satisfied. Using this criteria, and selecting the allowed design with the largest windows, leads to a final building which could be built. For this building, the wall insulation is 0.375m thick, the roof insulation is 0.458m thick, the ground insulation is 0.099m thick, the windows take up 94.2% of their allowed maximum space, the overhangs are 98.1% as long as the window heights, 12.4% of the window area can be opened, the roof has an emissivity value of 0.672, and the windows are triple-glazed. However, on reflection, this building only has an implausibility value of -0.057 which is not

very negative, indicating a lack of confidence that the building is in fact satisfactory. With the emulators, we can also calculate that this building has a 54.8% chance of meeting the overheating criteria and a 56.4% chance of meeting the energy usage criteria. Neither are particularly large. Additionally, since we have 2 criteria (treated independently), the joint probability of meeting both criteria is thus only 30.9%.

A stricter constraint, but not so strict as $I(\mathbf{x}) < -3$, is therefore a more sensible final requirement. Only considering buildings where $I(\mathbf{x}) < -2$ still provides 1383 buildings to choose from (0.14% of the original candidate buildings). Choosing the building with the largest windows from this subset then leads to a building design with an implausibility of -2.06, a 99.2% probability of meeting the overheating criteria, a 98.0% probability of meeting the energy usage criteria, and an acceptable 97.2% probability of meeting both criteria. This building has 0.195m of wall insulation, 0.379m of roof insulation, 0.092m of ground insulation, the windows take up 52.3% of their allowed maximum area, the overhangs are 86.72% of the window height, 21.9% of the window areas are openable, the roof emissivity is 0.674, and the windows are triple-glazed.[3]

Emulators can be flawed, and so we can also use EnergyPlus to check that this building does indeed meet the criteria. Simulating this building 100 times yields 1 simulation where the building overheats, and an average energy usage of 13.0kWh/m^2, suggesting this building is indeed acceptable.

Several possible criteria for deciding the final building design can be imagined. In practice, the decision depends on the priorities and desires of the practitioners involved, but the illustration above serves as a good example. The emulators built (and improved with the level set methodology) provide the information needed to make such decisions, and provide insight into the various trade-offs a practitioner will have to balance.

To summarise the overall process, wave 1 was essentially unable to find any viable building designs, but it was able to clearly rule-out many designs. Later waves were able to leverage this information by simulating more densely in the NROY/NRIY space, allowing many viable building designs to be discovered.

[3]This building is only the acceptable building with the largest windows out of the initial candidate set of buildings. Further initialisation of candidate buildings and subsequent implausibility checks (using the already obtained emulators) could provide an acceptable building with even larger windows.

4.6 Validation

History matching is based on the notion that regions of simulator space which obviously do not contain the answer need not receive attention. This is what motivates the conservative threshold used to rule-out (and in our case rule-in) input settings, providing some degree of robustness to emulator error. Nonetheless, it is important that the emulators used still represent the simulator reasonably well.

Here, we will validate the emulators used using more standard techniques along with some of the developed diagnostics from Chapter 2. For the more standard diagnostics, we obtain 400 new out-of-sample validation simulations, with points randomly chosen. For the stochastic emulator diagnostics, 80 new out-of-sample locations are chosen, each run 5 times.

4.6.1 Wave 1

To begin with, we validate the wave 1 emulators. For the energy usage emulator, Figure 4.6 presents the standardised error plots.

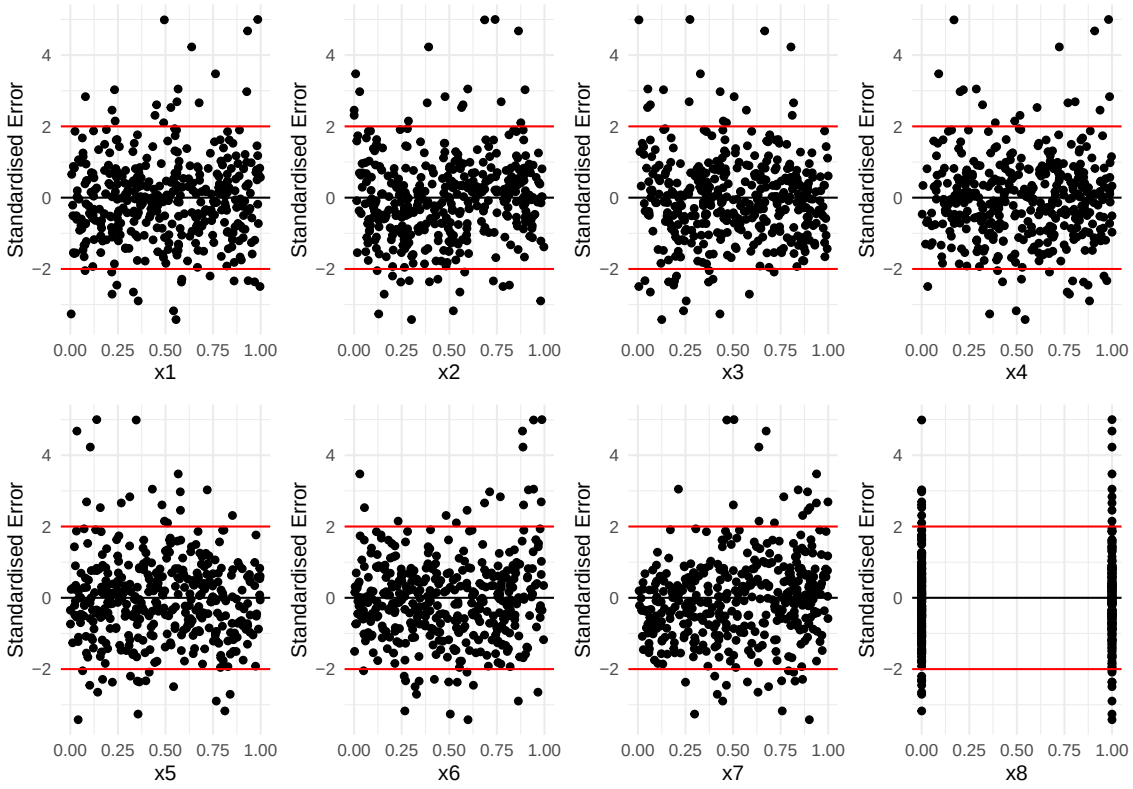


Figure 4.6: 400 individual standardised errors for the wave 1 energy usage emulator.

Here, 33 points lie outside the 2 standard deviation intervals, which implies the wave 1 emulator is overconfident. 4 of the points also have a standardised error

greater than 4. This would usually be cause for concern, providing some degree of doubt into the analysis. However, Figure 4.7 alleviates these worries.

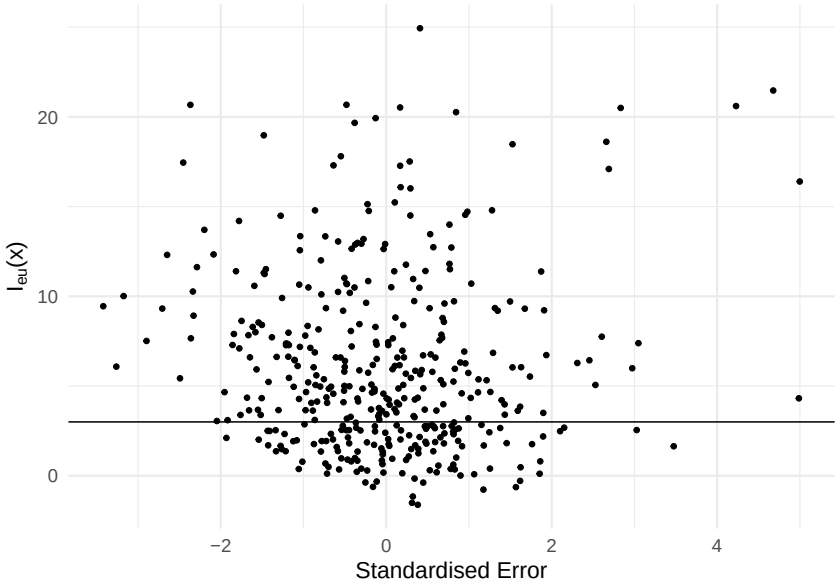


Figure 4.7: 400 individual standardised errors for the wave 1 energy usage emulator plotted against the energy usage implausibility. Also superimposed is the implausibility threshold for ruling-out points.

From this plot, we can see that all of the standardised errors with value greater than 4 are being ruled-out as implausible. Additionally, only considering the not ruled-out space after wave 1, the standardised errors are much nicer, with only 4 out of 130 points outside the 2 standard deviation intervals. Also, with the level set procedure, a standardised error of 4 simply implies that we should be ruling out this point more strongly than we are, which is not a particularly worrying problem. This is one of the strengths of history matching - regions of space which are hard to emulate can be ignored if that region of space is not of interest regarding our objective.

Similarly, we can obtain the sample mean unexpectedness plots (from Chapter 2).

From Figure 4.8 we can see that the mean is, in general, acceptable. There are 4 mean unexpectedness values with absolute value greater than 0.95, which is as we would expect. The extreme unexpectedness values do appear concentrated around small values of x_2 and large values of x_6 (a pattern seen in the standardised errors), but not particularly severely. So, despite the slightly worrying standardised errors, the mean (which is our quantity of interest) appears acceptable.

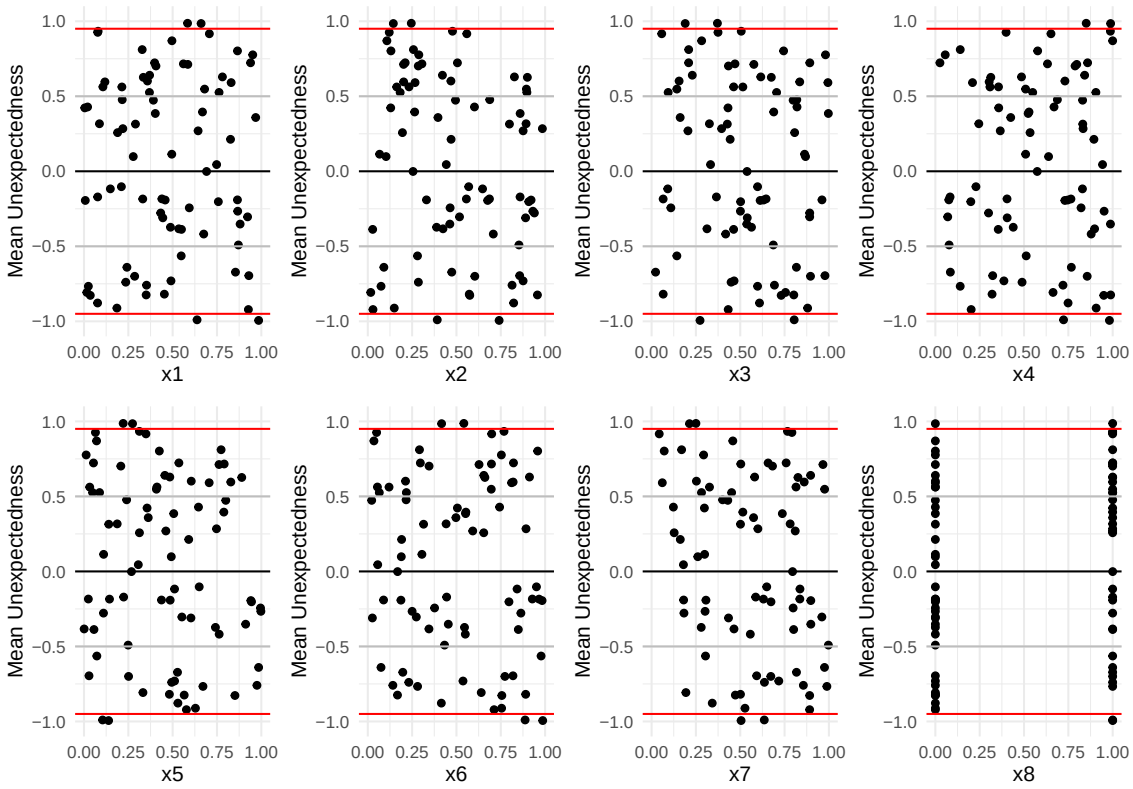


Figure 4.8: 80 sample mean unexpectedness for the wave 1 energy usage emulator.

Validating the overheating emulator is more difficult because the output is only a binary variable (and the quantity of interest is a latent probability). We use the ranked probability scores (Epstein, 1969; Hersbach, 2000), which are metrics for scoring the accuracy of probabilistic forecasts. After obtaining the observed ranked probability score for the overheating emulator, we obtain a reference distribution by sampling multiple hypothetical data sets from the *emulator* and using each to re-score the emulator. Then we can compare this reference distribution for the ranked probability score with our observed score - if the observed ranked probability score does not seem out of place compared to the reference distribution, confidence is gained in the overheating emulators. Using the 400 individual simulations, we obtain a ranked probability score of 0.159. This is less than the 95% sample quantile from the reference distribution (0.174), and so we conclude that the overheating emulator is, in general, acceptable.

As far as we know, obtaining local validation results, wherein the quality of the overheating emulator can be assessed at each input location, is not possible with existing methods; at least not without an abundance of validation data.

We can however, apply the unexpectedness diagnostic developed in Chapter 2.

With 5 simulations per validation point in the replicated validation dataset, we can obtain the proportions of simulations which have the building overheating. These can be 0, 0.2, 0.4, 0.6, 0.8,or 1. We can then also use the overheating classifier to simulate pseudo observed proportions, obtain a distribution for the sample proportions, and then use this to obtain the unexpectedness values for the observed proportions. With this, we can see whether the classifier, in general, overestimates the underlying probability or underestimates it, and whether there are any local problems or clear patterns. Figure 4.9 plots these unexpectedness values for the wave 1 overheating emulator.

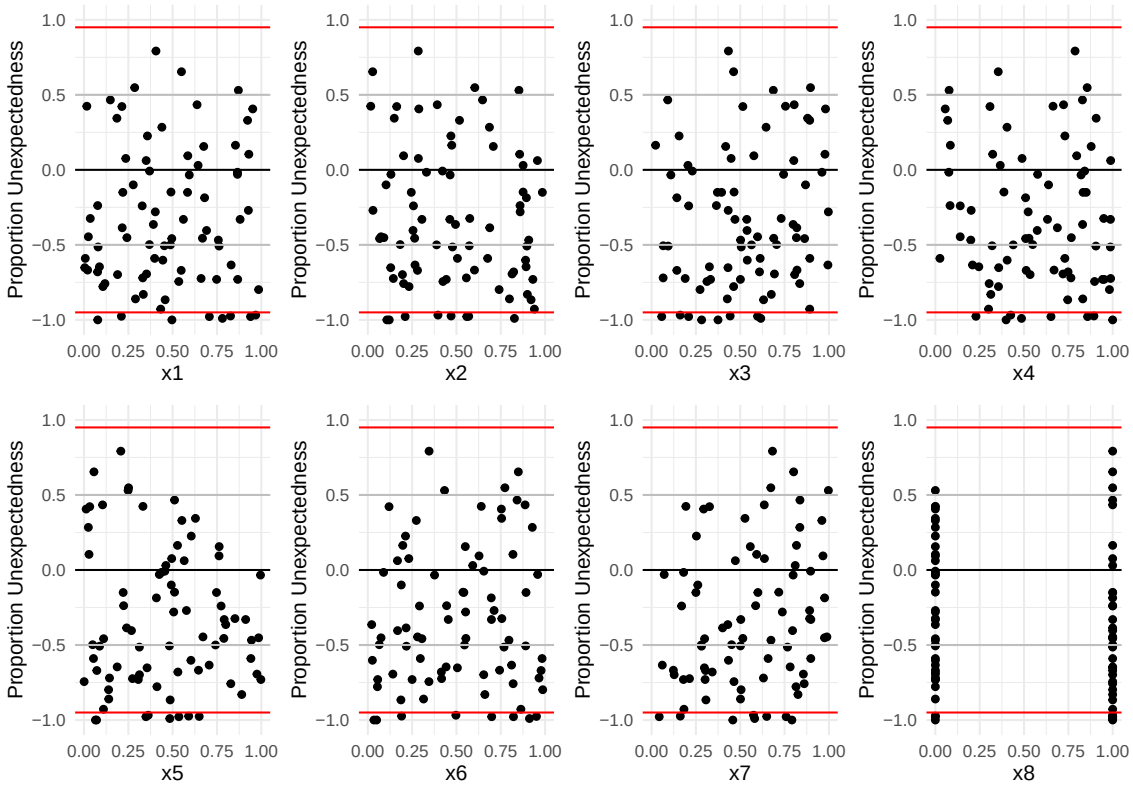


Figure 4.9: 80 uncorrected sample proportion unexpectedness for the wave 1 overheating emulator.

We can see here, that these diagnostics suggests the overheating emulator is poor, with 8 unexpectedness values less than -0.95, and 2 less than -0.995. However, this is primarily because of an issue with the diagnostics, rather than with the emulator.

The proportion of simulations which overheat is not continuous, at least not with a finite number of simulations. With only 5 simulations each, the statistic being investigated is discrete, with 6 possible values. This is an issue. The unexpectedness is based around $P(Z \leq \hat{Z})$, where Z is the statistic of interest, and $\hat{Z}$ is an observed

value for it. With only 6 possible values, there can be a stark difference between $P(Z \leq \hat{Z})$ and $P(Z < \hat{Z})$. For example, if the true sample distribution has 70% of statistics equalling 0 and 30% equalling 0.2, an observed statistic of 0.2 should be acceptable, as this is a likely value. However, $P(Z \leq \hat{Z})$ would yield a value of 1 (which is an extreme value) and $P(Z < \hat{Z})$ would yield 0.7, which is very different (and not an extreme value). With continuous distributions, this is not an issue, as the two converge, but for discrete distributions the choice of $P(Z \leq \hat{Z})$ or $P(Z < \hat{Z})$ has a big impact. In the above example, $P(Z < \hat{Z})$ seems to be the better choice, but this is not guaranteed (it is easy to construct an example where the alternative is true). Additionally, it is possible to construct an example where both are (wrongly) extreme. For example, if the true sample distribution has 100% of statistics equalling 0, then $P(Z \leq \hat{Z})$ would equal 1 and $P(Z < \hat{Z})$ would equal 0; both extreme but indicating the exact opposite problem (the former suggesting an underestimated probability and the latter suggesting an overestimated probability).

Our solution, which is mostly heuristic, is to use the average of $P(Z \leq \hat{Z})$ and $P(Z < \hat{Z})$ in the calculation of the unexpectedness for the overheating proportion. This safeguards against the worst results of having a discrete distribution, although strange patterns can still arise.

Figure 4.10 plots these (corrected) unexpectedness values for the wave 1 overheating emulator.

Here, the results are now much better, with only 1 unexpectedness value with absolute value larger than 0.95. This is perhaps an indication of underconfidence, but this is not particularly worrying.

4.6.2 Wave 2

To check the second wave emulators, we obtain another 400 simulations, with points chosen randomly from NROY/NRIY from the previous wave. For the stochastic emulator diagnostics, we also obtain 80 simulations repeated 4 times in the same way.

Figure 4.12 shows the individual standardised errors for the 400 simulations.

Here the standardised errors are better than in wave 1, with no standardised errors exceeding 4 (although one gets close), and only 26 points outside the 2 standard deviation interval. There does seem to be a local problem for small values of x_2.

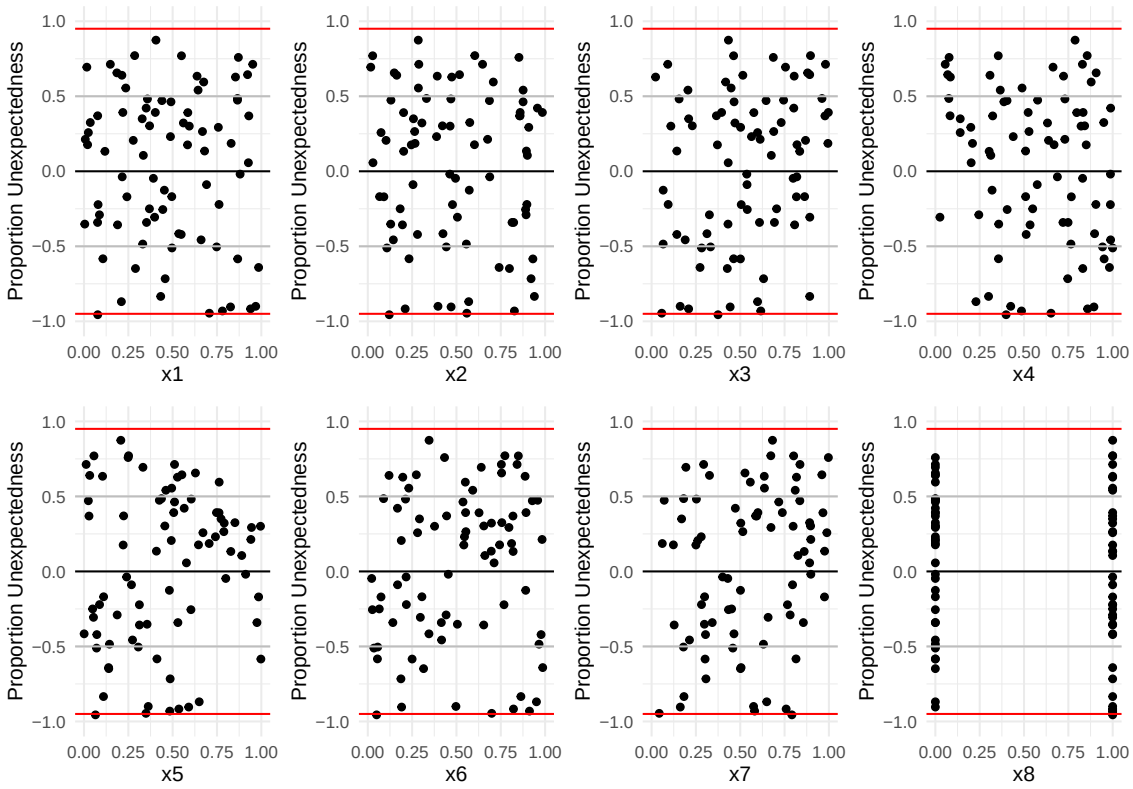


Figure 4.10: 80 corrected sample proportion unexpectedness for the wave 1 overheating emulator.

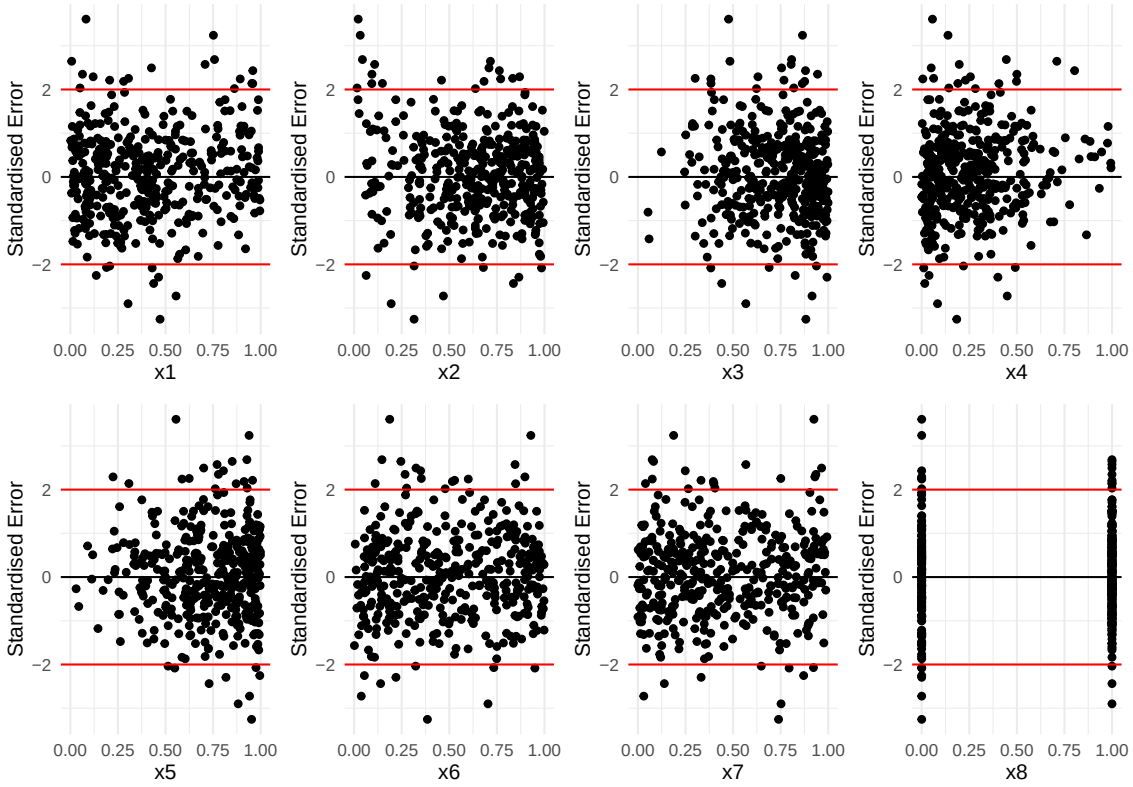


Figure 4.11: 400 individual standardised errors for the wave 2 energy usage emulator.

Again, plotting these standardised errors against the implausibility values, we see that the worst standardised errors are not in the region of primary interest (i.e. they are ruled-out); although for this wave, the standardised errors more generally seem acceptable.

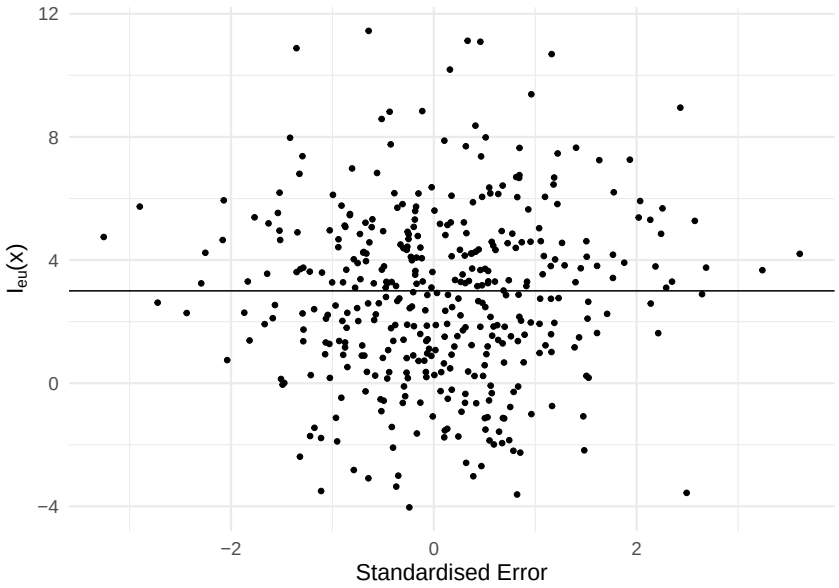


Figure 4.12: 400 individual standardised errors for the wave 2 energy usage emulator plotted against the energy usage implausibility. Also superimposed is the implausibility threshold for ruling-out points.

Figure 4.13 plots the sample mean unexpectedness.

From these we can see that, again, no major problems seem to arise for the wave 2 emulator. There is again a local problem for low values of x_2, which is slightly worrying. However, Figure 4.14 plots the sample mean unexpectedness against the implausibility, and this reveals that the problematic points are ruled-out, and the potential emulator error simply means that they should have been ruled-out with even more confidence.

What is interesting from the validation plots, is that we can begin to see the growth of the ruled-out space in them. For example, we can see a general sparseness of validation points for small x_3, small x_5, and large x_4.

For the overheating emulator, the ranked probability score is 0.029, which is much less than the 95% reference quantile (0.040).

Using the replicated validation data, Figure 4.15 plots the (corrected) proportion unexpectedness for the wave 2 overheating emulator.

These proportion unexpectedness values now appear very strange, with a large

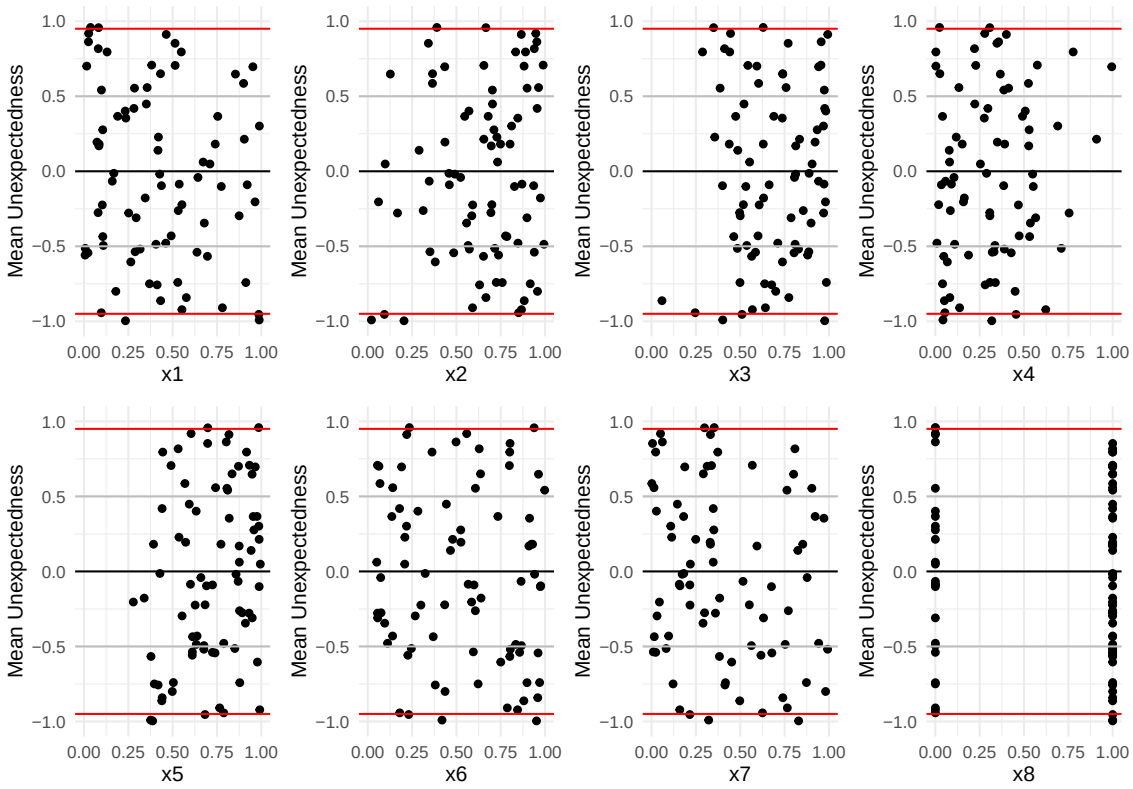


Figure 4.13: 80 sample mean unexpectedness for the wave 2 energy usage emulator.

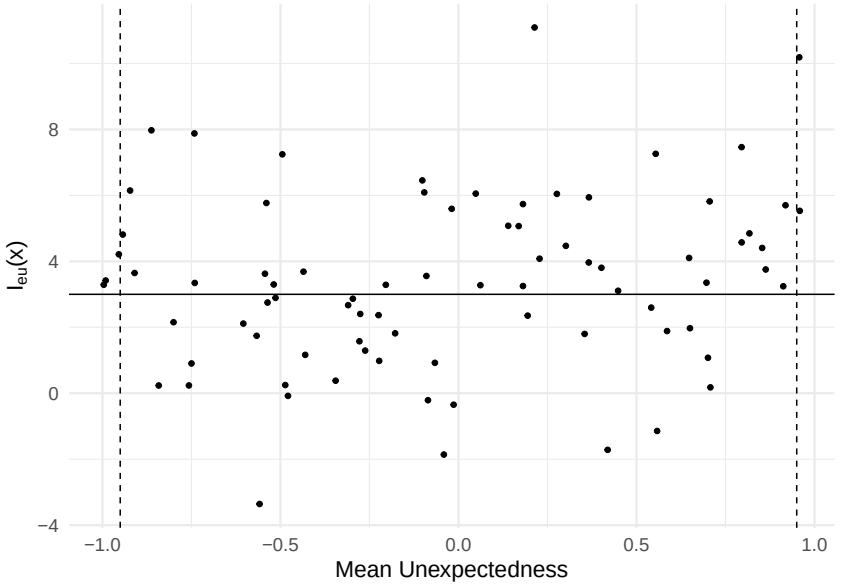


Figure 4.14: 80 sample mean unexpectedness for the wave 2 energy usage emulator plotted against the energy usage implausibility. Also superimposed is the implausibility threshold for ruling-out points.

grouping of points near 0. This is a result of the discrete nature of the distribution, which our correction can only mitigate (preventing spurious extreme values from occurring). Nonetheless, only 2 values have absolute value larger than 0.95, suggesting

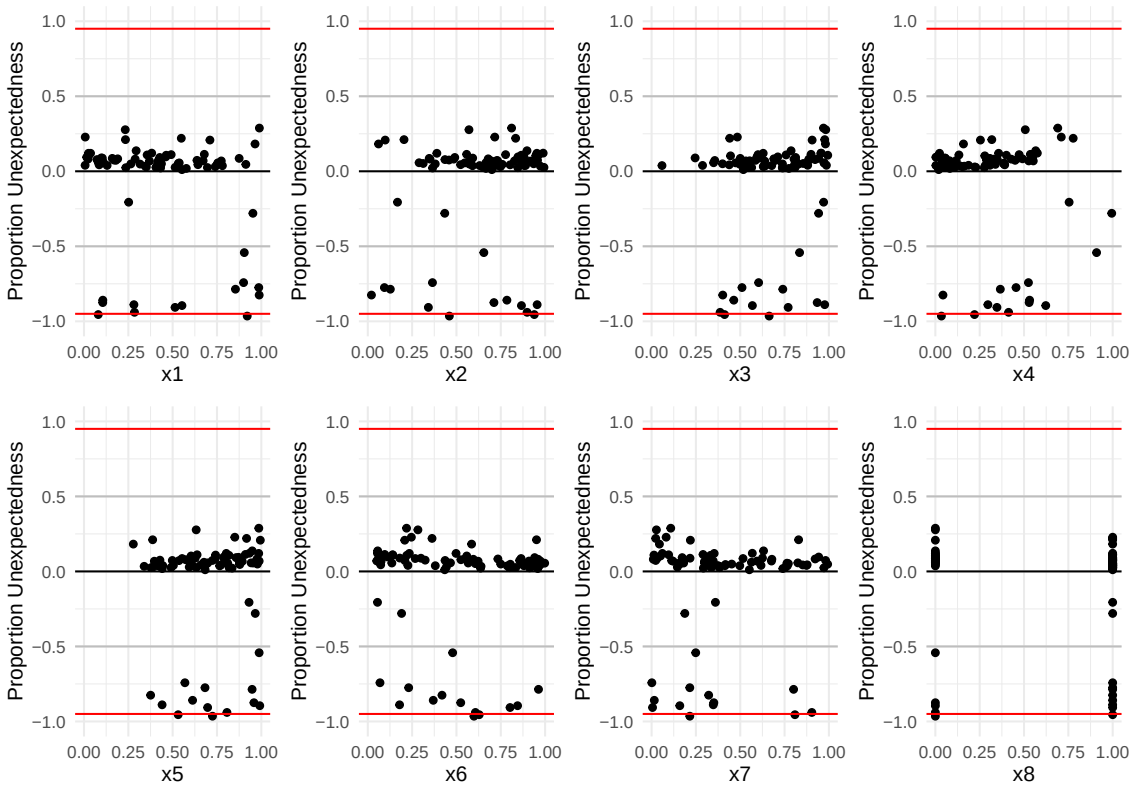


Figure 4.15: 80 corrected sample proportion unexpectedness for the wave 2 overheating emulator.

the emulator is adequate.

Collectively, this evidence suggests the wave 2 emulators are acceptable.

4.6.3 Wave 3

We repeat the same procedure now, but for the wave 3 emulators. Figures 4.16 to 4.18 plot the standardised errors and the sample mean unexpectedness for the wave 3 energy usage emulator.

From these, we can see that, generally, the wave 3 energy usage emulator is also acceptable. There is again potentially some minor local issue for small x_2, with one mean unexpectedness value of -0.992, although this the only problematic point this time. It is likely that the non-stationarity for this input, as observed in Chapter 3, is present for this building; with a large increase in the mean for small values of the wall insulation thickness. Luckily large values are undesirable, and so this region of space is cut-out more and more in each wave; although it does seem that, even by wave 3, the beginning of this increase is still not completely ruled-out yet. This is not a major issue, and by wave 3, the energy usage emulator seems generally acceptable.

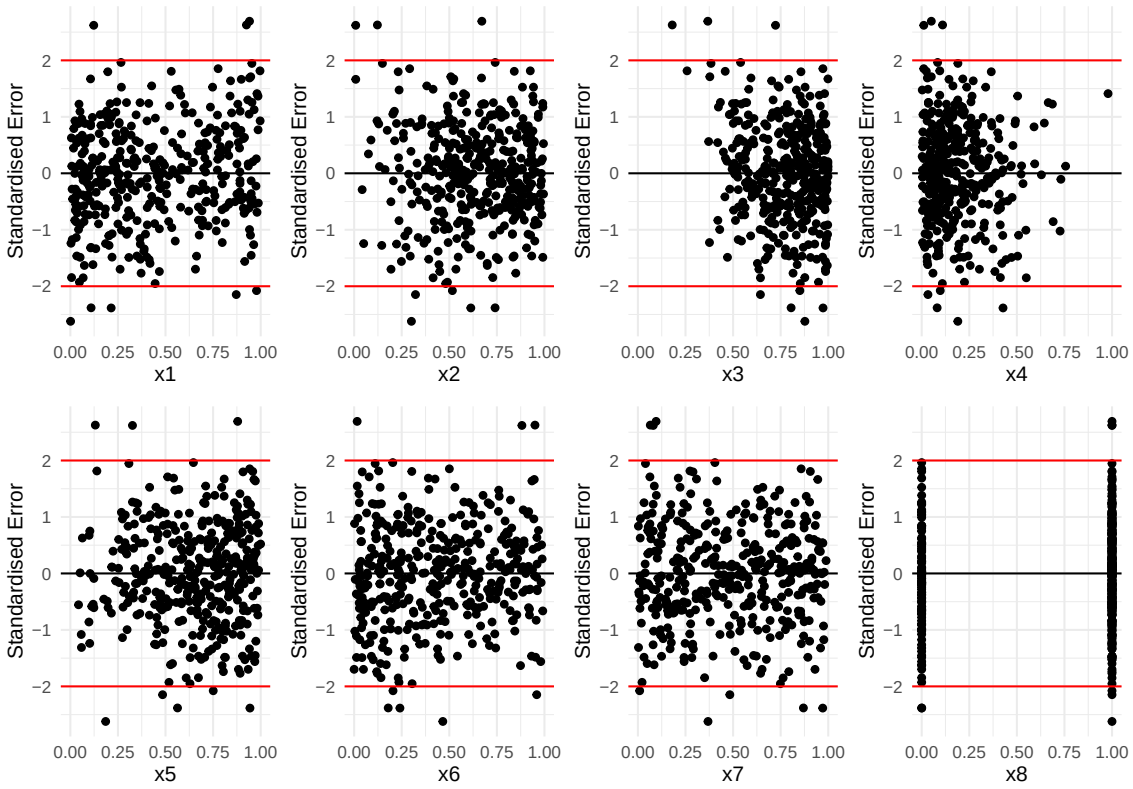


Figure 4.16: 400 individual standardised errors for the wave 3 energy usage emulator.

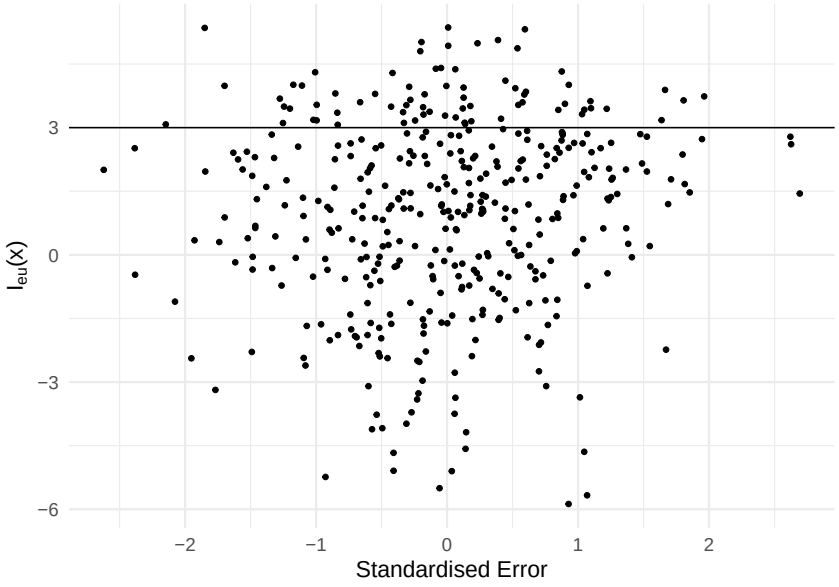


Figure 4.17: 400 individual standardised errors for the wave 3 energy usage emulator plotted against the energy usage implausibility. Also superimposed is the implausibility threshold for ruling-out points.

Additionally, for the overheating emulator the ranked probability score is 0.020, which is less than the 95% reference quantile (0.025).

Using the replicated validation data, Figure 4.19 plots the (corrected) proportion

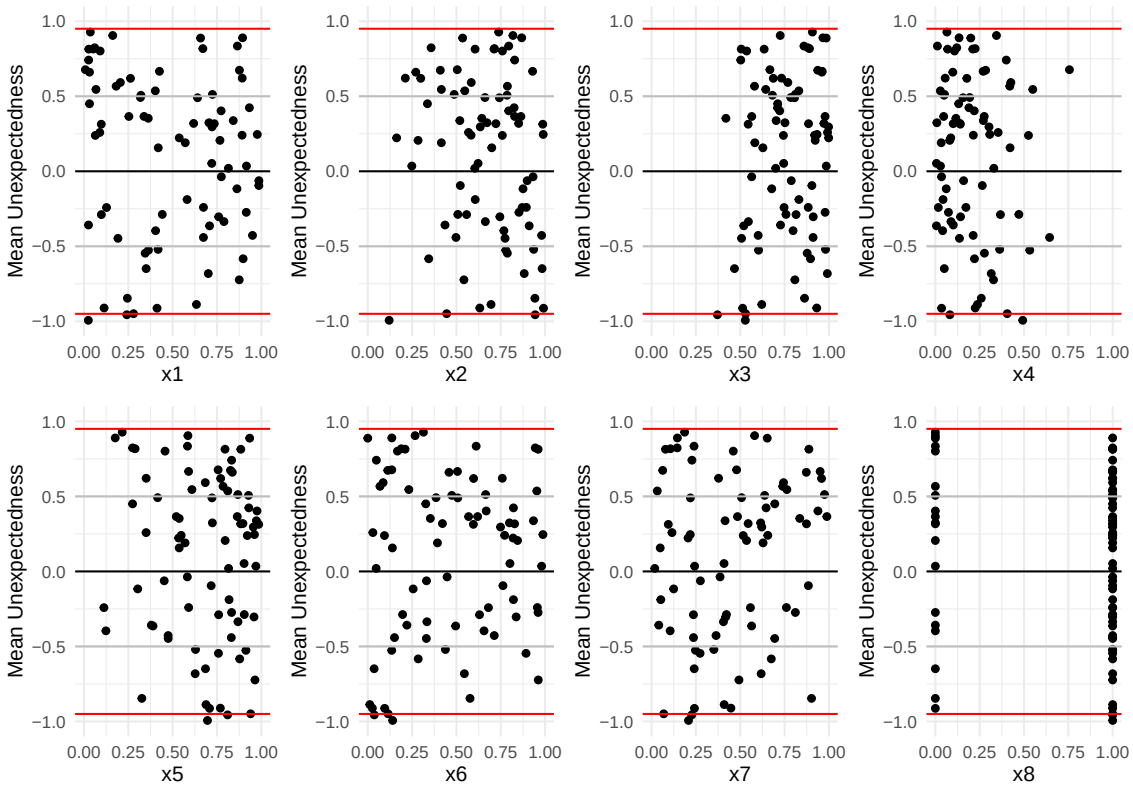


Figure 4.18: 80 sample mean unexpectedness for the wave 3 energy usage emulator.

unexpectedness for the wave 3 overheating emulator.

These again appear strange, with a concentrated grouping of points near 0. Again, this is a result of the discrete nature of the sample distribution, and the correction made to the unexpectedness. However, five of the unexpectedness values have absolute value larger than 0.95, which does begin to raise issues as all are less than -0.95. More worrying is that one of the unexpectedness values has a value of -1. The true value is probably not actually -1, and instead is some value very close to -1, but these unexpectedness values were calculated with 1000 samples from the sample distribution, resulting in some Monte Carlo rounding error. Either way, this indicates a serious underestimation of the overheating probability by the emulator at this location. Upon further inspection, this validation point is for a building which overheated five times out of five simulations. This building was also not ruled-out, as the emulator incorrectly estimated the probability of overheating to be small. This is unfortunate, and is a serious problem. However, in the wave 2 and wave 3 simulations, which were replicated once each, none of the training points overheated twice (out of 2 simulations each). This suggests that we were unlucky to some degree, as the training data in waves 2 and 3 never found any buildings which overheated

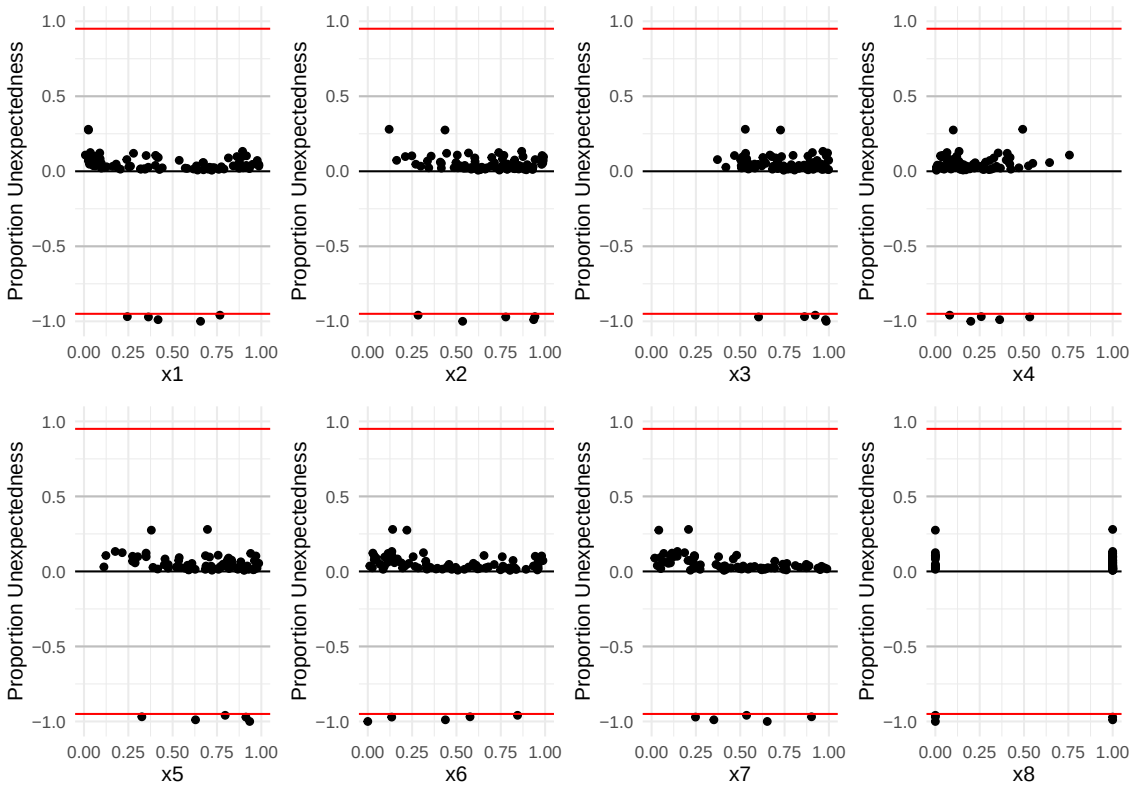


Figure 4.19: 80 corrected sample proportion unexpectedness for the wave 3 overheating emulator.

regularly, but the validation data (which covers space much less) somehow did find one. This extreme point is for a *very* low value (0.000028) of x_6 (window opening amount), suggesting that there may be some highly local pattern near this edge of the input space, which was not picked up by the emulator.

This acts as an important warning - emulators have to interpolate space, and increasingly in higher dimensions they have to extrapolate, which opens up the possibility of missing local relationships. There is little that can be done about this aside from simply simulating more and covering space more, and therefore the possibility will always remain. More practically, for this example, the problematic point does not have an implausibility of less than -2 (0.8825722 overall, -0.7141923 for just the overheating emulator), and so our decision strategy is not compromised. To be safe, one should also avoid choosing any final building with near-0 values for x_6. Nonetheless, this outcome less than ideal, and with the developed classifier unexpectedness diagnostic we have some evidence that the overheating emulator in wave 3 is underestimating the overheating risk to some degree. From this point, we could choose to simulate more from NROY/NRIY and attempt to obtain an

improved emulator, and/or the validation data could be added to the emulator and the combined data-set emulator could be used to make final judgements. For this example, we leave the emulator as it is because, aside from the very extreme point, the remaining 4 unexpectedness values smaller than -0.95 are at least all greater than -0.99 (corresponding to standardised errors less than 2.58), suggesting these overheating predictions may be flawed but not overwhelmingly so.

As such, we leave this finding as a reminder that emulators should only be used as a decision support tool, and any final input choices should be properly simulated and examined before being used in practice.

4.7 Implausibility vs. Probability

History matching descends from Bayes Linear ideas (Goldstein and Wooff, 2007), where statistical modelling avoids the use of probability distributions in favour of only expectations and (co)variances (i.e. first and second order expectations). This is done for computational reasons, and also philosophical reasons (for example, some would argue that expectations and variances are easier to specify than full probability distributions). As a result, implausibility does not use probability in its definition, and instead only expectations and variances are used.

However, history matching is often implemented using probabilistic statistical models, usually full Gaussian process models (Boukouvalas et al., 2014b; Andrianakis et al., 2015; Salter and Williamson, 2016; Lawson et al., 2016). Since these probabilistic predictions are often produced, it is possible to use these probabilistic predictions to rule-out (and rule-in) input space, rather than only the predictive means and variances.

For normally distributed predictions (as with Gaussian processes), in the absence of model discrepancy, there is no computational difference between the two strategies because the normal distribution is defined entirely by its mean and variance. The implausibility threshold of 3 is often used to rule-out points with at least 95% confidence using Pukelsheim's rule. With normally distributed predictions, this is then known to be a stronger statement, ruling out points with approximately 99.7% confidence. With normally distributed predictions, no change in implementation is required; the two are the same and only the interpretation changes. A similar argument is made by those that frame standard history matching (for calibration)

as a form of ABC (approximate Bayesian computation, Holden et al. (2018)). This argument involves assuming a uniform prior distribution for the "true" calibration input setting, and samples from this prior are accepted/rejected according to some distance between the simulator output and observations. Opponents to this view may argue that history matching is different because history matching does not assume that there is one true input setting (which is desirable in part because, since the model may be flawed, one input setting might be a better fit for one output but worse for others).

Regardless, we could still choose to treat a history matching procedure as a probabilistic inference where the decision process about which inputs to consider involves a hard threshold. The wave based procedure of history matching could then be viewed as a fairly standard sequential design scheme, where rather than sequentially choosing individual points according to some optimality criteria (e.g. in the case of Bayesian optimisation), a more conservative decision is taken to "sub-optimally" simulate batches of many runs, constraining possible runs to only the inputs with a sufficient probability of producing matching outputs (or a sufficient probability of being on the boundary in level set estimation).

In this context, the advantages of a history matching design scheme are that it is conservative (with less trust placed in the emulators at any one stage), and that the ruling-out philosophy, where regions of space are iteratively rejected and subsequently ignored, helps to simplify a problem over many waves (which is particularly useful if the output is complex, whether due to multiple outputs, the presence of non-stationarity, or otherwise). Computationally, the two strategies can be the same, but the justification of each is different.

Greater differences between the two strategies appear when model discrepancy is considered. The Bayes linear interpretation becomes more attractive, as it avoids placing a specific distribution on the discrepancy. A probabilistic equivalent to the history matching procedure can still be formulated if one assumes the model discrepancy is normally distributed. In history matching, the data is not used to learn about the discrepancy, and instead the prior beliefs about the discrepancy are maintained (at least formally, in practice the discrepancy terms can be manually changed). This strategy could be replicated in the probabilistic framework, using constant values for a normally distributed discrepancy process and not updating it with data. The computational details of both strategies would then be the same. In

history matching the model discrepancy is sometimes treated as a 'tolerance to error' (Williamson et al., 2017), but the probabilistic interpretation would explicitly believe in a structure for the discrepancy (while believing observed data is not informative about this structure). In both cases, not using data to update the discrepancy is a pragmatic implementation detail because of the issues with learning discrepancy processes (Brynjarsdóttir and O'Hagan, 2014).

Whilst the probabilistic interpretation difference may initially seem trivial, it does have the potential to lead to divergences in results; especially with stochastic simulators. If a stochastic simulator is not modelled with a normal distribution, then whether or not we reject an input according to the implausibility definition as in Section 4.4, or instead via the quantile of the predictive distribution, will lead to differing results. For example, this divergence would have arisen in this chapter if the overheating implausibility was calculated on the probability scale (rather than the logit scale, which maintained normality). In that case, ruling-out points according to whether the required threshold L lay outside 3 standard deviations (the Bayes Linear inspired view) or whether it lay outside the central 99.7% interval (the probabilistic view) would have lead to different results.

The discussion above only goes as far as replacing implausibility with probability (with the ruling-out procedure maintained and the objective still to find 'plausible' inputs). A fully Bayesian procedure which does not update the discrepancy process could also be imagined (with yet more interpretation differences), but this idea and implementation remains future work.

Whilst the Bayes Linear inspired view was used in this chapter, I prefer the probabilistic view, due to its greater interpretability, at least for level set estimation. This can be seen in Section 4.5, where after the final wave, probabilities were used in the discussion of how a final building could be chosen. In this example, probabilities and implausibilities could be used fairly interchangeably, and there is a direct relationship between the two, but this will not always be the case.

The probabilistic view can be additionally useful in the level set context. With two outputs, we ruled-out inputs if the implausibility for either output was greater than 3. With normally distributed predictions, the probabilistic interpretation is then that we ruled-out inputs if the probability of meeting either criteria was less than $\sim 0.13\%$. Treating them separately for ruling-out purposes is somewhat sensible, as it doesn't matter how well one criteria is met if the other criteria is not met.

For ruling-in purposes, treating them separately was not as sensible (and we were therefore, in some ways, lucky that we rarely ruled-in any inputs). Since the outputs were modelled independently, a 99.87% chance of meeting each criteria individually becomes jointly smaller the more outputs there are. With 2 outputs, the joint probability would still be at least 99.73%, and thus this is not a problem, but with many outputs this joint probability threshold could get quite small.

A workaround could involve adjusting the threshold of implausibility where we rule-in points to maintain the same minimum probability of correct joint classification. Using the normal assumption, this would be by using -3.205058 rather than -3 ($\Phi(\sqrt{(}1-\phi(-3))$), where Φ is the standard normal cumulative distribution function). This would be a minor change, and would not have made much difference, as we barely ruled-in any points using the threshold of -3, let alone -3.2. Alternatively, avoiding using the normality assumption for ruling-in (useful say, if a model discrepancy variance was included), the Vysochanskij–Petunin inequality (Vysochanskij and Petunin, 1980) could be used to adjust the value of -3. The inequality states that, for any unimodal distribution, at least $4/9k^2$ of the distribution will be contained within k standard deviations. This could then be used to specify a value for the minimum probability of being in the level set for each output ($\sqrt{0.95}$ would guarantee a joint probability of 95% with two independent outputs), and thus the required implausibility threshold. For our case, this would be -4.47 (or approximately -4.5). This is even more difficult to obtain than -3.2, and so would have had minimal effect on the analysis as well. Note that by not using the normality assumption, the required implausibility to rule-in gets more negative. This implausibility can be even more negative if one uses the Chebychev inequality, which does not assume the distribution is unimodal either.

Alternatively, the joint implausibility measure as in (Vernon et al., 2010) for calibration could be considered. This would account for any correlations between the different outputs. However, this would need modifying to allow for negative implausibilities. Additionally, it would not work as well for level set estimation because it averages the implausibility of the different outputs. With calibration, one output being matched very well and another very poorly is treated similarly to two outputs that are matched moderately well. To see that this is the case, consider a situation where there are two outputs, with observations $\mathbf{y} = (y_1, y_2)$, with two independent corresponding emulator predictions (for a single, specific value for $\mathbf{x}$),

$\boldsymbol{\mu} = (\mu_1, \mu_2)$, and two uncertainties, $\boldsymbol{\sigma^2} = (\sigma_1^2, \sigma_2^2)$. The multivariate implausibility measure, $\mathbf{I}$ would then equal $\begin{pmatrix} y_1 - \mu_1, & y_2 - \mu_2 \end{pmatrix} \begin{pmatrix} 1/\sigma_1^2 & 0 \\ 0 & 1/\sigma_2^2 \end{pmatrix} \begin{pmatrix} y_1 - \mu_1 \\ y_2 - \mu_2 \end{pmatrix}$ which equals $\frac{(y_1-\mu_1)^2}{\sigma_1^2} + \frac{(y_2-\mu_2)^2}{\sigma_2^2}$, which is exactly the sum of the squared individual implausibilities for each output (which we will refer to as I_1^2 and I_2^2). As such, if the two outputs were independent, the multivariate implausibility if I_1^2 was 0.2 and I_2^2 was 12 would be exactly the same as if I_1^2 was 6.1 and I_2^2 was also 6.1. For ruling-in with level set estimation, this is not desirable - we (may) have a clear preference for an input where both outputs are believed to be in the level set with reasonable confidence, over an input setting where we strongly believe one output is in the level set but we are very unsure whether the other is.

This would suggest that a new multivariate implausibility metric should be defined, at least for ruling-in purposes. With the probabilistic view however, this would not be necessary, and we could instead simply decide that any input is ruled-in if it has a joint probability of being in the level set over 99.87% (or some other threshold). This is probably the better decision for multivariate level set estimation, but comparable choices in the implausibility framework remains an open question.

For the example in this chapter, because there were only 2 outputs and we barely ruled-in any buildings, the issue is more academic rather than practical. However, with hindsight, the probabilistic implementation was perhaps preferable.

4.8 Discussion

To conclude, we presented a case study - using ideas from the wider uncertainty quantification community to improve upon a building design. After just three waves of simulation, 92.56% of possible modifications were discarded as implausible. The procedure in general is accessible and would allow an engineer to choose from a set of acceptable building designs according to other more subjective (or less tangible) requirements.

Throughout, we have ignored the notion of 'model discrepancy', where one acknowledges the simulator is not perfect and is itself flawed. It is straightforward and common to add an additive, constant, zero-mean measure of this discrepancy in history matching (Vernon et al., 2014; Andrianakis et al., 2015), by simply replacing the variance term in the implausibility equation, $V(y(\mathbf{x}))$, with $V(y(\mathbf{x})) +$

V_{MD}, where V_{MD} represents the subjective uncertainty around what the difference between the simulated quantity and the real world quantity could be. If V_{MD} is not believed constant, or additive, or zero-mean (all possible within a level set estimation procedure), then more must be done. Model discrepancy is an important question when it comes to any form of simulator analysis (Goldstein and Rougier, 2009). How to include model discrepancy into building design requirements is a question that requires discussion between statisticians, practitioners and policymakers. Additionally, when it comes to level set estimation, there is an implicit relationship between the required threshold and the model discrepancy. For example, there is little difference between mandating that a building use no more than 15kWh/m^2 while not accounting for model discrepancy, and mandating 10kWh/m^2 but with a belief that the model discrepancy could be up to 5kWh/m^2.

Additionally, the level set methodology we use is certainly not the only one available. For calibration, there are alternative methods to history matching; for example, standard Bayesian inference (Kennedy and O'Hagan, 2001) provides an alternative methodology, although this is not without its flaws (Brynjarsdóttir and O'Hagan, 2014). The problem of efficient simulator level set estimation is also an open research question, and other efforts exist (see Lyu et al., 2018, for a review of some alternatives). A comprehensive comparison between history matching techniques in general and alternatives is sorely missing from the literature, and remains an important topic for future research. In practice, history matching techniques serve as batch design schemes which are easily implemented and understood for a wide range of problems, whilst also being fairly robust due to the ruling-out process and the conservative nature. An initial comparison between history matching and sequential strategies for level set estimation (in the next chapter) points mildly to the increased value of the latter. As such, this case-study could perhaps have involved adapting these sequential strategies to account for multiple outputs (with one being a classifier, rather than a standard emulator) instead. In any case, for our problem the history matching technique was sufficient and could provide value to practitioners. Although we never reached it, the concept of ruling-in is interesting and also optional - if one is interested in the specific performance of the buildings beyond simply whether or not they meet the required criteria, buildings need not be ruled-in, and the level set design scheme will then continue to improve all not ruled-out buildings.

For the two stochastic outputs we considered (energy usage and overheating

classification), we were interested in improving specific summary statistics for each; the mean energy usage and the overheating probability. For the binary overheating classification output, the overheating probability fully summarises the output. For the energy usage however, summaries other than the mean could have been used with a different interpretation. In this case, because buildings are often used for many years and the energy usage for any one individual year isn't particularly important, the mean average is a sensible quantity of interest. For many problems however, other quantities of interest can be more important, such as the median or the 90% quantile. The outlined level set procedure can still work in these situations, although different emulators might be more suitable, such as a quantile Kriging emulator (Plumlee and Tuo, 2014).

If a large number of additional waves were to be performed, it would also be possible for the number of NROY/NRIY buildings to reach 0; with all candidate buildings either ruled-out or ruled-in. This would imply that all possible buildings are have been categorised as satisfying or not-satisfying the criteria. However, because we only considered 1000000 possible designs, it is also possible that other hypothetical building designs would still be both NROY and NRIY. In these scenarios, several methods exist in the history matching literature for sampling new points from the not ruled-out space (such as the method from Drovandi et al. (2017)), allowing the level set boundary to be more precisely defined. We don't believe this to be particularly useful however, as there is a limit to how precisely a building can actually be built in practice, and so there is a limit to how precisely we would want to design a building.

For this specific example, we also found some problems with the overheating emulator in the final wave. It is possible that the strategy of throwing away data between each wave (because the domain of the emulator is reduced in each wave to exclude ruled-out simulations) has made it difficult for the emulator to learn the latent probability process. By the third wave, very few of the training simulations were 1s (i.e. very few of the simulated buildings ever overheated). In fact, only 7 of the 770 simulations used to train the wave 3 overheating emulator actually resulted in an overheating building, a result of the ruled-out space having excluded the worst buildings already. This is good news in the sense that it confirms that most buildings with a high propensity to overheat have been ruled-out, but it is also bad news because it can make it difficult for the emulator to learn the global patterns in the overheating probability. For the overheating emulator, a better strategy here

might have been to make use of all the training data (not excluding ruled-out data from previous waves), allowing the global structures to be learnt. However, we also found that the energy usage likely exhibits some degree of non-stationarity, and the throwing away of data helped here (which may also be true for the overheating emulator). Without a consensus here, for future examples a better strategy might be to target easier overheating probability thresholds (say 5% rather than 1%) or to include more simulations in each wave (which is possible because of the simulator's relative cheapness). We discuss the decision to shrink the domain of the emulator in history matching more in the next chapter.

Furthermore, a more comprehensive study might have included a random occupancy profile as well as a random weather generation process. This would have accounted for more of the uncertainty we have when modelling buildings without requiring major methodological changes. Similarly, UKCP09 is out-of-date at this point, and a different weather generator would have been preferable (even if this required reducing the scale of the case-study to only modern day weather). An additional extension to this work could involve increasing the number of inputs and outputs. 8 inputs is not small for emulation, but it is possible that a practitioner may want to adjust many more inputs in a building design (for example, say every window were to be adjusted individually). With outputs, the building used was reasonably simple, with only one "thermal zone". In more complex buildings with more thermal zones, each thermal zone would produce its own overheating outcome and energy requirements, increasing the number of outputs.

Regardless, we believe that emulation, the described level set methodology, and the ideas discussed more generally can be useful tools for the field of building performance simulation. Practical improvements to existing or planned buildings could easily be facilitated by utilising the tools outlined here, and accounting for the uncertainty caused by weather (or indeed other 'random' variables) can be important.

Chapter 5

Comments on History Matching

5.1 Introduction

The previous chapter showed how history matching techniques could be used to aid building design decisions. This chapter will discuss, compare, and comment on history matching in the context of level set estimation, and also more generally.

The first main question we discuss here is about whether the ruling-out procedure of history matching, where the domain of the emulator is shrunk between waves, is practical and efficient. This is explored in Section 5.2. We find that shrinking the domain helps with extremely non-stationary problems, but maintaining a larger domain helps when there are multiple distinct regions of space. An added benefit of not shrinking the domain is that the NROY/NRIY space can then grow, as well as shrink, between waves because the resulting emulators are valid beyond where the current NROY/NRIY space is.

The second main question relates to how history matching compares with alternative design schemes. For level set estimation, Lyu et al. (2018) provide several fully sequential design schemes for stochastic problems, adapted from the wider level set estimation and optimisation literature. Section 5.3 compares history matching level set estimation with these fully sequential alternatives. Our general conclusion is that these optimal designs can sometimes be better than history matching, although history matching is substantially easier to implement and faster to compute, making it more suitable for relatively fast simulators.

All of the examples deal here with homoscedastic simulators and therefore homoscedastic emulators. This was done as a compromise between investigating the full

heteroscedastic case and the deterministic case. Extending the discussion explicitly to the heteroscedastic case or the deterministic case remains future work.

5.2 The Ruling-Out Procedure

Our first set of experiments deal with the ruling-out procedure of history matching.

At each wave in history matching, a new NROY(/NRIY) space is calculated. This NROY(/NRIY) space is a set of inputs which we believe could still lead to an acceptable match (where 'match' means possible agreement with observed outputs in calibration, or possibility of lying on the boundary in level set estimation). For additional simulation, the NROY(/NRIY) space can be sampled from to provide a new wave of more focussed simulations in the region of interest, improving the emulator in this region (helping to further shrink the NROY(/NRIY) space). It is also common to shrink the domain of the emulator (i.e. the region in which the emulator is valid) to be equal to the current NROY(/NRIY) space (Vernon et al., 2010; Lawson et al., 2016; Andrianakis et al., 2017). This means that simulations from the previous wave(s) which were ruled-out (or potentially ruled-in) are not used to fit the new wave's emulator, because they are not within the domain of the new emulator. This also means that the NROY(/NRIY) space will only ever shrink (or at the very least, maintain its current volume), it can never grow because the subsequent emulator(s) will not (and cannot) be used to make predictions in the ruled-out (or ruled-in) space.

It can be argued that shrinking the domain of the emulator helps deal with non-stationarity, as the underlying system is more likely to be stationary in the smaller region of space and so a stationary Gaussian process assumption becomes more valid (Salter and Williamson, 2016). For some models, a philosophical argument could also be made that inputs outside of the NROY(/NRIY) space are not physically possible, and thus should not be considered. However, for many models, ruled-out inputs may still be physically possible but are not relevant for the given objective (for example, a calibration objective in Baker et al. (2020b) is to determine the number of fish in an area - all numbers of fish are physically possible, but only one is the true value). Additionally, non-physical input settings can still be informative about physical input settings, especially if the simulator response is relatively smooth.

The focus of this section is to explore the merits of this domain shrinkage in

moderate contexts. Problems where the initial domain is incredibly large (for example, the final NROY space in Andrianakis et al. (2015) was $1.3 \times 10^{-9}\%$ of the initial domain), will likely require *some* shrinking of the domain, unless the simulator is incredibly simple. This section does not conclude that one is universally better than the other, but it is interesting to compare the two different strategies, especially given that shrinking the domain is common with history matching (Vernon et al., 2014; McKinley et al., 2018; Andrianakis et al., 2015; Williamson et al., 2017; Salter and Williamson, 2016; Lawson et al., 2016; Baker et al., 2021), but less common in other objective-focussed design strategies in the wider statistics and machine learning literature (including optimisation (Jones et al., 1998; Frazier, 2018), global emulator design (Boukouvalas et al., 2014a; Binois et al., 2019), level set estimation (Picheny et al., 2010; Lyu et al., 2018), and calibration (Damblin et al., 2018)).

Our examples focus on the level set history matching modification outlined in the previous chapter. When the domain is shrunk, we shrink to only the NROY/NRIY space (whereas in the previous chapter, we did not exclude the ruled-in space). This can be sensible if the goal is only to find the boundary of the level set; if the goal is instead to find and investigate those inputs which are ruled-in, then the domain should include the ruled-in space as well. This set-up mirrors the calibration procedure, and so any discussion should be broadly relevant for both objectives.

5.2.1 Should We Shrink the Domain?

Consider the 2 dimensional toy simulator in Equation (5.1).

$$\begin{gathered} y = 12x_1x_2 - 5(1 - x_1x_2)^{12}\sin(40\pi x_1x_2) + \epsilon; \\ \epsilon \sim N\big(0, 0.5\big). \end{gathered} \tag{5.1}$$

This simulator is *extremely* non-stationary, with a very complex mean for low value of x_1 and x_2, but a very simple mean for large values.

We fit an initial homoscedastic emulator (using the `hetGP` package). To ensure the non-stationarity in the mean is picked up (and not smoothed over, as in examples in Chapters 2 and 3), we fit the emulator to 200 initial simulations at 20 unique locations, each replicated 10 times. We then perform level set estimation history matching, with 3 additional waves of data each of 20 runs, assuming our goal is to find the boundary where the output equals 5 (ruling-in for regions that are less than 5, ruling-out for those above).

This is done twice, one time shrinking the domain at each wave to be equal to the current NROY/NRIY space, and the other time maintaining the full domain. In both cases, the current NROY/NRIY space is used to determine where new simulations can be, and the NROY/NRIY space at each wave must be a subset of the NROY/NRIY space of the previous wave. Figure 5.2 shows the predictive surfaces of these final emulators for each procedure, along with the shapes of the ruled-out and ruled-in spaces.

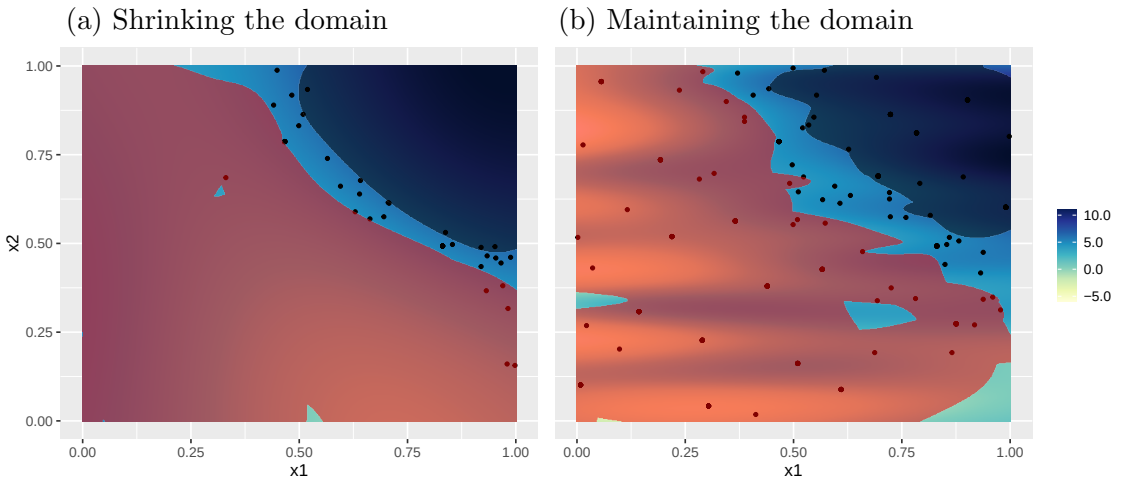


Figure 5.1: Predictive mean surfaces for the two emulators of the simulator in Equation (5.1) after three waves of level set estimation history matching. The emulator on the left shrank the domain between each wave and the emulator on the right did not. Also superimposed is a transparent black overlay representing the ruled-out space, and a transparent red overlay representing the ruled-in space.

With these plots, the ruled-out space is represented by a superimposed transparent black overlay, and the ruled-in space is represented by a superimposed transparent red overlay. In standard history matching for calibration, both the black and red regions would be the ruled-out space, with the black space representing the region where the output is too large, and the red region representing the region where the output is too small.

We can see from these plots that the NROY/NRIY space for the strategy which does shrink the domain is much smoother and smaller, even though fewer simulations in total are used to fit the final wave emulator. This is because, by shrinking the domain between each wave to only apply to the NROY/NRIY space, the emulator only needs to be able to model a smaller, simpler, region of space. The strategy which did not shrink the domain has to model the entire initial input space in each

wave, which it struggles to do using a stationary Gaussian process (given the output is not stationary over the initial input space).

With a toy simulator, we have access to the truth, and we therefore know how accurate the emulators are with their final wave predictions. In this example, shrinking the domain not only increases the degree of refocussing, but it also leads to a more accurate final wave emulator. We can calculate the error of the final classifications (for large a set of potential input combinations, what percentage of predictions classify the output on the wrong side of the level set, taking any ruled-out points as above the level set and ruled-in as below), and the bias (of those which are classified wrongly, the percentage which are classified as too low minus the percentage classified too high)[1]. For the strategy which shrinks the domain, the error is 0.966% and the bias is -0.957%. The strategy that does not shrink the domain results in an error of 2.21% and a bias of -1.28%, which is substantially worse.

By shrinking the domain, a stationary Gaussian process assumption becomes more valid in the NROY/NRIY space, which leads to better results, as might be expected.

This is not always the case, however. Consider the toy simulator in Equation (5.2):

$$\begin{aligned} y &= \sin(3\pi x_1)\cos(3\pi x_2) + \epsilon; \\ \epsilon &\sim N\big(0, 0.05\big). \end{aligned} \tag{5.2}$$

This simulator has many local minima and maxima and, assuming we are targeting the regions of space where the mean equals -0.5 (ruling-in if the mean is less than -0.5, and ruling-out if it is more than -0.5), the NROY/NRIY space will be in many distinct locations. For this example, we fit an initial emulator using 20 runs, and then perform level set estimation history matching, with 3 additional waves of data each of 20 runs. This is again done twice, one time shrinking the domain, and the other time maintaining the initial domain. Figure 5.2 shows the predictive surfaces of these final emulators for each procedure, along with the shapes of the ruled-out and ruled-in spaces superimposed.

[1]In standard calibration the notion of a "true NROY" set (Salter and Williamson, 2016) can also be used to assess history matching performance. With level set estimation the accuracy of the level set estimates can be more interesting, and this directly corresponds with the accuracy of the ruled-out and ruled-in sets when simulation data is abundant.

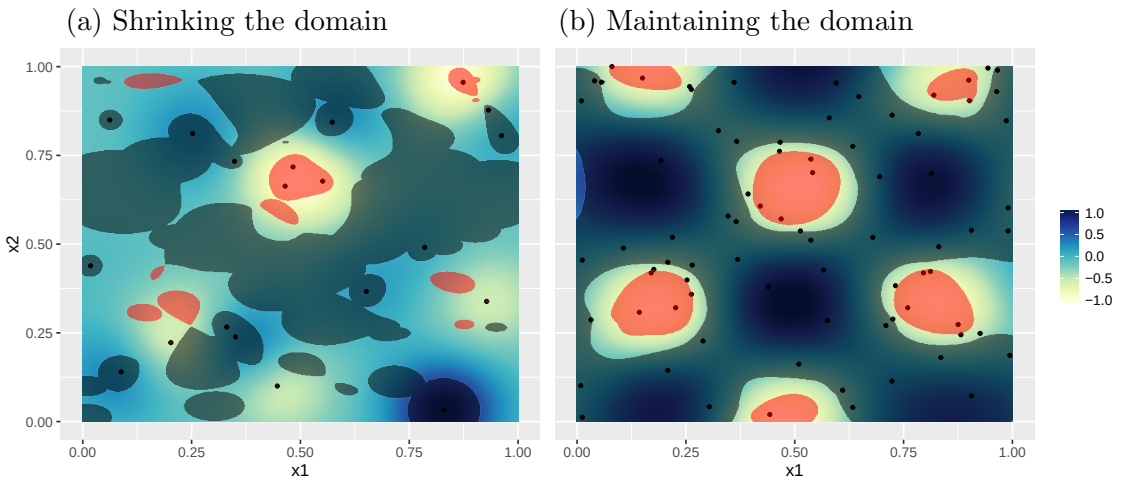


Figure 5.2: Predictive mean surfaces for the two emulators of the simulator in Equation (5.2) after three waves of level set estimation history matching. The emulator on the left shrank the domain between each wave and the emulator on the right did not. Also superimposed is a transparent black overlay representing the ruled-out space, and a transparent red overlay representing the ruled-in space.

Interestingly, these plots show that shrinking the domain leads to a greatly reduced amount of refocussing, with much less of the space ruled-out or ruled-in. Maintaining the initial domain leads to an NROY/NRIY space which is smaller, and also more cohesive and smoother.

For the strategy which shrinks the domain, the error is 10.7% and the bias is −8.46%. The strategy that maintains the larger domain results in an error of 2.57% and a bias of −0.689%, which is substantially better (less than a third the value for both metrics).

This points to how shrinking the domain can, at least in some circumstances where there are multiple distinct regions of NROY/NRIY space, be a poor strategy. Andrianakis et al. (2017) do argue that in an ideal situation such distinct regions of space would be identified and emulated separately, but they also acknowledge that such identification can be difficult. For this example, a stationary Gaussian process for the initial domain was an acceptable assumption, and shrinking the domain (and thus reducing the number of simulations used to fit the later-wave emulators) is detrimental. If a stationary Gaussian process is a reasonable assumption, then shrinking the domain can be a poor decision to make, because simulations which are ruled-out (or ruled-in) can still be informative about the region of interest. In fact, shrinking the domain implicitly claims that these ruled-out (or ruled-in) simulations

are not informative about the region of interest, which may not always be accurate (especially if the simulator response is particularity smooth or there are key global structures).

With extreme non-stationarity seemingly favouring a strategy which iteratively shrinks the domain, and stationarity seemingly favouring a strategy which maintains the domain, it becomes interesting to see if either strategy is preferable in the presence of weak stationarity.

Consider the toy simulator in Equation (5.3).

$$\begin{gathered} y = \sin(1/((0.7x_1 + 0.3)(0.7x_2 + 0.3))) + \epsilon; \\ \epsilon \sim N\big(0, 0.1\big). \end{gathered} \tag{5.3}$$

This is the non-stationary toy simulator explored in Ba et al. (2012) and Volodina and Williamson (2020), with some additional white noise. Fitting an initial emulator with 20 simulations, and then adding 3 waves for 20 simulations via history matching (targeting the regions of space where the mean equals -0.25, ruling-in if the mean is less than -0.25 and ruling-out if it is more than -0.25), we obtain the results in Figure 5.3.

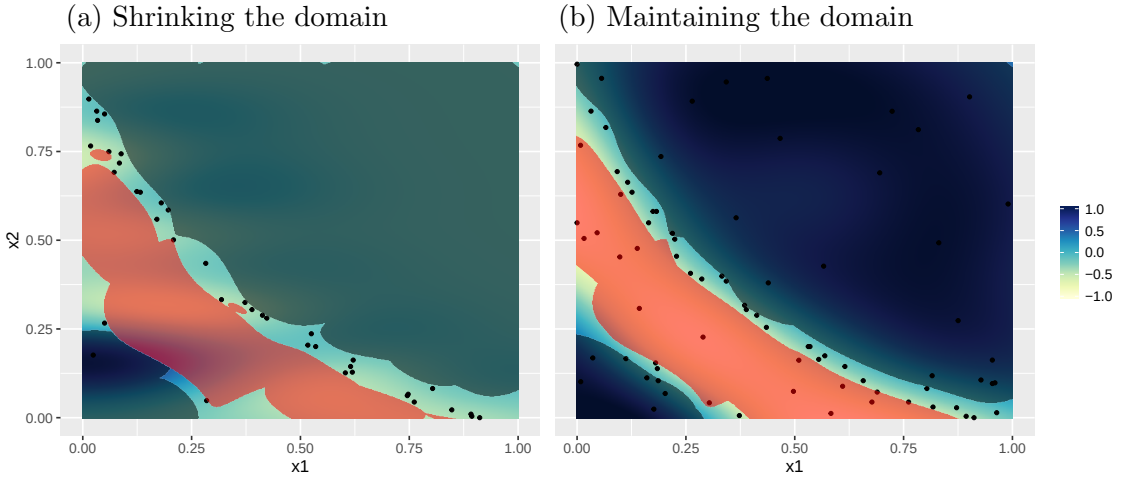


Figure 5.3: Predictive mean surfaces for the two emulators of the simulator in Equation (5.3) after three waves of level set estimation history matching. The emulator on the left threw shrank the domain between each wave and the emulator on the right did not. Also superimposed is a transparent black overlay representing the ruled-out space, and a transparent red overlay representing the ruled-in space.

These results are both very similar, and visually difficult to distinguish or decide whether one is clearly better. Both seem to shrink the NROY/NRIY space relatively

equally. Numerically, the error for the strategy that shrinks the domain is 4.16%, and the bias is 0.311%; whilst the error for the strategy that does not shrink the domain is 1.14%, and the bias is −0.408%. This suggests that of the two, the strategy that maintains the domain has the edge. Even though the simulator is non-stationary, the region(s) of interest is in the more complex part of the initial domain, and so shrinking the domain is less useful here. Additionally, NROY/NRIY space again appears to be made up of two distinct regions, which likely favours maintaining a larger domain.

For another example, in higher dimensions, we look at the 10 dimensional wing weight simulator from Forrester et al. (2008) in Equation (5.4), used by Davis et al. (2020) to test non-stationary emulators. This simulator is modified to include some intrinsic noise.

$$y = 0.036 S_w^{0.758} W_{fw}^{0.0035} \left(\frac{A}{\cos^2(\Lambda)}\right)^{0.6} q^{0.006} \lambda^{0.04} \left(\frac{100 t_c}{\cos(\Lambda)}\right)^{-0.3} (N_z W_{dg})^{0.49} + S_w W_p + \epsilon;$$
$$\epsilon \sim N\big(0, 5\big), \tag{5.4}$$

where

$$S_w \in [150, 200], \quad W_{fw} \in [220, 300], \quad A \in [6, 10], \quad \Lambda \in [-\pi/18, \pi/18],$$
$$q \in [16, 45], \quad \lambda \in [0.5, 1], \quad t_c \in [0.08, 0.18], \quad N_z \in [2.5, 6],$$
$$W_{dg} \in [1700, 2500], \quad W_p \in [0.025, 0.08].$$

We perform three waves of level set estimation history matching for both strategies, using an initial emulator built from 100 simulations. Each subsequent wave involves an additional 100 simulations and we target the regions of space where the mean equals 300 (ruling-in if the mean is less than 300, and ruling-out if it is more than 300). Shrinking the domain at each wave leads to an error of 1.42% and a bias of −0.292%. Maintaining the larger domain leads to an error of 1.24% and a bias of −0.283%. These results are approximately the same, with the strategy that maintains the domain taking a slight edge. Here we find that, even for non-stationary problems, even in reasonably high dimensions, maintaining a larger domain can be an acceptable decision.

Collectively, these examples are frustrating. There does not seem to be a good one-size-fits-all recommendation for whether one should shrink the domain or maintain the initial domain. The common belief that non-stationarity favours shrinking the

domain does seem to be true, but only when said non-stationarity is sufficiently strong. On the other hand, maintaining the domain can help when there are important global properties to capture and when a stationary Gaussian process assumption is reasonable (if a stationary Gaussian process emulator is used).

5.2.2 Flexible NROY

There is another advantage to maintaining a larger domain. Consider another 2D example, this time the one given by Equation (5.5), with history matching level set estimation results given in Figure 5.4 (after an initial emulator using 20 simulations, adding 3 waves of 20 extra simulations, and targeting regions where the mean equals 1).

$$y = -2exp\left(-\frac{(x_1 - 0.8)^2}{0.3}\right) exp\left(-\frac{(x_2 - 0.2)^2}{0.3}\right) + \epsilon; \quad \epsilon \sim N\big(0, 0.15\big). \tag{5.5}$$

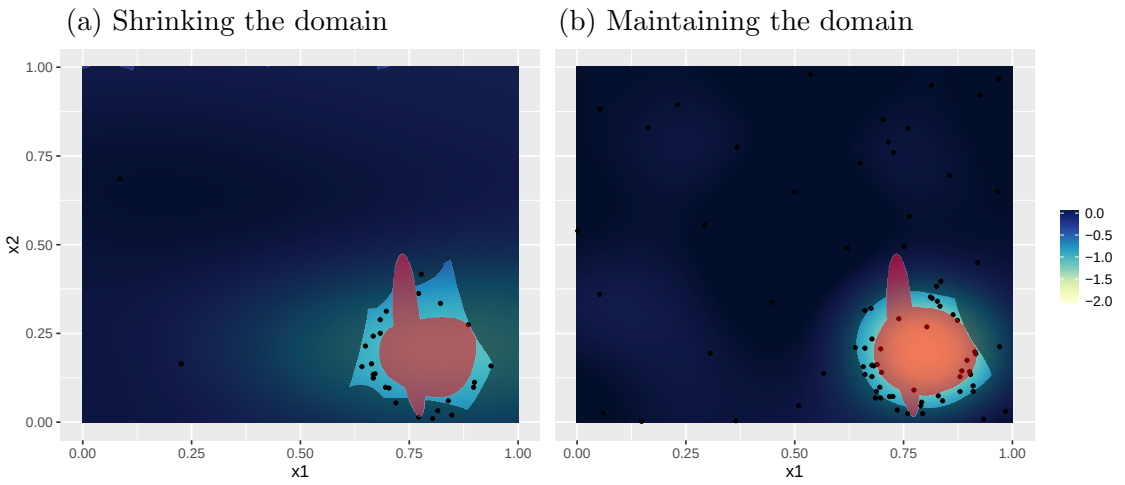


Figure 5.4: Predictive mean surfaces for the two emulators of the simulator in Equation (5.5) after three waves of history matching level set estimation. The emulator on the left threw shrank the domain between each wave and the emulator on the right did not. Also superimposed is a transparent black overlay representing the ruled-out space, and a transparent red overlay representing the ruled-in space.

We can see from the plots, for both strategies, that there is a tall oval of ruled-in space which seems to be erroneous. This is ruled-in in the initial wave, and history matching does not allow for this mistake to be rectified in later waves. This is because, at least when the domain is shrunk between waves, the emulator is only

valid for the current NROY/NRIY space; as such, one cannot have confidence in any emulator predictions in the ruled-out (or ruled-in) space. This issue is present in all the examples previous to some extent, but it is particularly worrying for this example because the ruled-in space appears to extend far into what should perhaps instead be ruled-out.

For completeness, the error for the strategy which shrinks the domain is 8.75%, the bias −8.26%, the error for the strategy which maintains the initial domain is 7.66% and the bias is −7.17%.

With the strategy where the initial domain is maintained, the ruled-out space (and the ruled-in space for level set estimation) can be made flexible - recalculating it each wave for the entire input space, allowing it to grow as well as shrink. This would allow the emulator to correct previous misclassifications, something not previously possible.

Doing so for this example, performing the same history matching as before but this time allowing NROY and NRIY to be flexible between waves, we obtain the results in Figure 5.5.

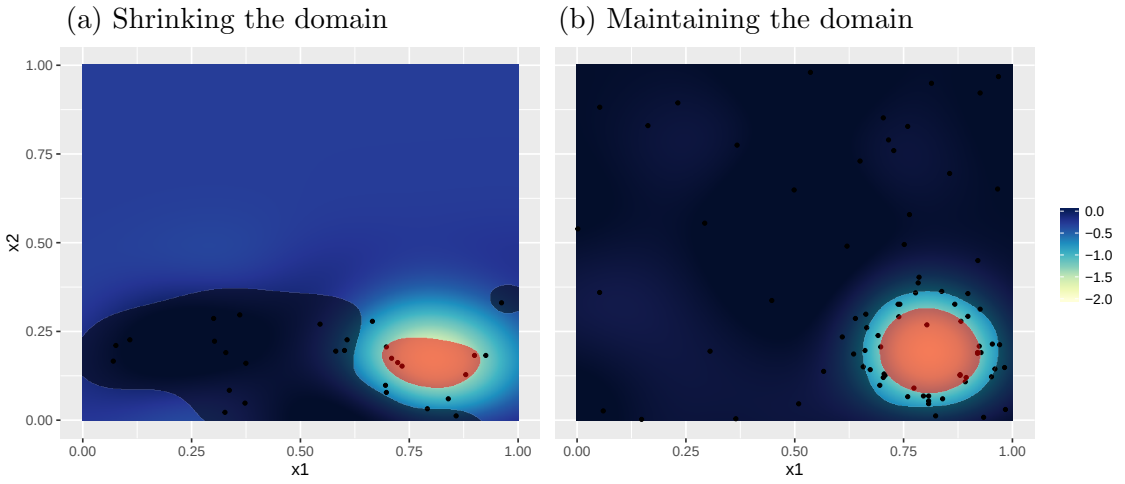


Figure 5.5: Predictive mean surfaces for the two emulators of the simulator in Equation (5.5) after three waves of history matching level set estimation. The emulator on the left shrank the domain between each wave and the emulator on the right did not. Both strategies allowed NROY and NRIY to flexibly change between waves. Also superimposed is a transparent black overlay representing the ruled-out space, and a transparent red overlay representing the ruled-in space.

With NROY and NRIY now flexible, the error for the strategy which shrinks the domain is 8.16%, the bias 8.16%, the error for the strategy which maintains a larger

domain is 7.80% and the bias is −7.80%.

The strategy which shrank the domain here is not fundamentally sound, as allowing NROY (and NRIY) to grow between waves should not be permitted, because the emulator is not valid for the ruled-out (or ruled-in) space. Despite minor numerical improvements, we can visually see that the results are also worse, as much less space is ruled-out. Additionally, this procedure could never converge: a new wave of data would rule-out much of the current NROY space, but the currently ruled-out space would become NROY again (because the simulations currently responsible for ruling-out this space would not be included in the emulator). This could continue indefinitely, with large parts of space being ruled-out, only to be un-ruled-out in the next wave because of data not being included in the emulator.

However, by maintaining a larger domain for the emulator, NROY (and NRIY) can be made flexible, resulting in a much improved emulator for this example, with the ruled-out space (and ruled-in space) very smoothly defined, with no obvious misclassifications. This is then a strong advantage for maintaining a larger domain - the emulators in later waves can rectify flaws in earlier wave emulator predictions.

For the other examples used in this section, numerical results are provided in Table 5.1 for the equivalent experiments, taking NROY (and NRIY) to be flexible.

	Error		Bias	
Multimodal	0.0204	(-0.0053)	-0.00202	(-0.00487)
Wavy	0.0114	(0.0000)	-0.00175	(-0.00233)
Wing weight	0.0113	(-0.0011)	-0.00175	(-0.00108)
Non-stationary	0.0214	(-0.0007)	0.0175	(+0.0047)

Table 5.1: Error and bias values for the history matching level set strategy which maintained a larger domain whilst allowing NROY and NRIY to be flexible. The values in brackets indicate the changes vs the non-flexible counterpart (difference in absolute values for the bias).

Numerically, allowing NROY/NRIY when a larger domain is maintained improves the error for all examples except the wavy 2D simulator where it is unchanged, and the bias is improved for all examples except the extreme non-stationary simulator where it is marginally worsened.

In general, one needs to be especially careful when shrinking the domain that

areas of input space are not wrongly excluded. The method of Xu et al. (2021) is one technique for helping with this, although the possibility of wrongly excluding a region will always remain.

Overall, it seems that shrinking the domain between waves is a beneficial strategy when the simulator exhibits significant non-stationary. On the other hand, maintaining a larger domain seems to be preferable when there are important global structures (for example when there are multiple distinct modes) and when a stationary Gaussian process assumption is reasonable (and does not appear to be particularly detrimental in the presence of weak non-stationarity). Additionally, maintaining a larger domain allows NROY/NRIY to be flexible, shrinking and growing between waves. This may not always have much impact, but it does protect against emulator flaws in earlier waves, and in some situations this could be essential.

With these results in mind, it may be better common practice to maintain a larger emulator domain than the NROY(/NRIY) space, and allow NROY(/NRIY) to be flexible within this larger domain. An open question remains as to how much larger the emulator domain should be to the NROY(/NRIY) space. Too much and a stationarity assumption across the input space can be untenable. Too little and early-wave simulations which are informative to the region of interest might be excluded, and regions of space which were wrongly ruled-out (or ruled-in) in earlier waves won't be corrected.

One strategy might involve using a different implausibility threshold to rule-out (and rule-in) inputs to the threshold used to decide the domain of the emulator. In other words, the threshold used to estimate the current NROY(/NRIY) space and choose new simulation runs could be different to the threshold used to decide which previously obtained simulation runs should be included in the next wave's emulator and for what regions NROY(/NRIY) will be recalculated in the next wave. For example, one might use 3 as the threshold for ruling-out (and -3 for ruling-in inputs), but 5 for determining where the next wave's emulator is valid[2].

[2]Note that, with a unimodal univariate output, 3 would imply *at least* 95% confidence that any ruled-out input should be ruled-out using Pukelsheim's rule (Pukelsheim, 1994), and 99.7% confidence assuming the output is normally distributed. 5 would imply *at least* 98% confidence that any ruled-out input should be ruled-out using the related Vysochanskij–Petunin inequality (Vysochanskij and Petunin, 1980), and more than 99.99% confidence assuming the output is normally distributed.

The problem with this heuristic strategy, is that while it could protect against the worst of the two competing strategies (fully shrinking the domain and fully maintaining the domain), it is unlikely to ever be the best strategy. Simulations with very high implausibility (say, above 5) can still be informative about the region of interest if the response is sufficiently stationary. Similarly, inputs with moderate implausibility (say, above 3 but below 5) can still belong to a non-physical regime that bears no resemblance to the region of interest. Preliminary investigation seems to confirm this, with this heuristic strategy never providing the best error rate on the examples used previously (and in fact, providing similar error rates for each example to whichever was the worse of the two competing strategies).

A different solution might involve using an emulator which can deal with non-stationarity (Gramacy and Lee (2008), Ba et al. (2012), Damianou and Lawrence (2013), Volodina and Williamson (2020)) and avoiding domain shrinkage altogether. However, precisely and accurately modelling extremely non-stationary processes can be time-consuming, difficult, and can be a wasteful use of a simulation budget.

Alternatively, if expert knowledge is available about the validity of the stationarity assumption, then this knowledge could be used to choose whether or not to shrink the domain (and by how much). Additionally, diagnostics for a given wave could be used to check for non-stationarity (as is done in Volodina and Williamson (2020)), which could help decide whether to shrink the domain at a given wave.

To conclude, whether or not the domain should be shrunk with history matching is not an easy decision, and there are potential negatives to choosing to shrink the domain by default. The nature of a specific problem and the extent of prior knowledge should play an important role in making this decision. In the previous chapter, the domain was shrunk at each wave and diagnostics revealed this strategy to have merit, because there was a substantially different simulator behaviour for small values of x_2 which were not in the region of interest. Regardless, it can be important to note that each strategy has its own flaws, and that the degree of additional simulation refocussing does not need to be tied to the degree of emulator domain shrinkage.

5.3 Comparison with Alternatives

History matching is not without competitors. For calibration, Bayesian calibration provides a holistic alternative (Kennedy and O'Hagan, 2001), although this can be

computationally taxing and can struggle with identifiability (Brynjarsdóttir and O'Hagan, 2014). Least-squares can also be used to estimate calibration parameters (Tuo et al., 2015), although this only provides a single final value (i.e. no probability distribution, or substitute like NROY space). For optimisation, Bayesian optimisation has proved to be widely popular, especially in the wider machine learning literature (Frazier, 2018). With Bayesian optimisation, a Gaussian process (or other emulator) is used to sequentially, one-at-a-time, select additional simulations according to some acquisition function, $\alpha(\mathbf{x})$, refitting the emulator each time, homing in on the optimum (Jones et al., 1998; Jalali et al., 2017). These sequential optimisation strategies can also be adapted to level set estimation problems (Picheny et al., 2010; Gotovos et al., 2013). The existence of alternatives thus prompts comparison with history matching.

Lyu et al. (2018) provides some sequential level set estimation techniques, developed for stochastic problems. These are implemented in the `hetGP` package, and thus are readily available for comparison[3]. For level set estimation, acquisition functions typically aim to simulate near where the boundary of the region of interest is believed to be, accounting for uncertainty. This is quite similar to history matching, where the former favours "optimal" choices for simulations, and the latter favours "plausible" choices for simulations.

In all comparisons that follow, we use toy simulators with normally distributed intrinsic noise, and we fit homoscedastic Gaussian process emulators (much like in the previous section). Level set estimation is not constrained to stochastic problems, but the deterministic setting can be considered as a special case of the noisy setting, and so a lot of the comparison could therefore be considered broadly relevant. Additionally, non-normal noise is also possible in practice (and for many examples, expected), but this setting is not the focus of this thesis, and non-normal stochastic emulation mostly remains an open research question (Fadikar et al., 2018; Baker et al., 2020b). However, the more general conclusions are likely to be generally applicable.

Note that the alternative schemes considered all assume the objective is to find the level set where the mean equals 0. Other targets are easily obtained by simply adding a constant to the simulations.

[3]These do not make up an exhaustive list of all sequential level set schemes (for example, Azzimonti et al. (2021)).

5.3.1 The (Considered) Alternatives

Perhaps the simplest sequential level set scheme from Lyu et al. (2018) is the Minimum Contour Uncertainty (MCU) method. This bears resemblance to the popular GP-UCB optimisation method (Srinivas et al., 2012), sequentially selecting a new simulation run according to which maximises the acquisition function in Equation (5.6):

$$\alpha_{MCU}(\mathbf{x}) = -|\hat{f}(\mathbf{x})| + \gamma s(\mathbf{x}), \tag{5.6}$$

where $\hat{f}$ is the predicted mean of the emulator, $s(\mathbf{x})$ is the epistemic emulator standard deviation (i.e. the standard deviation for the mean), and γ is a parameter (which can change throughout the sequential process). The $-|\hat{f}(\mathbf{x})|$ term favours points which are currently believed to be close to the boundary, the $s(\mathbf{x})$ term favours points which have a high degree of uncertainty, and the γ term weights the importance of each. By default, the `hetGP` package sets $\gamma = 2$, but Lyu et al. (2018) recommend using a value equal to the interquartile range of the mean divided by three times the average epistemic standard deviation, and so we use this value in subsequent examples.

Another criteria, cSUR, aims to reduce the local empirical error $\hat{E}(\mathbf{x})$, which is the probability of wrongly classifying $\mathbf{x}$ (which, for the normal case, equals $\Phi(-|\hat{f}(\mathbf{x})|/s(\mathbf{x}))$ where Φ is the cumulative distribution function of the standard normal). It does so by comparing the current value of $\hat{E}(\mathbf{x})$ to the (approximate) expected value if the considered point is run. As such, it aims to maximise the (expected) improvement in the local empirical error. Specifically, the acquisition function is:

$$\alpha_{cSUR}(\mathbf{x}) = \Phi\left(-\frac{|\hat{f}(\mathbf{x})|}{s(\mathbf{x})}\right) - \Phi\left(-\frac{|\hat{f}(\mathbf{x})|}{\hat{s}^{n+1}(\mathbf{x})|_{\mathbf{x}}}\right); \tag{5.7}$$

where $\hat{s}^{n+1}(\mathbf{x})$ is the one-step-ahead standard deviation, derived in Lyu et al. (2018) as:

$$\hat{s}^{n+1}(\mathbf{x}^*)|_{\mathbf{x}} = \sqrt{s^2(\mathbf{x}^*) - \frac{\nu((\mathbf{x}^*, \mathbf{x}))^2}{\sigma^2(\mathbf{x}) + s^2(\mathbf{x})}}, \tag{5.8}$$

where ν is the predicted *epistemic* covariance ($\mathcal{V} - \sigma^2$) and $\mathbf{x}$ is the considered point to be run (note that for α_{cSUR}, $\mathbf{x}^*$ and $\mathbf{x}$ are the same).

Another alternative, Integrated Contour Uncertainty (ICU), uses similar ideas but targets global improvements in the empirical error. Its acquisition function is

given by:

$$\alpha_{ICU}(\mathbf{x}) = -\sum_{\mathbf{x}_m \in X_m} \Phi\left(-\frac{|\hat{f}(\mathbf{x}_m)|}{\hat{s}^{n+1}(\mathbf{x}_m)|_{\mathbf{x}}}\right), \tag{5.9}$$

where X_m is a finite subset of the total input space.

And then as a final alternative, targeted mean squared error (tMSE), is a modified version of a criterion originally from Picheny et al. (2010):

$$\alpha_{tMSE}(\mathbf{x}) = \frac{s(\mathbf{x})^2}{\sqrt{2\pi(s(\mathbf{x})^2 + \sigma_\epsilon^2)}} exp\left(-\frac{1}{2}\left(\frac{\hat{f}(\mathbf{x})}{\sqrt{(s(\mathbf{x})^2 + \sigma_\epsilon^2}}\right)^2\right). \tag{5.10}$$

A larger σ_ϵ^2 decreases the weighting towards simulating near the boundary (and therefore increases the relative weighting towards uncertain regions), and therefore acts in a similar way to γ in MCU. For deterministic experiments, Picheny et al. (2010) recommend a value for σ_ϵ of 5% of the range of the mean, which we follow here, however Lyu et al. (2018) find that the value has little impact in stochastic settings.

In the following sections we will the compare history matching level set estimation to these sequential alternatives. For the sequential alternatives, we use the fast updating of the model fits (from `hetGP`) each time a new data point is simulated, refitting the model from scratch every 25 simulations (if the re-fit provides an improved likelihood over the fast update). When appropriate, the values of γ and σ_ϵ are also recalculated every 25 simulations, using the predicted values for 10000 new input points (chosen via a Latin hypercube). With ICU, X_m is taken as 1000 points chosen via a Latin hypercube. For history matching, we use three waves of data, do maintain the entire domain between waves, and allow NROY (and NRIY) to be flexible.

5.3.2 1D Example

As an initial example, consider the toy simulator in Equation (5.11), and assume we are aiming for values less than 0.4.

$$\begin{aligned} y &= \sin(0.5\pi x) + \epsilon; \\ \epsilon &\sim N\big(0, 0.1\big). \end{aligned} \tag{5.11}$$

Fitting an initial emulator to 10 simulations, and then adding 30 extra simulations (either via MCU or via 3 waves of history matching level set estimation (HMLSE)) we obtain the results in Figure 5.6:

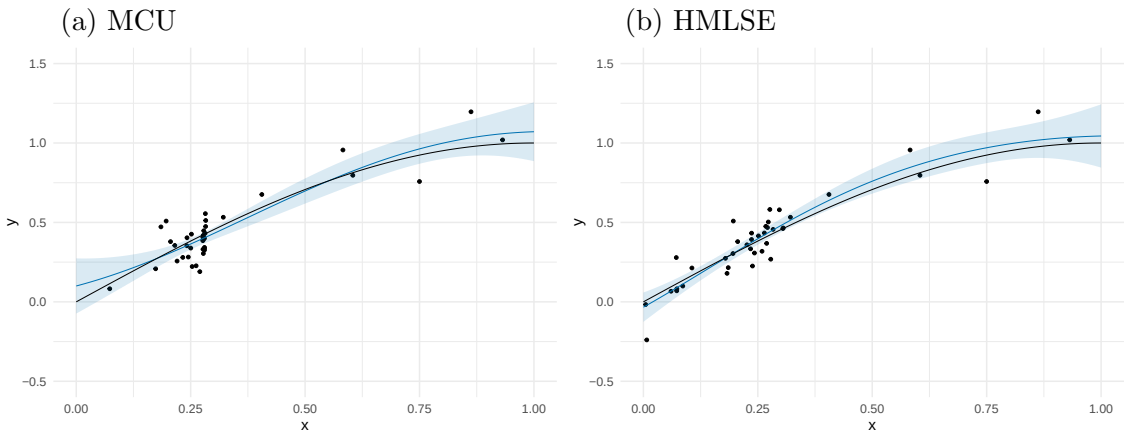


Figure 5.6: Mean predictions for the simulator in Equation (5.11). The uncertainty intervals are only for the epistemic uncertainty around the mean, and do not include the intrinsic noise estimates. The left plot shows results after obtaining 30 additional simulations via the MCU criteria, and the right plot shows results after 3 waves of history matching (each with 10 simulations).

From these plots, it's not clear if either method is preferable. MCU does seem to simulate more concentratedly around 0.4, although the history matching simulations are still fairly concentrated around 0.4. Numerically, MCU yields an error of 0.0155 (and bias of 0.0155), whereas HMLSE yields an error of 0.0103 (and a bias of -0.0103). This seems to give a small edge to HMLSE, although the overall evidence points to parity between the two methods (at least for as simple example as this one). We can also provide a baseline example to compare these results to. By adding the additional simulations via a simple maximin Latin hypercube (instead of via HMLSE or MCU), we get an error of 0.0137 and a bias of -0.0137. That this specific result is actually better than MCU implies that the differences between MCU and HMLSE are trivial, and that the used example is probably overly simplistic.

For completeness, repeating the experiment but for the other alternative methods: the cSUR error is 0.0100 (with a bias of -0.0100), the ICU error is 0.0002 (with a bias of -0.0002) and the tMSE error is 0.0187 (and a bias of 0.0187). With these, the results are all similar, except ICU which yields an improved result.

For a slightly more complex example, consider again the wavy simulator from Equation (5.3). Again assume we target values less than -0.25, using 20 initial simulations and 60 additional simulations (in 3 waves for HMLSE). Figure 5.7 presents these results.

The MCU error is 0.0134 (with a bias of -0.00984) and the HMLSE error is 0.0114

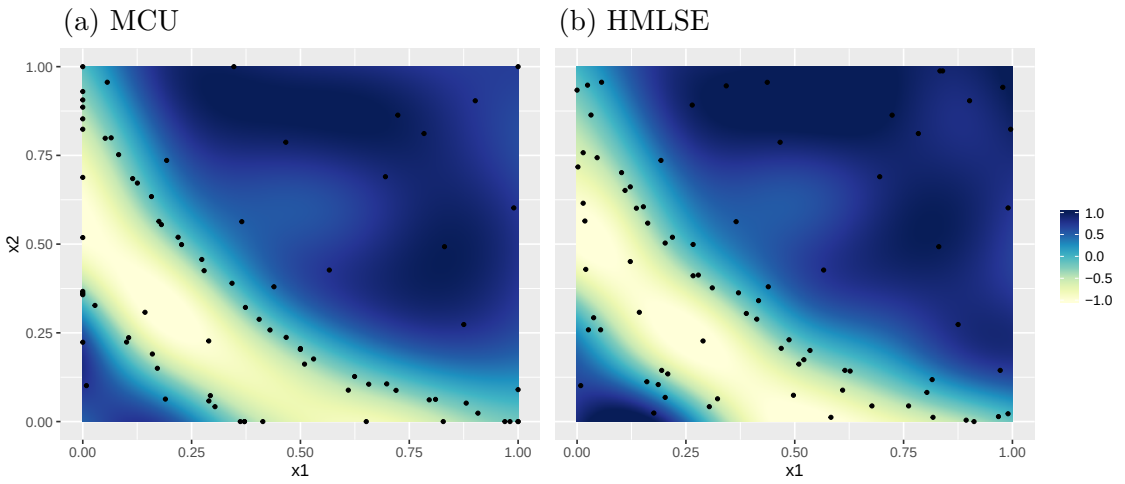


Figure 5.7: Mean predictions for the simulator in Equation (5.3). The left plot shows results after obtaining 60 additional simulations via the MCU criteria, and the right plot shows results after 3 waves of history matching (each with 20 simulations).

(with a bias of -0.00175). Even for this non-stationary problem, the two results are very similar. MCU again seems to simulate more orderly along the boundary, whereas HMLSE is more diffuse. Again for completeness, the cSUR error is 0.01225 (with a bias of -0.000936), the ICU error is 0.0141 (with a bias of -0.00755), the tMSE error is 0.0140 (with a bias of -0.00519). For a baseline, adding 60 additional simulations using a maximin Latin hypercube yields an error of 0.0194 and a bias of -0.00957. In this case, all the methods provide better results than the baseline. HMLSE is the best here, but by a small margin, suggesting HMLSE can compete with the alternatives.

5.3.3 6D Example

And then for a higher dimensional, more complex example, consider the toy simulator in Equation (5.12), which is a modified version of a toy simulator used in Lyu et al.

(2018):

$$y = -10\left(\sum_{i=1}^{4} C_i exp\big(-\sum_{j=1}^{6} a_{ji}(x_j - p_{ji})^2\big) - 0.1\right) + \epsilon$$

$$\epsilon \sim N\big(0, 0.25\big);$$

$$where \quad \mathbf{C} = (0.2, 0.22, 0.28, 0.3),$$

$$\mathbf{a} = \begin{pmatrix} 8.0 & 0.5 & 3.0 & 10.0 \\ 3.0 & 8.0 & 3.5 & 6.0 \\ 10.0 & 10.0 & 1.7 & 0.5 \\ 3.5 & 1.0 & 8.0 & 8.0 \\ 1.7 & 6.0 & 10.0 & 1.0 \\ 6.0 & 9.0 & 6.0 & 9.0 \end{pmatrix}, \text{ and } \quad \mathbf{p} = \frac{1}{10^4}\begin{pmatrix} 1312 & 2329 & 2348 & 4047 \\ 1696 & 4.135 & 1451 & 8828 \\ 5569 & 8307 & 3522 & 8732 \\ 124 & 3736 & 2883 & 5743 \\ 8283 & 1004 & 3047 & 1091 \\ 5886 & 9991 & 6650 & 381 \end{pmatrix}. \tag{5.12}$$

With an initial homoscedastic emulator fit to 60 simulations, adding 180 more via MCU (targeting the level set of 0) yields an error of 0.039708 (and a bias of -0.0159). Adding 180 simulations via 3 waves of history matching yields an error of 0.0508 (and a bias of -0.0101). The cSUR error is 0.0431 (with a bias of -0.0173), the ICU error is 0.0365 (with a bias of -0.00791), and the tMSE error is 0.0412 (with a bias of -0.00281). As a baseline, adding an additional 180 simulations as chosen via a maximin Latin hypercube yields an error of 0.0574 and a bias of -0.00600.

These results suggest that, for some examples, HMLSE can perform worse than the alternatives, at least for some more complex, noisier, problems. This difference is perhaps small, but it is measurably present. This provides some evidence that HMLSE can be less 'optimal' than the alternatives, which is perhaps to be expected, given that it does not fully optimise the criteria it uses to select simulations (implausibility). Additionally, in this case, the advantage provided by HMLSE over a standard design scheme is quite small, which could be due to the larger intrinsic noise in this example.

5.3.4 Low Signal-to-Noise Ratio Example

As further evidence that HMLSE can sometimes be worse than the fully sequential alternatives, consider the 1D toy simulator again from Equation (5.11) and compared previously. This time the intrinsic noise standard deviation is increased from 0.1 to 0.4. With this higher degree of intrinsic noise, the MCU error is 0.0556 (with a bias of -0.0556) and the HMLSE error is 0.116 (with a bias of -0.116). Additionally, the

cSUR error is 0.0432 (with a bias of 0.0432), the ICU error is 0.0128 (with a bias of 0.0128), and the tMSE error is 0.0475 (with a bias of 0.0475). The baseline error (using the design which adds 30 simulations via maximin Latin hypercube) is 0.111, and the baseline bias is -0.0882.

We can see here that, whilst they all do worse than before, HMLSE really struggles to compete with the alternatives, and does marginally worse than the baseline. This is because, with a large degree of intrinsic noise, there is a large degree of uncertainty, and so much of the input space remains NROY (and NRIY). This means that HMLSE cannot effectively refocus towards the solution, and as such it proposes essentially random designs. Optimal designs do not suffer from this as much, as even when the overall degree of uncertainty is large, and therefore the confidence that any specific simulation will be particularity valuable is low, some simulations will still be believed more valuable than others, and so some degree of refocussing can still occur.

This is likely to be an effect in calibration as well - problems with a high degree of noise will likely mitigate the refocussing capability of history matching, due to its reliance on completely ruling-out regions of space. Workarounds to this could be envisioned (for example, by weighting simulations based on their implausibility, or using different thresholds to rule-out points), but by default history matching seems worse than the alternatives in high noise scenarios, or other high uncertainty scenario.

In these situations, one can at least be aware that this issue may be present, as the degree of ruling-out can be reported a-priori to any additional simulations. If the amount of ruling-out is considered insufficiently large, one could consider alternatives or modifications. For example, with the 6D toy simulator above, the NROY/NRIY space after one wave was 65.59% and still 52.98% after 3 waves. With the noisier 1D simulator, NROY/NRIY was still 100% after 1 wave (and 25.45% after 3 waves). As such, were these to be practical examples, a practitioner could be made aware that a harsher design criteria might be useful, as the degree of refocusing is small.

This points to an open question in general, which is how to adapt history matching in general to particularly noisy simulators. One may wish to use a stricter ruling-out procedure (for example, ruling-out points where the implausibility is larger than 2, rather than 3) to obtain a greater degree of refocussing. The counterargument to this could be that history matching does not refocus much in noisy settings because it acknowledges the degree of uncertainty, and so a lesser degree of refocussing is the

sensible, conservative, strategy. Regardless, this problem was not an issue with the building model in Chapter 4, as the noise was sufficiently low, and a large amount of input space was ruled-out.

5.3.5 History Matching is Fast

The above examples suggest that history matching can sometimes make less optimal design choices than alternative strategies. The degree of this sub-optimality appears to be problem specific: with some problems history matching is competitive, and others it can be measurably worse (the latter due to insufficient reduction of NROY/NRIY space, which is at least known before additional simulation).

However, history matching does have advantages outside of pure simulation efficiency. Notably, history matching is a simple design scheme, which leads to it also being computationally cheap. Fully-sequential, optimal design strategies on the other hand involve optimisation over the input space and re-fitting or updating of the emulator for every new simulation; which can be more expensive. When the simulator itself is incredibly expensive to run, this difference can be trivial, but if the simulator is cheaper, this difference can be critical.

To provide some understanding of the difference in speeds, we return to the examples used above. For the 1D example, which is a low dimensional problem with few simulations, MCU took 2.2 seconds, cSUR took 1.55 seconds, ICU took 29.95 seconds and tMSE took 1.28 seconds (note how ICU takes much longer than the other alternatives). HMLSE took only 0.47 seconds, which is less than half the time of the fastest alternative. For this example, the difference is clearly trivial; the computational savings of history matching would not be important for such a simple example, but it is evidence that there is a computational difference.

For the 2D example, we have a similar story. The MCU time is 12.25 seconds, cSUR 16.82, ICU 320.51, tMSE 14.45 and HMLSE 4.11. Again HMLSE is the fastest by a wide margin, but again the times are all sufficiently small that this difference is not important. Were the simulator fast enough that these differences were important, one might question the value of an emulator at all.

However, the total time required for these design schemes, and the advantage history matching provides, increases as the dimension of the problem increases and the number of simulations increases.

Repeating the 2D experiment, but this time with 180 additional simulations rather than 60 (and an initial 60 simulations to fit the emulator, so 240 total simulations rather than 80), we get an MCU time of 87.64 seconds, a cSUR time of 56.64 seconds, an ICU time of 3039.47 seconds, a tMSE time of 69.05 seconds, and a HMLSE time of 18.76 seconds. Which is noticeably slower than previously. It is also noticeably slower, even when one considers the time spent per simulation. This is because Gaussian processes scale with $O(n^3)$ where n is the number of data points (or can be the number of unique data point locations in stochastic settings (Binois et al., 2018) and is the case in the `hetGP` package).

For the 6D experiment, the MCU time was 422.93 seconds, the cSUR time was 666.00 seconds, the ICU time was 12131 seconds, the tMSE time was 422.19 seconds, and the HMLSE time was 74.58 seconds. This experiment had the same number of data points as the repeated 2D experiment above, but in a higher dimension, and is again noticeably slower. We can see from these examples, that as the number of simulations increase and the number of dimensions increase, the computational cost of the sequential methods becomes increasingly more prohibitive. ICU is also revealed to scale especially poorly into more taxing problems, a result found before by Lyu et al. (2018).

Taking this further, and redoing the 6D experiment with 5 times the number of simulations (60 unique locations each replicated 5 times for the initial emulator, and then adding 900 simulations afterwards) we get an MCU time of 16436.47 seconds, a cSUR time of 30212.55 seconds, an ICU time of 547498.76 seconds, a tMSE time of 23546.55 seconds and a HMLSE time of 859.25 seconds.

For examples like this, if the simulations are sufficiently expensive (hours say), this difference still would not matter (except ICU, which is incredibly slow). However, for simulations which take a few minutes say, this difference can be the equivalent of several hundreds of additional simulations, and so the fully sequential methods become much less practical. As such, for reasonably cheap simulators, the computational simplicity of history matching can be a great advantage.

With the building model problem in Chapter 4 each simulation took roughly a minute. Additionally, although fewer unique simulations were used than the above example; more simulations were used in total, and the problem had a higher dimensionality. This suggests that HMLSE is indeed a practical methodology to use for the building model problem, despite potentially sub-optimal individual simulation

runs being used.

As an important note, although we found in Section 5.2 that shrinking the domain in history matching can sometimes be worse, especially since it prevents NROY (and NRIY) from being flexible, it does provide even faster analysis because it reduces the number of valid unique training points (and so $O(n^3)$ can be much less). For the example above, the HMLSE time was only 146.20 seconds when shrinking the domain, yet the error was not massively increased (0.0280 vs 0.0208, with a bias of 0.00533 vs 0.000862). As such, shrinking the domain may can be additionally valuable in high data scenarios, when a simulation budget is particularly large.

History matching also provides another benefit over fully sequential methods in that it is well suited to high performance computing, as the history matching procedure provides *batches* of recommended simulations. Batches can easily make use of parallelisation - allowing multiple simulations to be run at once, whereas fully sequential designs do not. So for especially slow simulations, the ability to leverage parallelised simulations can provide a potentially massive advantage, despite the individual simulation sub-optimality. However, as the uncertainty predictions in a Gaussian process depends only on the location of the training points, 'optimal' level set schemes could be modified to allow for batches of simulations. This adjustment has already been done for global emulator improvement designs (with the sequential procedure of Binois et al. (2019) extended to allow for batches by Zhang et al. (2020)).

5.4 Conclusion

In this chapter a discussion of some implementation choices regarding history matching and history matching level set estimation has been provided.

To begin with, we questioned the common strategy of shrinking the domain of the emulator in successive waves. Established practice recommends shrinking the domain to protect against non-stationarity. We find this can indeed be an advantage (and this advantage is likely larger than it seems in this discussion, especially in deterministic problems where non-stationarity can be more of a problem, or when the number of waves and degree of refocussing is higher). However, we also find it to be detrimental when there are important global structures which can be captured by an emulator. This disadvantage should not be too surprising, as aggressively shrinking the domain of the emulator is similar to throwing away data. Sadly, whether this

data improves an emulator or worsens an emulator is likely to be problem dependent, and so any decision here should require expert knowledge.

We also questioned the common strategy of permanently ruling-out (or ruling-in) regions of input space (which is essential when the domain of the emulator is shrunk to equal the NROY(/NRIY) space). Emulators can be incredibly flawed, and a flaw in an early wave emulator can still have large repercussions in later waves if ruling-out is permanent. This could be especially problematic if we eventually rule-out all parameter settings (the terminal case (Salter et al., 2019)), as we would not know if this was because of a *simulator* flaw, or an (early wave) *emulator* flaw.

Finally, we also compared history matching level set estimation to alternative sequential schemes. Here we found that, in some situations, the sequential, 'optimal', methods can perform better. On the other hand, in many situations history matching level set estimation performs competitively, and it is also substantially simpler and faster. This suggests that the choice of design scheme may also be problem dependent.

Chapter 6

Conclusion

If I have any overarching conclusion from the work done for this thesis, it is that emulators are fickle tools. Emulators can be powerful aids in extracting value from simulators, especially to ensure uncertainty is properly accounted for (whether this uncertainty is aleatoric or epistemic). The usually stated trade-off for this, is that the level of epistemic uncertainty increases, due to the emulator's added uncertainty. I would add to this that *error* is also increased, as a distinct separate entity to uncertainty. Whilst an emulator does provide uncertainty bounds for any prediction, these uncertainty bounds are not always sensible. In the deterministic setting, smoothness and stationarity are both reasonably strong assumptions which can lead to incorrect uncertainty estimates (but this can be dealt with if one is careful). Additionally, important regions of input space can be missed by training data, leading to poor parameter estimates and poor uncertainty estimates. In the stochastic setting, separating the signal from the noise can also be difficult, both conceptually and practically, leading to yet more risk of poor uncertainty estimates.

This is not a particularly novel conclusion, and the understanding that any model will ultimately be flawed despite its potential usefulness is already responsible for a widely overused quote[1]. Despite this, I believe it is a point worth underlining, and the use of any method should try to account for this.

Chapter 2 perhaps most directly tackles this idea, as we develop and investigate tools for checking the validity of a stochastic emulator. The general idea proposed is that component predictions (i.e. the mean and the variance) and assumptions (i.e. normality) of an emulator should be checked independently. This allows us

[1] "All models are wrong, but some are useful", often attributed to George Box. With this quotation, its overuse is added to further still.

to assess the epistemic (mean) uncertainty in a stochastic emulator independently from the aleatoric uncertainty estimates. This idea is perhaps less useful if only very basic predictions are required from the emulator, and in that case general diagnostic methods are likely to suffice. However, if a more extensive analysis is desired, then a more extensive set of diagnostics should be performed, using diagnostics specifically designed for stochastic emulators.

With chapter 3, we developed the technique of using deterministic runs to better inform about stochastic simulator analysis. We find deterministic simulations to be incredibly informative, even for stochastic simulators, especially if the total simulation budget is low. This idea is quite natural, as deterministic simulators exhibit a certain degree of sophistication that stochastic simulators lack. Having access to the entire truth at any given point is extremely powerful and informative. With stochastic simulators, the noisiness introduces a certain level of doubt, and an increased reliance on the emulator assumptions (smoothness, stationarity, normality, correlation structures), unless the simulation budget is exhaustive. Being able to use a deterministic simulator as a baseline for any conclusion can be a much stronger, and safer position than relying entirely on noisy simulations and incredibly flexible stochastic emulators. Of course, this relies on believing the deterministic simulator itself is informative, which may not always be true. This links quite closely to the fundamental assumption of the field of uncertainty quantification more generally - simulators need to be useful. If a simulator is not informative, then it is likely to be a better use of time and resources to obtain physical data and perform standard statistical analyses. Assuming simulators are useful (to some extent), then the coupling and averaging of multiple simulators is likely to (continue to) be an important avenue for future research.

In Chapter 4 we dealt with a more practical example: attempting to future-proof a building using stochastic simulations. This example is useful for emphasising a couple of important points. The first, which is hopefully reasonably obvious, is that simulators need not be incredibly computationally expensive to warrant sophisticated statistical approaches. Many simulators take what we might call 'non-trivial' amounts of time to run, but are not exactly slow either. This is almost a necessity with stochastic simulators, as our ability to draw any inference is seriously impacted by stochasticity, especially if we are unwilling to make key assumptions (for example, a linear trend and constant variance). Being able to extract similarly

precise and accurate inferences in a stochastic setting as we would in a deterministic setting can require a tenfold increase in the number of simulations. This exact ratio is yet unknown (and clearly dependent on the amount of noise), but it affects both training and validation. This increase in required simulations opens up many 'fast' simulators to the field of emulation, whilst also closing off more expensive simulators. This changing of acceptable simulator speed is likely to have many influences on which statistical methods are practical[2]. The second important point emphasised by Chapter 4 is that the history matching technique can be adapted to other experimental objectives. Of course, a question arises as to whether this is still history matching, but many of the important traits are maintained. For one, history matching is simple, which makes it easily applied to new situations, as shown by using it on both a stochastic GP emulator and a logistic GP classifier simultaneously. This simplicity also makes it fast (which we explored more in Chapter 5), which naturally makes it well suited to fast simulators. This simplicity also makes it easily understood, which is an under-appreciated attribute when dealing with simulators. Preferably, when working with complex systems that are simulated, the inclusion of practitioners (who are typically not statisticians) in the statistical analysis is preferable, and the simple 'ruling-out' procedure is quite intuitive (even if other implementation details may be more complex). Additionally, history matching is also (typically) quite conservative. This conservative nature may not always be optimal, as found in Chapter 5. However, I believe that being conservative can be a desirable trait once we acknowledge that an emulator (and indeed a simulator) can be flawed.

With all this in mind, there are still numerous open questions and issues. For one, we mostly deal with, and presume the existence of, a reasonably high signal-to-noise ratio. It should come as no surprise to find that higher noise ratios will make analysis in stochastic regimes even more difficult. How this affects the methods and ideas discussed in this thesis, and what other methods could be developed for high noise scenarios, remains an open issue. For example, we discuss individually assessing the mean of an emulator in Chapter 2, but in high noise ratios, the notion of an "accurate" mean perhaps becomes less important, less interesting, and less clear.

We also assume throughout that the simulator variability is normally distributed. For non-normal settings, there are several strategies to take. The first, is to assume

[2]It should also influence the complexity of *simulators* when stochasticity is used, but I have doubts as to whether modellers can accept this constraint.

a different distribution. For example, Xie and Chen (2017) and Lyu et al. (2018) deal with t-processes rather than a Gaussian process, and in Chapter 4 we used a Bernoulli distribution, with a GP modelling the latent probability. However, this requires knowledge of which distribution to use, and the implementation details can be more complex. Transforming the data to enforce some normality is also a potential route. For example, Henderson et al. (2009) use the logit transformation to deal with proportion data. However, this strategy requires knowledge of a suitable transformation. A third strategy is to avoid making any distributional assumption. For example, quantile Kriging interpolates over key sample quantiles (Fadikar et al., 2018; Plumlee and Tuo, 2014). Whilst this is a simple method, it requires more simulations than other methods. At the end of the day, this is the main trade-off that needs to be made in emulation (and statistical modelling more generally): The fewer assumptions one is willing to make; the more data one needs. Whether one is willing to make assumptions as strong as the normal assumption will be application dependent, and finding alternative strategies for when one is not willing to make that assumption remains an ongoing research problem. As an example, diagnostic tools for assessing whether a quantile Kriging emulator captures the underlying distribution and the various trends are captured well remains future work.

Other research directions also remain future work. For example, a rule-of-thumb for how many unique simulations and how many replicated simulations is currently lacking for stochastic emulators. Rules-of-thumb are exactly that, and they are not gospel, but having some preliminary estimate for new problems would be quite useful. In a similar way, the methodology outlined in Chapter 3 could also receive attention regarding experimental design. After a preliminary emulator has been built, whether to subsequently add more deterministic runs or more stochastic runs, and how to choose the relevant ratios and locations for them, is not yet known. Developing a technique for this could be interesting, but would likely require a more complex simulator than EnergyPlus.

For EnergyPlus, and the work done here with it, several potential extensions also exist. For one, the outlined strategy in Chapter 4 only acknowledged one type of 'aleatoric' uncertainty: weather uncertainty. Including occupancy uncertainty in the same way (by using another simulator to randomly sample potential occupancy profiles, as in Wate et al. (2020)) along with the weather, and any other potential 'aleatoric' uncertain properties, would be necessary for a complete uncertainty

quantification. Similarly, accounting for model discrepancy, and importantly, the construction gap (where a building is not built to the exact specifications designed) is important. The construction gap could be included in a history matching procedure in much the same way as model discrepancy, or alternatively it could be included as a kind of input uncertainty (whether done 'stochastically', by perturbing slightly any input to a simulator whenever ran, or by propagating input uncertainty through an emulator after its construction). Furthermore, the weather generator used in Chapter 4, UKCP09, is quite outdated. With the more recent climate predictions, UKCP18, no stochastic weather generator has been provided (it is not clear to me why this decision was made). As such, a different weather generator should be used, whether this is a standard, common, weather generator, or modified weather generator to again allow future climate weather generation.

Additionally, a more comprehensive building design procedure, whilst still following the strategy (or similar) outlined in Chapter 4, might wish to target more (or different) criteria. In Chapter 4 we required the overheating risk and heating energy usage to meet certain criteria, but other criteria can also be imagined (cooling energy usage, underheating, dampness, etc.), and may also want to be improved. Also, one might want to initially threshold some criteria (as done in Chapter 4), and then conditional on these criteria, optimise some other outputs. Doing so (and potentially developing additional methodology to do so) remains future work.

Overall, I think that stochastic emulation requires additional attention from statisticians, and building performance simulation could similarly adopt increased statistical expertise. There are many open questions and many gains that can be (reasonably) easily achieved. Additionally, I add my voice to the choir warning of overly-confident trust in complex statistical methods, and recommending increased checking and validation of such tools when used.

Appendix A

Further Details

A.1 Justification of Equation (2.5)

In Chapter 2, an equation was used for the predictions for the sample mean. What follows is a short justification for this equation.

Suppose we are predicting what the sample means would be if we simulated n times each for new points X^* ($\bar{y}(X^*) = \frac{1}{n}\sum_{i=1}^{n} y_i(X^*)$).

The standard noisy predictions for $\sum_{i=1}^{n} y_i(X^*)|\mathbf{y}$ (dropping the dependence on $\sigma^2(X^*), \sigma^2(X)$ for notational simplicity) given in Equation (1.3) can be rewritten as follows:

$$y_i(X^*) \mid \mathbf{y} = \mu(X^*) + \epsilon_i(X^*); \tag{A.1}$$

where

$$\mu(X^*) \sim N\big(\mathcal{M}(X^*), \mathcal{V}_\mu(X^*)\big),$$

with

$$\mathcal{M}(X^*) = m(X^*) + K(X^*, X)\big(K(X,X) + \sigma^2(X)I\big)^{-1}\big(\mathbf{y} - m(X)\big),$$

$$\mathcal{V}_\mu(X^*) = K(X^*, X^*) - K(X^*, X)\big(K(X,X) + \sigma^2(X)I\big)^{-1}K(X, X^*),$$

and

$$\epsilon_i(X^*) \sim N\big(0, \sigma^2(X^*)I\big).$$

Here, $\mathcal{M}(X^*)$ is as it is in Equation (1.3) and $\mathcal{V}_\mu(X^*)$ is simply the $\mathcal{V}(X^*)$ from Equation (1.3) minus $\sigma^2(X^*)$. This makes it clear how noisy Gaussian process predictions are made up of two independent processes: the (uncertain) simulator mean $\mu(X^*)$ and the intrinsic stochasticity $\epsilon_i(X^*)$, both of which are normally distributed.

As such, predicted values for the sample mean $\bar{y}(X^*)|\mathbf{y}$ are equal to $\frac{1}{n}\sum_{i=1}^{n}\mu(X^*)+\frac{1}{n}\sum_{i=1}^{n}\epsilon_i(X^*)$. The variance for the first term does not decrease as n increases, as no amount of additional predictive simulations will change the amount of *epistemic* uncertainty around the simulator's mean (remember that here, n is the number of prospective samples for a sample mean we want to predict for, it does not represent the number of simulations used to fit the emulator). The variance for the second term does decrease as n increases, as additional predictive simulations will decrease how noisy we expect a sample mean to be.

More formally, we know that $\sum_{i=1}^{n}\mu(X^*)$ is normal, with a mean $\mathcal{M}_{\bar{\mu}}(X^*)$ and a variance $\mathcal{V}_{\bar{\mu}}(X^*)$. We know that $\mathcal{M}_{\bar{\mu}} = n\mathcal{M}(X^*)$ as a standard result from the sum of normal distributions. To obtain $\mathcal{V}_{\bar{\mu}}(X^*)$, we use the standard result for normally distributed variables Z_i: $\text{var}(\sum_{i=1}^{n}Z_i) = \sum_{i=1}^{n}\sum_{j=1}^{n}\text{Cov}(Z_i, Z_j)$. With this, we have:

$$\mathcal{V}_{\bar{\mu}}(X^*) = \sum_{i=1}^{n}\sum_{j=1}^{n}\text{Cov}(\mu(X^*), \mu(X^*)) = \sum_{i=1}^{n}\sum_{j=1}^{n}\mathcal{V}_{\mu}(X^*) = n^2\mathcal{V}_{\mu}(X^*).$$

This means $\sum_{i=1}^{n}\mu(X^*) \sim N\big(n\mathcal{M}(X^*), n^2\mathcal{V}_{\mu}(X^*)\big)$, and therefore $\frac{1}{n}\sum_{i=1}^{n}\mu(X^*) \sim N\big(\mathcal{M}(X^*), \mathcal{V}_{\mu}(X^*)\big)$. Which is exactly the same as $\mu(X^*)$ itself, because the mean process prediction and uncertainty has no relation to the number of hypothetical simulations made at prediction points. The uncertainty around what the true mean is will not decrease even if we imagine predicting for a sample mean obtained with a million simulations.

What will change with additional simulation, is our belief on how noisy said sample mean will be. $\sum_{i=1}^{n}\epsilon_i(X^*) \sim N\big(0, n\sigma^2(X^*)I\big)$ as a standard result for the sum of independent normal distributions. And so $\frac{1}{n}\sum_{i=1}^{n}\epsilon_i(X^*) \sim N\big(0, \frac{1}{n}\sigma^2(X^*)I\big)$, which does decrease as the number of hypothetical prediction point simulation replicates n increases. And so for this example, with $\bar{y}(X^*)|\mathbf{y} \sim N\big(\mathcal{M}(X^*), \mathcal{V}_{\mu}(X^*) + \frac{1}{n}\sigma^2(X^*)I\big)$, as a standard result of the sum of normal distributions.

If the number of hypothetical replicates at each prediction point were to be different at each prediction point (with allocation r_i at each), then $\bar{y}(X^*)|\mathbf{y} \sim N\big(\mathcal{M}(X^*), \mathcal{V}_{\mu}(X^*) + R^*\sigma^2(X^*)I\big)$, where R^* is a diagonal matrix with entries r_i, with a similar argument. This then leads to the equation given in Equation (1.3), where $\mathcal{V}(X^*)$ there equals $\mathcal{V}_{\mu}(X^*) + R^*\sigma^2(X^*)I$.

A.2 Other Normality Tests

Many methods for testing for normality exist in the literature, and using the skewness and kurtosis to assess normality in Chapter 2 is not a new idea (for example, the Jarque-Bera test uses both to test for normality (Jarque and Bera, 1980)). For emulator diagnostics, the skewness and kurtosis diagnostics used in this thesis check across the input space (with several unique locations each with a few replicates), making use of the same data that can be used to assess the mean and variance (and in some cases, could be used to train the emulator). This trait can be combined with more standard normal tests; the skewness and kurtosis unexpectedness are not the only options. To show this, Figures A.1 and A.2 present two common normality test results applied to the gamma toy simulator and bimodal toy simulator using the same validation data as in Section 2.2.3. These tests are the Lilliefors test (Lilliefors, 1967), which is a modified version of the Kolmogorov–Smirnov test (Daniel et al. (1990); itself a check for the distance between the empirical distribution function and a known reference distribution function) for the normal case where the mean and variance are not known; and the Shapiro-Wilk test (Shapiro and Wilk, 1965).

We can see from Figure A.1 that both normality tests are able to identify non-normality with the gamma simulator: both have two p-values less than 0.05 (which is unlikely to happen by chance). Note that this shows more clearly the similarity between unexpectedness and p-values. The two are essentially the same (although unexpectedness is scaled differently), with the variable being checked determining the interpretation of extreme results. "Unexpectedness" rather than p-values were used in this thesis as a means of encouraging different practice. For example, with unexpectedness, the distribution of the unexpectedness values is important, and one must acknowledge that a degree of randomness is present. Using multiple unexpectedness values across input space to spot trends or repeating extreme values is valuable. For example, a unexpectedness value larger than 0.99 (or 0.95) does not guarantee that the emulator is flawed, as it could be present by chance. As such, multiple extreme values, and unlikely trends, should be used to learn about possible flaws. This same practice can be used with p-values, and is what we do with the tests in this appendix.

We can see from Figure A.2 that the Shapiro-Wilk test is clearly able to identify non-normality with the bimodal simulator, with 4 p-values less than 0.95. The

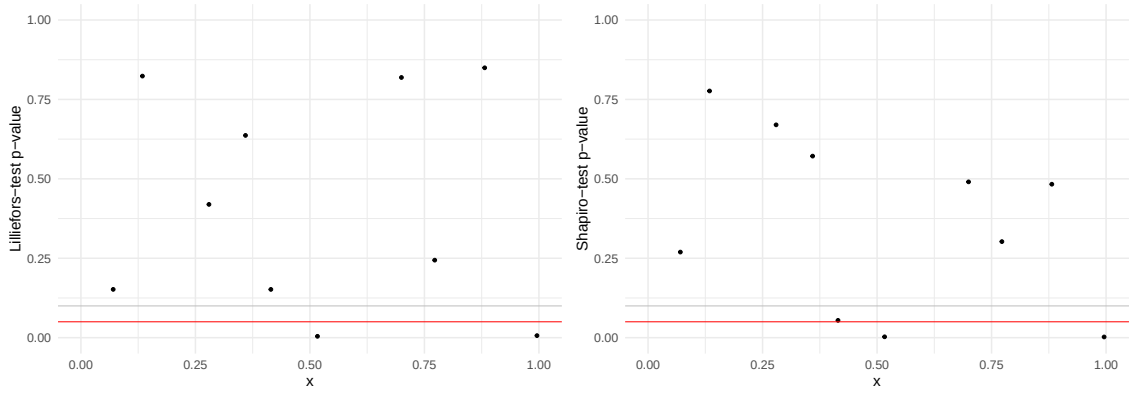


Figure A.1: Lilliefors test p-values and Shapiro-Wilk test p-values for the gamma distributed toy simulator. 0.05 is superimposed as a red horizontal line, as is 0.1 (as a light grey horizontal line).

Lilliefors test is less capable here, with only 2 less than 0.95, and one of those only just less than 0.05 (0.048). Additionally, these examples show how interpretability can be important, as these tests do not reveal anything about the distribution of the simulators, aside from that they are (probably) not normal. The skewness and kurtosis unexpectedness do better here, but not massively. For stochastic simulators, a useful diagnostic would be one which more robustly determines the type of non-normality, including bimodality, but this remains future work.

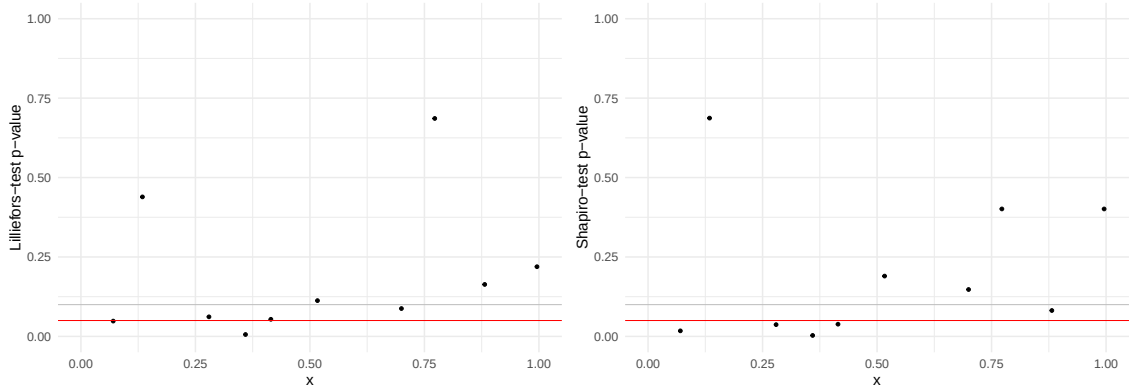


Figure A.2: Lilliefors test p-values and Shapiro-Wilk test p-values for the bimodal toy simulator. 0.05 is superimposed as a red horizontal line, as is 0.1 (as a light grey horizontal line).

A.3 Improving the Building Model Emulator

In Chapter 2 we emulated a hospital simulator, found issues using the developed diagnostics, and improved said emulator. Even after improving the emulator once, we did find some evidence of poor mean estimation for very small x_2. In this section, we will rectify this issue.

To do so, we simulate an additional 150 times, for building designs chosen via a maximin Latin hypercube, but constraining the x_2 values to be between 0 and 0.1. This should help capture any local pattern for small values of x_2. We also merge the validation data with the training data, and obtain a new validation data set in the same way as before (50 unique locations, chosen by a maximin Latin hyper cube, each run 4 times).

This results in the diagnostics given in Figures A.3 and A.4.

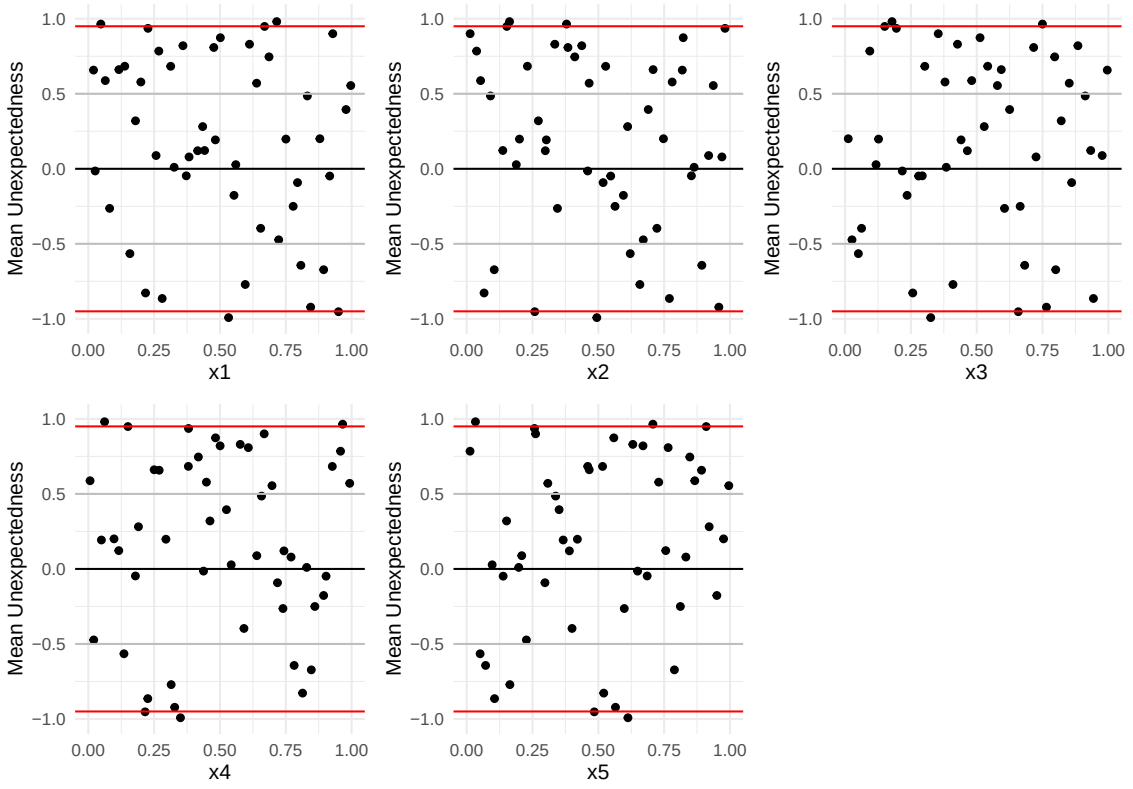


Figure A.3: Sample mean unexpectedness for the emulator of the building simulator using the combined 1100 training data points.

From these, we can see that there are no obvious problems with the emulator, and so we can conclude that this emulator is acceptable. There are 4 mean unexpectedness values with absolute value larger than 0.95, and no variance unexpectedness values with absolute value larger than 0.95. Four mean unexpectedness values is perhaps

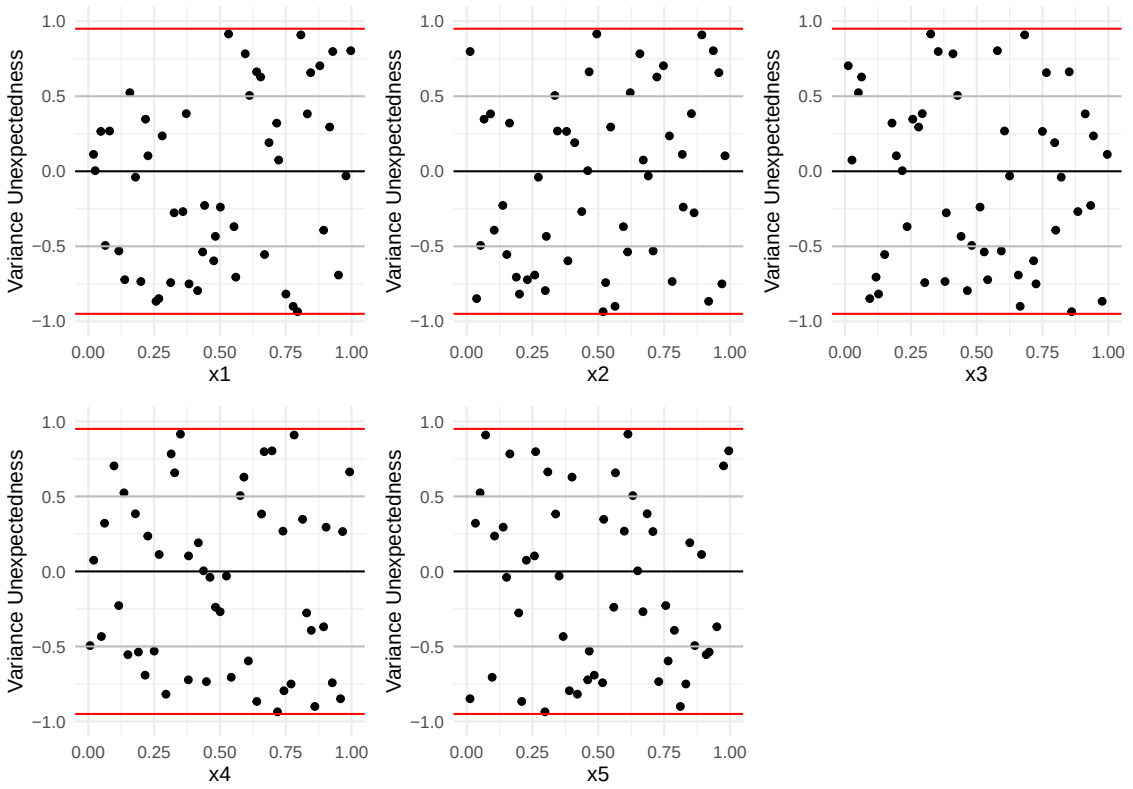


Figure A.4: Sample variance unexpectedness for the emulator of the building simulator using the combined 1100 training data points.

slightly higher than perfect, and one of these mean unexpectedness values is as small as -0.9918 (a different location than the previous extreme value, and this one is not as extreme), but overall this emulator is pretty much ideal now. However, 1100 simulations have been used to train this 5-dimensional emulator, which is a lot, and the cheaper 750 simulation emulator was likely sufficient for most practical purposes.

A.4 DetHetGP Lengthscale Priors

All the emulators fit in chapter 3 had an additional 'nugget variance' added for computational reasons (Neal, 1997), including the deterministic Gaussian processes. A value of 1e-4 was used.

For *DetHetGP*, the l_{D_i} prior was modified slightly. In practice $l_{D_i} = 0.01 + l^*_{D_i}$, and $l^*_{D_i}$ had a $Gamma(4, 4)$ prior. This constrains l_{D_i} to be greater than 0.01. Figure A.5 gives an example of what can happen in practice if l_{D_i} is not explicitly constrained to be reasonably larger than zero.

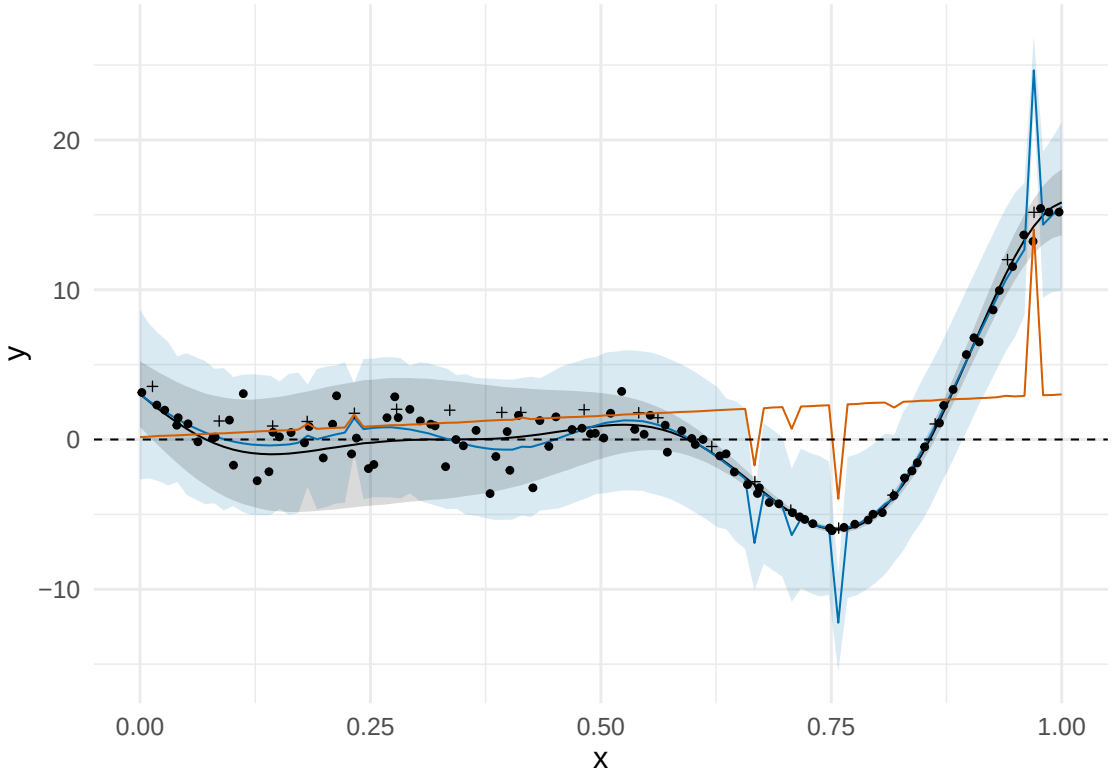


Figure A.5: Emulator predictions for the toy simulator from Binois et al. (2019), using the newly developed model that incorporates both stochastic and deterministic runs. The true mean and 95% interval are superimposed in black, and the emulator mean and 95% interval are in blue. The stochastic data points are circles, and the deterministic data points are plus symbols. The mean of the y_D component is in orange.

l_{D_i} has an estimated value of 0.000815 here, which is very small. This results in the deterministic Gaussian process (plotted in orange) being approximately a straight line, with steep jumps towards the observed deterministic points (which is an established problem discussed by Andrianakis and Challenor (2012), caused by

the inclusion of a nugget variance). This leads to the final stochastic emulator also having steep unnecessary jumps, and a poor overall fit, because there is not enough stochastic data for the δ component to smooth out the issues.

This issue could be prevented by decreasing the value of the nugget variance for the deterministic Gaussian process, but that can lead to computational errors. An alternative solution is the one implemented, fixing l_{D_i} to be sufficiently larger than zero, which seems to mostly avoid the problem.

A.5 Additional Building Model Diagnostics

In Chapter 4 validation was concerned mostly with checking the mean predictions of the energy usage (and the overheating probability), as this was the quantity of interest. It can still be informative to check the variances, as well as the normality assumption.

Figures A.6 to A.8 present the variance unexpectedness values (using the replicated validation data set).

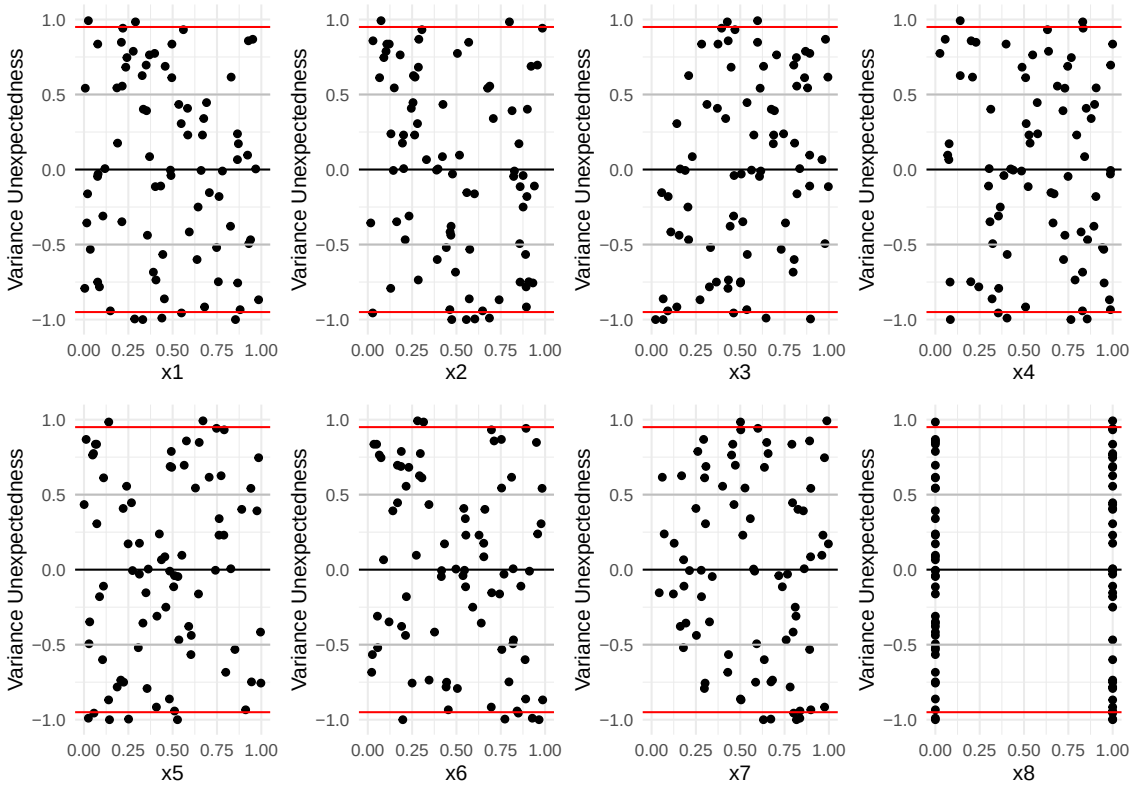


Figure A.6: 80 sample variance unexpectedness values for the wave 1 energy usage emulator.

The wave 2 results do not reveal any issues with the intrinsic variance. Wave 1 likely has issues with underestimated intrinsic variance, with 5 unexpectedness values less than -0.95, 2 of which are less than -0.995. Wave 3 also has some issue with overestimated intrinsic variance, with 59 out of 80 unexpectedness values above 0, 6 with value greater than 0.95, and 2 larger than 0.995. Since we are only interested in the mean in this example, these are not particularly problematic, although a well captured intrinsic variance process is obviously preferable.

For normality, we can use QQ plots to check the conditional normality assumption. Figures A.9 presents the QQ plots for each wave using the regular standardised errors

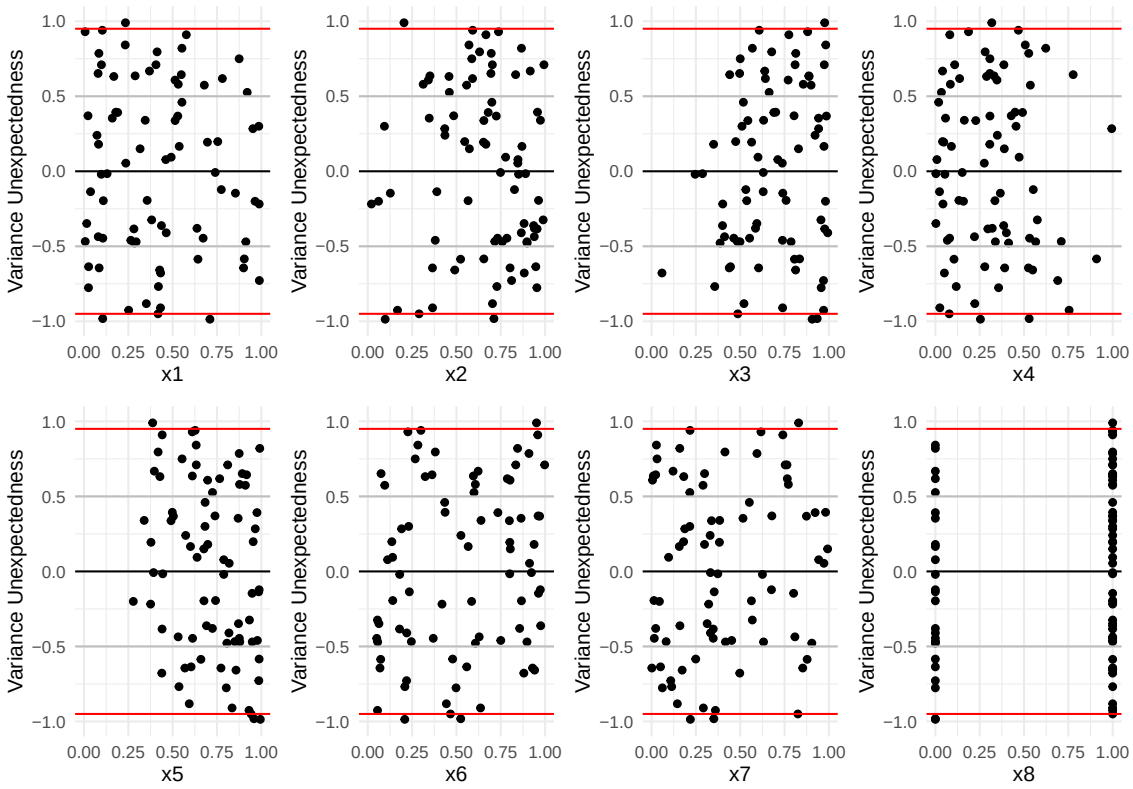


Figure A.7: 80 sample variance unexpectedness values for the wave 2 energy usage emulator.

using the non replicated validation data.

The wave 1 QQ plot reveals issues, but these issues were known before and are due to the poor emulator predictions for certain implausible points, rather than non-normality. The other QQ plots are fine. To check for normality in isolation, Figure A.10 plots the skewness unexpectedness values for wave 1 and Figure A.11 plots the kurtosis unexpectedness (both using the replicated validation data set, and including tolerance to error as outlined in Section 2.2.3).

These suggest that there are no non-normality issues, with 7 skewness unexpectedness values with absolute value more than 0.95 (which is more than ideal, but not excessively so), and 7 kurtosis unexpectedness values with absolute value more than 0.95. For wave 2, there are 3 skewness unexpectedness values and 4 kurtosis values with absolute value larger than 0.95. For wave 3, this is 3 and 0 respectively.

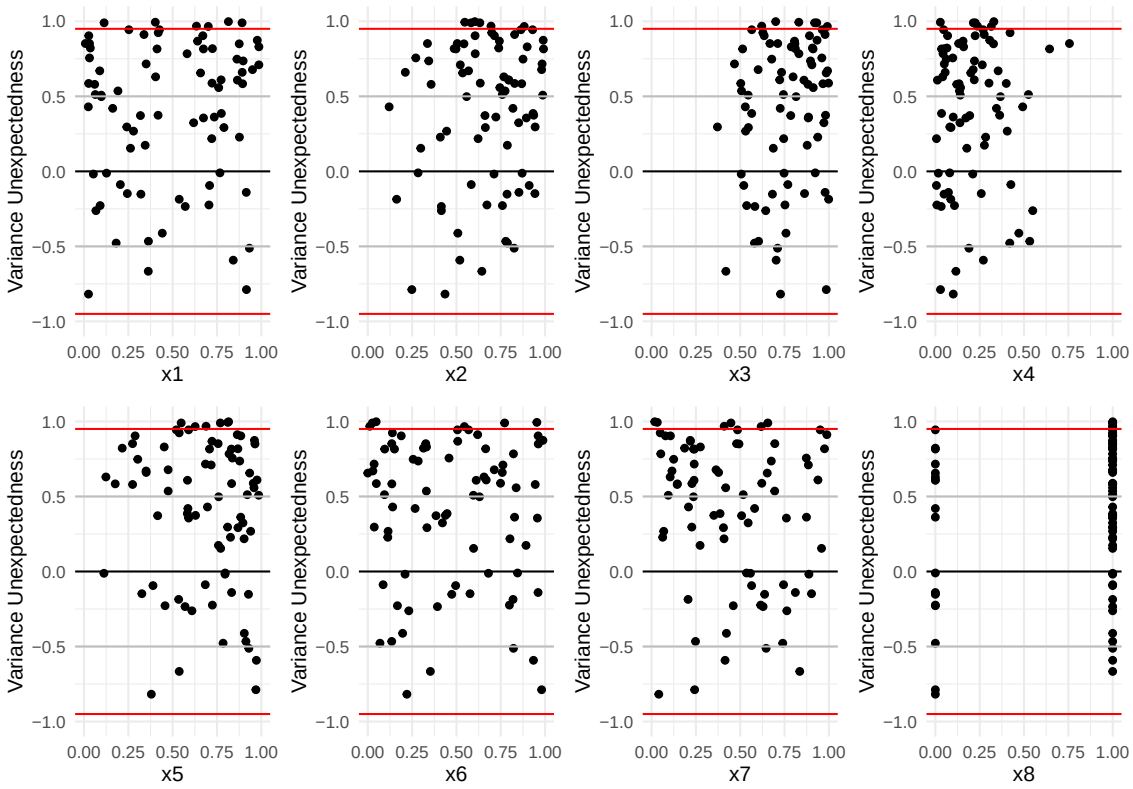


Figure A.8: 80 sample variance unexpectedness values for the wave 3 energy usage emulator.

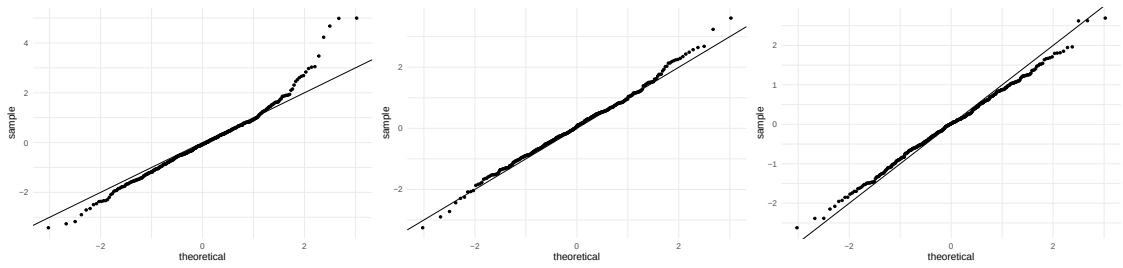


Figure A.9: The QQ plots for the emulators of the building model. Left is for the emulator at wave 1, middle for wave 2, and right for wave 3. All use the non-replicated validation data.

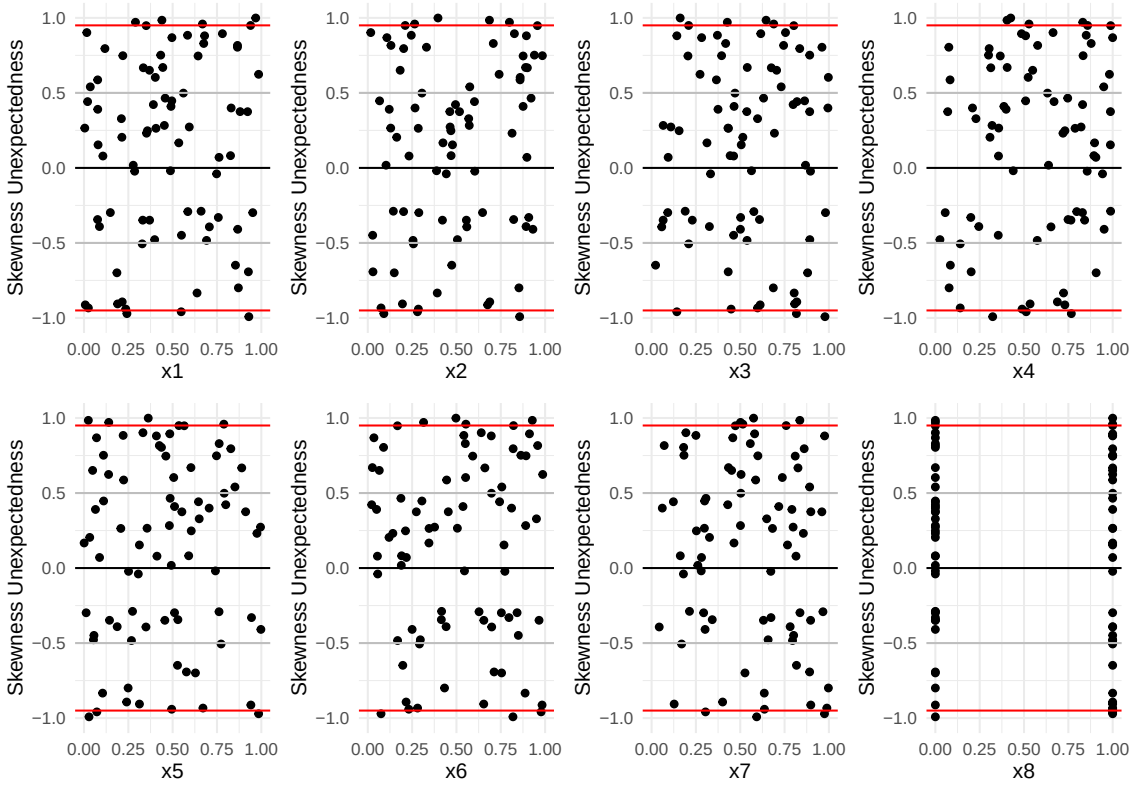


Figure A.10: 80 skewness unexpectedness values for the wave 1 energy usage emulator.

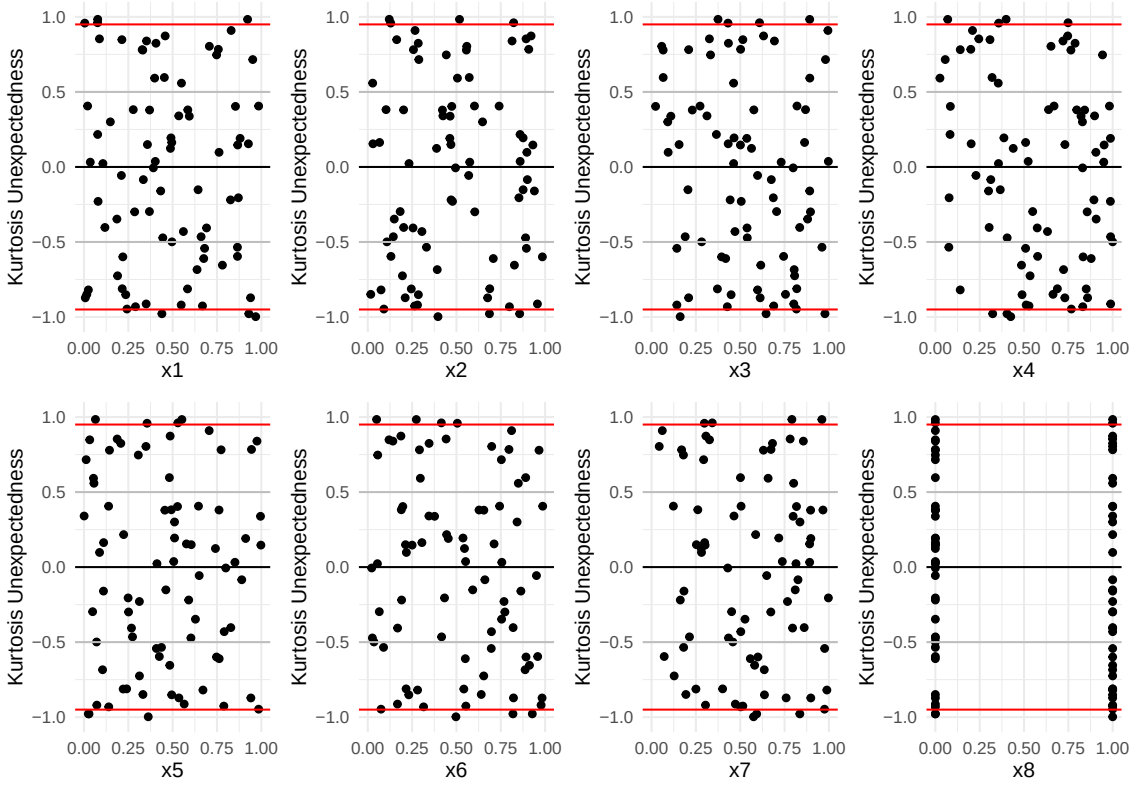


Figure A.11: 80 kurtosis unexpectedness values for the wave 1 energy usage emulator.

A.6 Maintaining the Domain for the Ruled-out Space

In Section 5.2.1 we discussed the advantages and disadvantages of shrinking the emulator domain between waves. We found, in general, that maintain a larger domain can be beneficial, except in the presence of extreme non-stationarity. For level set estimation, there is also the option of only shrinking the domain to exclude the ruled-out space, but maintaining the ruled-in space. This might be done to protect against non-stationarity while ensuring the emulator predictions remain valid in the interesting ruled-in space. We will see in the revisited examples here, that this strategy mostly acts as a compromise between the two other strategies. All examples here follow the exact same simulator and data setups as those in Section 5.2.1. The strategy which maintains the entire domain will be referred to as the 'maintain-domain' strategy, the strategy which shrinks the domain will be referred to as the 'shrink-domain' strategy, and the strategy which only shrinks the domain for the ruled-out space but maintains the ruled-in space will be referred to as the 'maintain-RI' strategy.

Revisiting the 'extremely' non-stationary simulator from Equation (5.1), we obtain the results in Figure A.12

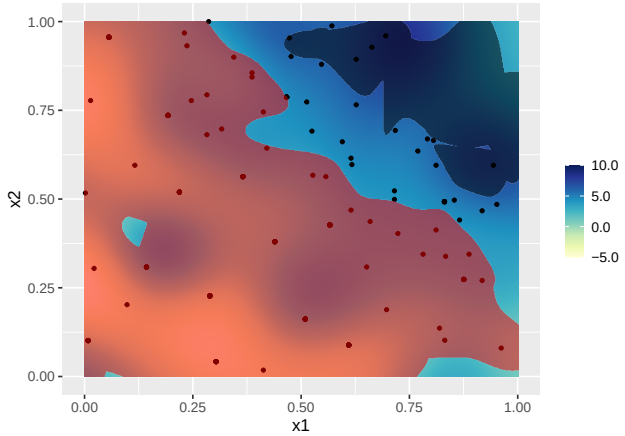


Figure A.12: Predictive mean surfaces for the emulator of the simulator in Equation (5.1) after three waves of level set estimation history matching. The domain was shrunk to exclude the ruled-out space in each wave. Also superimposed is a transparent black overlay representing the ruled-out space, and a transparent red overlay representing the ruled-in space.

These results look very similar to those from the maintain-domain strategy (right plot in Figure 5.1). This is likely because the ruled-out space is quite small, and the complex part of the simulator is in the ruled-in space. As such, the simulator surface is not much easier to emulate in the smaller domain (if at all). Numerically, the maintain-RI strategy error is 2.54% (worse than both the shrink-domain strategy and the maintain-domain strategy) and the bias is 0.116% (better than both).

However, correct classification of each region is unlikely to be the main motivation for following the maintain-RI strategy. This strategy will likely be taken if accurate predictions in ruled-in space are of primary interest. As such, a better metric for comparison here might be the root mean squared error (RMSE). With access to the truth, what we mean by this is the square root of the mean squared difference between the emulator's mean and the true mean (rather than the difference between some set of simulations and the emulator's mean). We also constrain this RMSE to only the locations where the true simulator mean is in the level set of interest (which we shall call the LS-RMSE from now on). For this example, the LS-RMSE is 1.87, which is substantially better than the shrink-domain strategy's LS-RMSE of 2.93 (as we would expect, because the shrink-domain emulator is not valid in the ruled-out space) and slightly better than the maintain-domain strategy's LS-RMSE of 1.95. So overall, for this simulator (and this setup) it seems that shrinking the domain to exclude the ruled-out space, but maintaining the domain over the ruled-in space can be beneficial (if one is primarily interested in the ruled-in space.

Revisiting the multimodal simulator from Equation (5.2), we obtain the results in Figure A.13

These results are similar to the results obtained by the shrink-domain strategy (left plot in Figure 5.2). Here, the important global structures (the multimodality) are still not captured as the relevant information was contained in the simulations no longer in the emulator's domain. The ruled-in space is reasonably small, and so the extra information contained within does not help significantly. We can see this in the numerical results, where the classification error is 9.11% (slightly less than the shrink-domain strategy, but much more than the maintain-domain strategy) and the bias is -3.36% (better than the shrink-domain strategy, but worse than the maintain-domain strategy). For this example, the LS-RMSE is 0.318 (better than the shrink-domain strategy's LS-RMSE of 0.436 but worse than the maintain-domain strategy's LS-RMSE of 0.083). For this simulator it seems that maintaining the

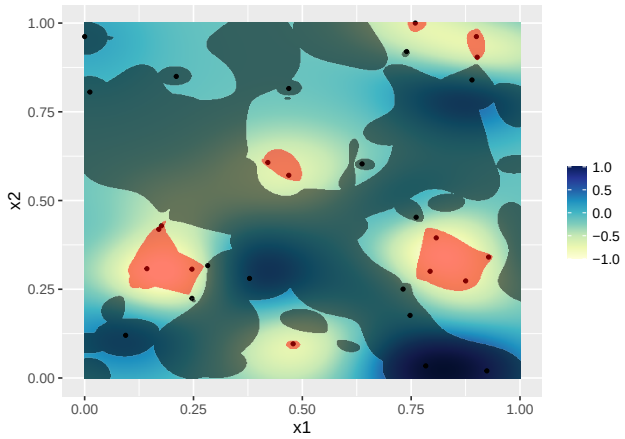


Figure A.13: Predictive mean surfaces for the emulator of the simulator in Equation (5.2) after three waves of level set estimation history matching. The domain was shrunk to exclude the ruled-out space in each wave. Also superimposed is a transparent black overlay representing the ruled-out space, and a transparent red overlay representing the ruled-in space.

entire domain is important, as there are important global structures which need to be captured and a stationary Gaussian process assumption is reasonable. If one does shrink the domain here, maintaining at least the ruled-in space is beneficial.

Revisiting the 'wavy' simulator from Equation (5.3), we obtain the results in Figure A.14

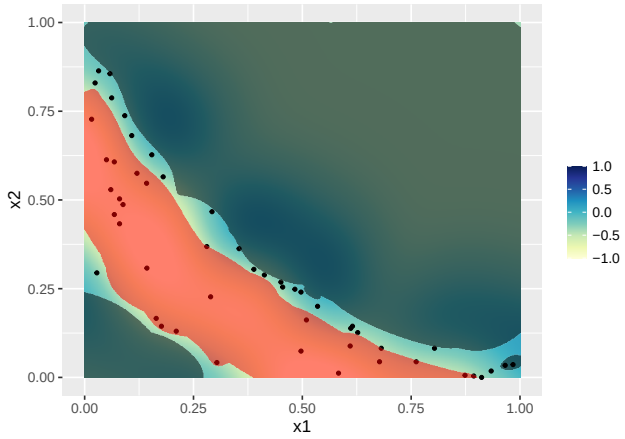


Figure A.14: Predictive mean surfaces for the emulator of the simulator in Equation (5.3) after three waves of level set estimation history matching. The domain was shrunk to exclude the ruled-out space in each wave. Also superimposed is a transparent black overlay representing the ruled-out space, and a transparent red overlay representing the ruled-in space.

These results look similar to both of those from before, and so numerical results are essential here. The error is 2.49% (which is better than the shrink-domain strategy but worse than the maintain-domain strategy) and the bias is 1.05% (which is worse than both). The LS-RMSE is 0.112, compared to the shrink-domain strategy's LS-RMSE of 0.535 and the maintain-domain strategy's LS-RMSE of 0.183. The maintain-RI strategy here has the best mean predictive performance in the region of interest, which is somewhat surprising. It is likely that shrinking the domain to exclude the ruled-out space but maintaining the ruled-in space helps to mitigate some problems from non-stationarity whilst still providing a valid emulator for the region of primary interest. That the classification error is worse does suggest that the loss of some global structures is detrimental in some ways, but if the main objective is to obtain good predictions in the region of interest (rather than say, correctly classifying points as in or out of this region), then the maintain-RI strategy here is a good compromise.

For the 10D wing weight simulator, we obtain an error for the maintain-RI strategy of 1.28% (better than the shrink-domain strategy but marginally worse than the maintain-domain strategy), a bias of 0.423% (worse than both). We also obtain an LS-RMSE of 3.80, which is better than the LS-RMSE for the shrink-domain strategy (19.64) and marginally worse than the LS-RMSE for the maintain-domain strategy (3.78). For this example, there does not seem to be much difference between maintaining the entire domain and only shrinking the domain to exclude the ruled-out space. This is probably because the ruled-out space is quite small, taking up only 15.0% of the total space for the maintain-RI strategy in the first wave, and only 19.0% after the final wave.

Overall, there are not really many general takeaways for the maintain-RI strategy. Whether this strategy is better or worse than the shrink-domain strategy or the maintain-domain strategy seems to also depend on the specific simulator, and also the size of the (true) ruled-in region compared to the (true) ruled-out region. The strategy does seem to sit, broadly, somewhere between the other two strategies, as one might expect. For the examples above, the results appear to be closer to the worse of the two, regardless of which one that is. With the 'extremely' non-stationary (where shrinking the domain was better), the maintain-RI strategy was more similar to the maintain-domain strategy. With the multimodal simulator (where maintaining the entire domain was better), the maintain-RI strategy was more similar to the

shrink-domain strategy. This could be because, the maintain-RI is more similar to the shrink-domain strategy if the ruled-out space is large, and more similar to the maintain-domain strategy if the ruled-out space is small. With the other two examples, differences between the strategies were less extreme, which reflects the smaller differences between the shrink-domain and maintain-domain strategies.

Regardless, if one wants to shrink the domain because of potential non-stationarity, but also still wants to obtain valid predictions in the ruled-in space, the maintain-RI is the only option out of the three. If this setup is then chosen, and a flexible NROY/NRIY is desired, it is important to note that only the ruled-in space can be flexible, not the ruled-out space (for the same reasons why a flexible ruled-out space doesn't work for the shrink-domain strategy).

A.7 Shrinking the Domain and Stochasticity

In Section 5.3.1, history matching level set estimation was compared to other sequential level set schemes. The example for the simulator in Equation (5.11) can also be used to demonstrate an important problem that can arise with history matching in stochastic problems if the domain is shrunk between waves.

Figure A.15 presents the resulting emulator after performing 3 waves of history matching as was done for Figure 5.6, but this time shrinking the domain to exclude the ruled-out and ruled-in space between each wave.

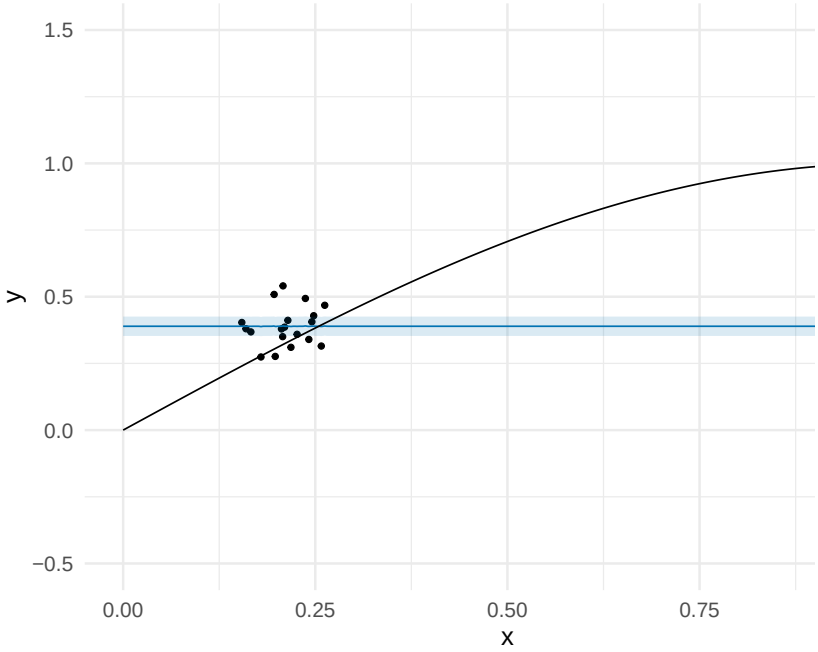


Figure A.15: Mean predictions for the simulator in Equation (5.11). The uncertainty intervals are only for the epistemic uncertainty around the mean, and do not include the intrinsic noise estimates. These are the results after 3 waves of history matching (each with 10 simulations), shrinking the emulator's domain each wave.

The result here is a very poor emulator, which does not pick up the simulator's mean. It instead assumes all variation in the simulations are due entirely to intrinsic noise. This is a similar (albeit worse) example to the introductory toy examples used in Chapters 2 and 3.

By shrinking the domain, important information contained in the ruled-out and ruled-in simulations, needed to separate the signal from the noise, is lost and the resulting emulator is worse for it. With this result, maintaining a larger domain becomes increasingly preferable for history matching of stochastic problems.

Continuing to shrink the domain is not completely discouraged, and as we found in Section 5.2, it can still be useful in stochastic problems if extreme non-stationarity is believed to be a problem; but additional care is needed to ensure the resulting emulators are still able to identify a (correct) signal from the noisy simulations in a constrained space.

Appendix B

Code

In this appendix, we provide some code which can be used to implement some of the techniques described in this thesis.

B.1 Diagnostics Code

Below is an R script which implements the stochastic diagnostics developed in Chapter 2. The script is a fully self-contained example: providing a toy simulator (the same as in Section 2.1, obtaining training simulations, fitting an emulator (using the `hetGP` package), obtaining validation data, and then applying the diagnostic methods.

```
library(hetGP)
library(ggplot2)
library(lhs)
library(dplyr)
library(MASS)
library(e1071)
library(sn)
library(gnorm)
set.seed(11111)

#Create a fake stochastic simulator
ToySim <- function(x){
  y <-   sin(16*x[,1]) + cos(24*x[,1]) + 8*x[,1] +
    rnorm(length(x[,1]),0,0.1+0.9*x[,1])
}

#and we know what the truth is:
TrueMean <- function(x){
  y <- sin(16*x[,1]) + cos(24*x[,1]) + 8*x[,1]
}
TrueVar <- function(x){
```

```
  y <- (0.1 + 0.9*x[,1])^2
}

#get training data set
X_unique <- maximinLHS(20, 1) #unique points
#function to replicate x:
rep.row<-function(x,n){ matrix(rep(x,each=n),ncol=ncol(x))}
X <- rep.row(X_unique,20) #replicate: 20 rep for good emul, 1 for bad
y <- ToySim(X)

#fit hetgp
hetgpmod <- mleHetGP(X=X, Z=y)

#get predictions
X_pred <- as.matrix(seq(0,1,len = 100))
pred <- predict(hetgpmod, X_pred)

predmu <-  pred$mean
predvar <- pred$sd2 + pred$nugs

#plot
ggplot()+
  geom_point(aes(x=X, y=y))+
  geom_line(aes(x=X_pred, y=pred$mean), color = "#0072B2")+
  geom_ribbon(aes(x=X_pred, ymin =pred$mean- 2*sqrt(predvar),
                  ymax = pred$mean + 2*sqrt(predvar)),
              alpha = 0.15, fill = "#0072B2")+
  geom_line(aes(x=X_pred, y=TrueMean(X_pred)), color = "black")+
  geom_ribbon(aes(x=X_pred, ymin =TrueMean(X_pred) - 2*sqrt(TrueVar(X_pred)),
                  ymax = TrueMean(X_pred) + 2*sqrt(TrueVar(X_pred))),
              alpha = 0.15)+
  theme_minimal(base_size = 16) +
  xlab("x")+
  ylab("y")



##############################################################################
#Stochastic Validation
##############################################################################
set.seed(22222)

#now get some validation data
X_valid_unique <-  maximinLHS(10, 1) #unique points
X_valid <- rep.row(X_valid_unique,5) #replicate them
y_valid <- ToySim(X_valid) #simulate

#use emulator to predict for validation points
pred_valid <- predict(hetgpmod, X_valid, xprime = X_valid, noise.var = TRUE)

#convert to dataframe
valid_df <- data.frame(X_valid, y_valid, GPmu = pred_valid$mean,
```

```
                        GPvar =pred_valid$nugs, GPuncert = pred_valid$sd2)
#make first name X1
names(valid_df)[1] = "X1"
#get sample means and sample variances, sample skewness and sample kurtosis,
#and save the emulator mean and variance predictions for each unique point:
valid_df = valid_df %>%
  group_by_at(vars(names(valid_df)[1:length(X_valid[1,])])) %>%
  summarise(mean = mean(y_valid), var = var(y_valid), rep = n(),
            GPmu = mean(GPmu), GPvar = mean(GPvar), GPuncert = mean(GPuncert),
            skew = skewness(y_valid), kurt = kurtosis(y_valid))


##############################################################################
#Mean
##############################################################################

#for the means, get the unexpectedness:
MeantUs <- rep(999999, len = nrow(X_valid_unique))
for (i in 1:nrow(X_valid_unique)){
  #get the simulated distribution for the sample mean
  GP_samplemean_dist = c()
  for (m in 1:10000){ #(10000 is number used for monte carlo U estimation)
    #get sample of sample mean:
    GP_samplemean = rt(1, valid_df$rep[i]-1)*
      sqrt(valid_df$var[i])/sqrt(valid_df$rep[i]) +
      rnorm(1, valid_df$GPmu[i], sqrt(valid_df$GPuncert[i]))
    GP_samplemean_dist = append(GP_samplemean_dist, GP_samplemean) #save sample
  }
  #get unexpectedness:
  MeantU <- 2* (0.5-ecdf(GP_samplemean_dist)(valid_df$mean[i]))
  MeantUs[i] <- MeantU
}

#plot mean t unexpectedness
for (i in 1:length(X_valid[1,])){
  print(ggplot()+
          geom_point(aes(x= pull(valid_df, i), y=MeantUs))+
          geom_hline(yintercept = 0.95, color = "red")+
          geom_hline(yintercept = -0.95, color = "red")+
          geom_hline(yintercept = 0, color = "black")+
          geom_hline(yintercept = -0.5, color = "grey")+
          geom_hline(yintercept = 0.5, color = "grey")+
          theme_minimal(base_size = 16) +
          ylim(-1,1)+
          xlim(0,1)+
          ylab("Mean_Unexpectedness")+
          xlab("x"))
}


##############################################################################
```

```
#Variance
################################################################################

#for the variances, get the unexpectedness:
VarUs <- rep(999999, len = nrow(X_valid_unique))
for (i in 1:nrow(X_valid_unique)){
  #get the simulated distribution for the sample variance
  GP_samplevar_dist = c()
  for (m in 1:10000){
    #get a sample for the predicted sd (including tolerance to error):
    GPvar = (sqrt(valid_df$GPvar[i])*(runif(1, 0.8, 1.2)))^2
    #get sample of sample variance:
    GP_samplevar = rchisq(1, valid_df$rep[i]-1)*GPvar/(valid_df$rep[i]-1)
    GP_samplevar_dist = append(GP_samplevar_dist, GP_samplevar) #save sample
  }
  #get unexpectedness:
  VarU <- 2* (0.5-ecdf(GP_samplevar_dist)(valid_df$var[i]))
  VarUs[i] <- VarU
}

for (i in 1:length(X_valid[1,])){
  print(ggplot()+
          geom_point(aes(x= pull(valid_df, i), y=VarUs))+
          geom_hline(yintercept = 0.95, color = "red")+
          geom_hline(yintercept = -0.95, color = "red")+
          geom_hline(yintercept = 0, color = "black")+
          geom_hline(yintercept = -0.5, color = "grey")+
          geom_hline(yintercept = 0.5, color = "grey")+
          theme_minimal(base_size = 16) +
          ylim(-1,1)+
          xlim(0,1)+
          ylab("Variance_Unexpectedness")+
          xlab("x"))
}


################################################################################
#Normality
################################################################################
#get QQ plot
#get standard standardised errors:
raw_std_errs <- (y_valid - pred_valid$mean) /
  sqrt((pred_valid$sd2 + pred_valid$nugs))

#plot qq plot
ggplot()+
  geom_qq(aes(sample = raw_std_errs)) +
  geom_abline(intercept = 0) +
  theme_minimal(base_size = 16) +
  ggtitle("Model_QQ_plot")
```

```
###
#skewness diagnostics
###

#get numerical inversion for skewness parameter
inverse = function(fn, interval = NULL, lower = min(interval),
                   upper = max(interval), ...){
  Vectorize(function(y){
    uniroot(f=function(x){fn(x)-y}, lower=lower, upper=upper, ...)$root
  })
}

skew_calc <- function(param){
  delta <- param / (sqrt(1+param^2))
  term1 <- sqrt(2/pi)*delta
  term2 <- 1 - 2*(delta^2)/pi
  skew <- ( (4-pi)/2) * ( (term1^3) / (term2^(3/2)))
  return(skew)
}

skew_inv <- inverse(skew_calc, lower=-10000000, upper=10000000)

#now we want the predictive distributions for the sample skewness:
set.seed(1234)
#unexpectedness:
SkewUs <- rep(999999, len = nrow(X_valid_unique))
for (i in 1:nrow(X_valid_unique)){
  #get the simulated distribution for the skewness
  GP_skew_dist = c()
  for (m in 1:10000){
    skew_tol <- runif(1, -0.5, 0.5) #skewness tolerance
    skew_param <- skew_inv(skew_tol) #convert to parameter value
    #get samples for ths point using the skew norm dist:
    GPsample <- rsn(valid_df$rep[i], valid_df$GPmu[i],
                    sqrt(valid_df$GPvar[i]+valid_df$GPuncert[i]),
                    skew_param)
    Gpskew = skewness(GPsample) #get sample skewness
    GP_skew_dist = append(GP_skew_dist, Gpskew) #save sample
  }
  SkewU <- 2* (0.5-ecdf(GP_skew_dist)(valid_df$skew[i]))
  SkewUs[i] <- SkewU
}

ggplot()+
  geom_point(aes(x= pull(valid_df, 1), y=SkewUs))+
  geom_hline(yintercept = 0.95, color = "red")+
  geom_hline(yintercept = -0.95, color = "red")+
  geom_hline(yintercept = 0, color = "black")+
  geom_hline(yintercept = -0.5, color = "grey")+
  geom_hline(yintercept = 0.5, color = "grey")+
```

```
  theme_minimal(base_size = 16) +
  ylim(-1,1)+
  xlim(0,1)+
  ylab("Skewness_Unexpectedness")+
  xlab("x")


###
#kurtosis diagnostics
###

#get numerical inversion for kurtosis parameter
kurt_calc <- function(param){
  ((gamma(5/param) * gamma(1/param)) / (gamma(3/param)^2) ) - 3
}

kurt_inv <- inverse(kurt_calc, lower=5e-2, upper=10000000)

#now we want the predictive distributions for the sample kurtosis:
set.seed(1234)
#unexpectedness
KurtUs <- rep(999999, len = nrow(X_valid_unique))
for (i in 1:nrow(X_valid_unique)){
  #get the simulated distribution for the kurtosis
  GP_kurt_dist = c()
  for (m in 1:10000){
    kurt_tol <- runif(1, -0.5, 0.5) #kurtosis tolerance
    kurt_param <- kurt_inv(kurt_tol) #convert to parameter value
    #get samples for ths point using the generalised norm dist:
    GPsample <- rgnorm(valid_df$rep[i], valid_df$GPmu[i],
                       sqrt(valid_df$GPvar[i]+valid_df$GPuncert[i]),
                       kurt_param)
    Gpkurt = kurtosis(GPsample) #get sample kurtosis
    GP_kurt_dist = append(GP_kurt_dist, Gpkurt) #save sample
  }
  KurtU <- 2* (0.5-ecdf(GP_kurt_dist)(valid_df$kurt[i]))
  KurtUs[i] <- KurtU
}

ggplot()+
  geom_point(aes(x= pull(valid_df, 1), y=KurtUs))+
  geom_hline(yintercept = 0.95, color = "red")+
  geom_hline(yintercept = -0.95, color = "red")+
  geom_hline(yintercept = 0, color = "black")+
  geom_hline(yintercept = -0.5, color = "grey")+
  geom_hline(yintercept = 0.5, color = "grey")+
  theme_minimal(base_size = 16) +
  ylim(-1,1)+
  xlim(0,1)+
  ylab("Kurtosis_Unexpectedness")+
  xlab("x")
```

B.2 DetHetGP Code

Here we provide code to implement the DetHetGP model outlined in Chapter 3, using the formulation detailed in Section 3.6.

B.2.1 DetHetGP Stan code

Below is the Stan code implementation of the model, which can be used to fit the parameters of the model. This uses the equations given in Equations (3.9) - (3.11).

```
//first we have to declare our data
data {
  //Stochastic Data:
  int<lower=1> d; //number of input dimensions
  int<lower=1> N_sto; //number of stochastic simulations
  vector[N_sto] y_sto; //stochastic simulation outputs
  row_vector[d] X_sto[N_sto]; //stochastic simulation inputs
  real<lower = 0> nugget; //computational nugget
  int<lower=1> dH_sto; //size of GP 'H' matrix
  matrix[N_sto, dH_sto] H_sto; //GP 'H' matrix
  int<lower=1> dH_var; //size of variance GP 'H' matrix
  matrix[N_sto, dH_var] H_var; //variance GP 'H' matrix

  //Deterministic Data:
  int<lower=1> N_det; //number of deterministic simulations
  vector[N_det] y_det; //deterministic simulation outputs
  row_vector[d] X_det[N_det]; //deterministic inputs
  int<lower=1> dH_det; //size of deterministic GP 'H' matrix
  matrix[N_det, dH_det] H_det; //det GP 'H' matrix
  matrix[N_sto, dH_sto] H_det_sto; //det GP 'H' at stochastic points
}

//We use a trick to allow non-isostropic covariance matrices
//using the isotropic cov_exp_quad squared-exp function.
//we rescale the data instead
//and input an 'identity'' length scale into cov_exp_quad.
//these lines construct said 'identity'' length scales:
transformed data{
  real fake_length_scale_sto;     //for the sto GP
  real fake_length_scale_var; //for the variance GP
  real fake_length_scale_det; //for the det GP

  fake_length_scale_sto = pow(sqrt(2), -1 );
  fake_length_scale_var = pow(sqrt(2), -1 );
  fake_length_scale_det = pow(sqrt(2), -1 );
}

//then we have to declare the parameters
//(which we want to estimate):
```

```
parameters {
  //stochastic GP:
  row_vector<lower=0>[d] length_scale_sto; //length scales
  real<lower=0> alpha_sto; //epistemic variance term
  vector[dH_sto] beta_sto; //mean function coefficients

  //for variance GP:
  row_vector<lower=0>[d] length_scale_var;
  real<lower=0> alpha_var;
  vector[dH_var] beta_var;
  real<lower=0> g; //nugget for variance GP

  vector[N_sto] logintvar; //log intrinsic var at sto inputs

  //for deterministic GP:
  row_vector<lower=0>[d] length_scale_det_raw;
  real<lower=0> alpha_det;
  vector[dH_det] beta_det;
}

//Then we can 'transform' these parameters.
//For GPs, we use this build the mean and cov functions
transformed parameters {
  //again we have to declare these transformed parameters:

  //for stochastic GP
  row_vector[d] Xl_sto[N_sto]; //transformed sto inputs
  matrix[N_sto,N_sto] K_sto; //The stochastic GP covariance matrix
  vector[N_sto] mu_sto; //The stochastic GP mean at the inputs

  //for variance GP
  row_vector[d] Xl_var[N_sto];
  matrix[N_sto, N_sto] K_var;
  matrix[N_sto, N_sto] cholK_var;//variance GP cholesky decomposition
  vector[N_sto] mu_var;
  vector<lower=0>[N_sto] intvar;//intrinsic var at the inputs

  //for det GP model
  row_vector<lower=0>[d] length_scale_det;
  row_vector[d] Xl_det[N_det];
  row_vector[d] Xl_det_sto[N_sto]; //transformed sto inputs (for detGP)
  matrix[N_det, N_det] K_det;
  vector[N_det] mu_det;
  vector[N_sto] mu_det_sto; //det GP mean func at stochastic inputs
  matrix[N_det, N_sto] K_det_detsto; //det GP cov between det and sto
  matrix[N_sto, N_det] K_det_stodet; //det GP cov between sto and det
  matrix[N_sto,N_sto] K_det_stosto;  //detGP cov for sto inputs

  //the big overall process
  vector[N_sto+N_det] y_full; //outputs for both simulators
  //final mean and covariances:
```

```
 vector[N_sto+N_det] ymu; //mean func at all inputs
 matrix[N_sto+N_det, N_sto+N_det] ycov; //cov for all inputs


//now we give the equations for the terms defined above:

 y_full = append_row(y_det, y_sto);

 //////////
 //variance GP:
 ////////
 intvar = exp(logintvar); //undo the variance log transform

 //do trick to allow non-isotropy
 for (i in 1:N_sto)
   Xl_var[i] = X_sto[i]./length_scale_var; //transform the data

 //calculate mean func
 mu_var = H_var * beta_var;

 //build covariance function
 K_var = cov_exp_quad(Xl_var, alpha_var, fake_length_scale_var);
 for (n in 1:N_sto)
   K_var[n, n] = K_var[n, n] + nugget + g; //add nugget

 cholK_var = cholesky_decompose(K_var);

 //////////
 //Stochastic GP:
 ////////

 //do trick to allow non-isotropy
 for (i in 1:N_sto)
   Xl_sto[i] = X_sto[i]./(length_scale_sto); //transform the data

 //calculate mean func
 mu_sto = H_sto * beta_sto;

 //build covariance function
 K_sto = cov_exp_quad(Xl_sto, alpha_sto, fake_length_scale_sto);
 // diagonal elements
 for (n in 1:N_sto)
   K_sto[n, n] = K_sto[n, n] + nugget + intvar[n]; // add var

 //////////
 //Det GP:
 ////////

 //add small number to det lengthscales
 //(see appendix of thesis):
 length_scale_det = length_scale_det_raw + 0.01;
```

```
  //do trick to allow non-isotropy
  for (i in 1:N_det)
    Xl_det[i] = X_det[i]./length_scale_det; //transform data

  //calculate mean func
  mu_det = H_det * beta_det;

  //build covariance function
  K_det = cov_exp_quad(Xl_det, alpha_det, fake_length_scale_det);
  // diagonal elements
  for (n in 1:N_det)
    K_det[n, n] = K_det[n, n] + nugget; // add nugget

  //calculate det mean func at the stochastic inputs:
  mu_det_sto = H_det_sto * beta_det;

  //do non-isotropy trick (for det cov at sto inputs):
  for (i in 1:N_sto)
    Xl_det_sto[i] = X_sto[i]./length_scale_det; //transform data

  /////////
  //build big overall process terms:
  /////////
  K_det_detsto = cov_exp_quad(Xl_det, Xl_det_sto, alpha_det,
  fake_length_scale_det);
  K_det_stodet = cov_exp_quad(Xl_det_sto, Xl_det, alpha_det,
  fake_length_scale_det);

  //build det covariance at sto inputs
  K_det_stosto = cov_exp_quad(Xl_det_sto, alpha_det, fake_length_scale_det);
  for (n in 1:N_sto)
    K_det_stosto[n, n] = K_det_stosto[n, n] + nugget;

  //final mean and final cov:
  ymu = append_row(mu_det, mu_det_sto + mu_sto);
  ycov = append_row(append_col(K_det, K_det_detsto),
  append_col(K_det_stodet, K_sto+K_det_stosto));
}


//now we specifiy the priors
//(for the hyperparameters, and the GP 'priors')
model {

//priors for sto GP
alpha_sto ~ inv_gamma(2,1);
length_scale_sto ~ gamma(4, 4);
beta_sto ~ normal(0, 10);

//priors for var GP
```

```
alpha_var ~ inv_gamma(2,1);
length_scale_var ~ gamma(4, 4);
beta_var ~ normal(0, 10);
g ~ normal(0,1);

//priors for det GP
alpha_det ~ inv_gamma(2,1);
beta_det ~ normal(0, 10);
length_scale_det_raw ~ gamma(4, 4);



//And now, the GP 'priors':
//(log) variance GP
logintvar ~ multi_normal_cholesky(mu_var, cholK_var);

//DetHetGP
y_full ~ multi_normal(ymu, ycov);
}
```

B.2.2 DetHetGP Prediction Code

Below is a set of R functions which will enable predictions from a DetHetGP, assuming parameters estimates have been obtained. This is an R implementation of Equation (3.12).

```
#we want predictions for new points
#but to do this we need a few functions

#we need the squared exponential covariance function
#(as parametrised in the Stan code)
K12 <- function(X1, X2, l) {
  #Calculates squared exponential covariance matrix for X1,X2
  n1 = length(X1[,1]) #number of input points
  n2 = length(X2[,1]) #number of validation points
  d = length(X1[1,])  #number of variables
  D = as.matrix(cdist(scale(X1,center=FALSE,scale=l),
                      scale(X2,center=FALSE,scale=l)))
  return(exp(-(D^2)))
}

#We need the mean function as parametrised in the Stan code
meanfunc <- function(H, beta) {
  #Given a 'H' matrix and the beta coefficients,
  #obtain the mean function values
  H%*%as.vector(beta)
}

#And the function to predict from DetHetGP
DetHetGPPredict <- function(X_sto, y_sto, X_det, y_det, parameterestimates,
```

```
                          X_pred) {

#compute H matrices (for mean functions)
H_sto <- Hmatrix_func(X_sto) #get H matrix for adjustment GP
dimenH_sto <- length(H_sto[1,]) #How many dimensions is it?

H_var <- Hmatrix_func(X_sto) #get H matrix for variance GP
dimenH_var <- length(H_var[1,])

##DetGP
H_det <- Hmatrix_func(X_det) #get detGP H at det points
H_det_sto <- Hmatrix_func(X_det) #get detGP H at sto points
dimenH_det <- length(H_det[1,])



#######
#Variance Prediction
#######
#get predictions (using standard GP prediction equations)
K_var <- (parameterestimates$alpha_var^2) *
  K12(X_sto, X_sto, parameterestimates$length_scale_var) +
  diag(rep(nugget, length(X_sto[,1])), nrow = length(X_sto[,1])) +
  diag(rep(parameterestimates$g, length(X_sto[,1])), nrow = length(X_sto[,1]))
K_var_inv <- chol2inv(chol(K_var)) #get inverse

Kyx_var <- (parameterestimates$alpha_var^2) *
  K12(X_pred, X_sto, parameterestimates$length_scale_var)
Kxy_var <- t(Kyx_var)
Kyy <- (parameterestimates$alpha_var^2) *
  K12(X_pred, X_pred, parameterestimates$length_scale_var) +
  diag(rep(nugget, length(X_pred[,1])), nrow = length(X_pred[,1])) +
  diag(parameterestimates$g, nrow = length(X_pred[,1]))

#obtain H matrix at pred points
H_pred <- Hmatrix_func(X_pred)

#get posterior GP mean and variance
postmean_var <- meanfunc(H_pred, parameterestimates$beta_var) +
  Kyx_var%*%K_var_inv%*%(t(t(parameterestimates$logintvar)) -
                           meanfunc(H_var, parameterestimates$beta_var))
postcov_var <- Kyy - Kyx_var%*%K_var_inv%*%t(Kyx_var)

#undo log transformation:
pred_intvar <- exp(postmean_var + diag(postcov_var)/2)



#######
#DetHetGP Prediction
#######
#Obtain the matrices outlined in Equation 3.12
```

```
  #get Cov(y_full):
  A11 <- (parameterestimates$alpha_det^2) *
    K12(X_det, X_det, parameterestimates$length_scale_det) +
    diag(rep(nugget, length(X_det[,1])), nrow = length(X_det[,1]))
  A12 <- (parameterestimates$alpha_det^2) *
    K12(X_det, X_sto, parameterestimates$length_scale_det)
  A21 <- (parameterestimates$alpha_det^2) *
    K12(X_sto, X_det, parameterestimates$length_scale_det)
  A22 <- (parameterestimates$alpha_det^2) *
    K12(X_sto, X_sto, parameterestimates$length_scale_det) +
    (parameterestimates$alpha_sto^2) *
    K12(X_sto, X_sto, parameterestimates$length_scale_sto) +
    diag(rep(2*nugget, length(X_sto[,1])), nrow = length(X_sto[,1])) +
    diag(parameterestimates$intvar, nrow = length(X_sto[,1]))
  A1 <- cbind(A11, A12)
  A2 <- cbind(A21, A22)
  A <- rbind(A1,A2)
  Ainv <- chol2inv(chol(A))

  #get other covariance matrices:
  Kyx_det <- (parameterestimates$alpha_det^2) *
    K12(X_pred, X_det, parameterestimates$length_scale_det)
  Kyx_sto <- (parameterestimates$alpha_det^2) *
    K12(X_pred, X_sto, parameterestimates$length_scale_det) +
    (parameterestimates$alpha_sto^2) *
    K12(X_pred, X_sto, parameterestimates$length_scale_sto)
  Kyx <- cbind(Kyx_det, Kyx_sto)
  Kyy <- (parameterestimates$alpha_det^2) *
    K12(X_pred, X_pred, parameterestimates$length_scale_det) +
    (parameterestimates$alpha_sto^2) *
    K12(X_pred, X_pred, parameterestimates$length_scale_sto) +
    diag(rep(2*nugget, length(X_pred[,1])), nrow = length(X_pred[,1])) +
    diag(as.numeric(pred_intvar))

  #get Mean(y_full):
  H_input_mat <- rbind(cbind(H_det, 0*H_det), cbind(H_sto, H_sto))
  Hpred <- cbind(Hmatrix_func(X_pred), Hmatrix_func(X_pred))
  Beta <- c(parameterestimates$beta_det, parameterestimates$beta_sto)

  #get posterior mean and variance (using Equation 3.12)
  postmean <- meanfunc(Hpred, Beta) +
    Kyx%*%Ainv%*%(c(y_det, y_sto) - meanfunc(H_input_mat, Beta))
  postcov <- Kyy - Kyx%*%Ainv%*%t(Kyx)

  return(list(mean = postmean, var = diag(postcov), intvar <- pred_intvar))
}
```

B.2.3 DetHetGP Example

Below is an R script which implements a DetHetGP model. The script is a fully self-contained example: providing a toy simulator (the same as in Section 3.1, obtaining training simulations, fitting the emulator, and making predictions. This script assumes the existence of a "DetHetGP.stan" Stan file (provided above in Section B.2.1) and a "DetHetGP Prediction Equations.R" R script (provided above in Section B.2.2).

```
library(lhs)
library(gridExtra)
library(ggplot2)
library(rdist)
library(rstan)

source('DetHetGP_Prediction_Equations.R') #load pred equations
StanLoc <- 'DetHetGP.stan' #location of Stan file used for parameter estimation

#toy simulator
dimen <- 1 #dimension of problem
simulator <- function(x){
  sin(pi+6*pi*x[,1])*(1-x[,1]) + log(0.2+x[,1])  +
    rnorm(length(x[,1]),0,1)*(1.2-x[,1])
}

#deterministic approximation
detsimulator <- function(x){
  sin(pi+6*pi*x[,1])*(1-x[,1]) + log(0.2+x[,1])  + 1*(1.2-x[,1])
}

#get stochastic data:
set.seed(99999)
N_sto <- 50 #number of stochastic training points
X_sto <- maximinLHS(N_sto, dimen) #x values

y_sto_clean <- simulator(X_sto) #run simulator

#get deterministic data:
set.seed(12345)
N_det <- 12 #number of deterministic training points
X_det <- maximinLHS(N_det, dimen) #x values

y_det_clean <- detsimulator(X_det) #run deterministic approximation

#standardise data:
meany <- mean(c(y_sto_clean)) #mean of data sample
sdy <- sd(c(y_sto_clean)) #standard deviation
y_sto <- (y_sto_clean-meany)/sdy #standardised output
y_det <- (y_det_clean-meany)/sdy
```

```
#####################################################################
#Fit DetHetGP
nugget <-  0.0001 #computational nugget
#get function which produces 'H' matrices
#(matrix of regression terms for mean function):
Hmatrix_func <- function(x){
  cbind(1,x) #linear mean - can change this
}
#compute H matrices (for mean functions at inputs):
H_sto <- Hmatrix_func(X_sto) #get H matrix for adjustment GP
dimenH_sto <- length(H_sto[1,]) #How many dimensions is it?
H_var <- Hmatrix_func(X_sto) #get H matrix for variance GP
dimenH_var <- length(H_var[1,])
H_det <- Hmatrix_func(X_det) #get H matrix for det GP at deterministic points
H_det_sto <- Hmatrix_func(X_sto) #get H matrix for det GP at stochastic points
dimenH_det <- length(H_det[1,])


#Prepare everything for Stan
#initialise data
data <- list(d = dimen, N_sto = N_sto, y_sto = array(y_sto, dim = N_sto),
             X_sto = X_sto, nugget = nugget, dH_sto = dimenH_sto, H_sto = H_sto,
             dH_var = dimenH_var, H_var = H_var, N_det = N_det, X_det = X_det,
             y_det = array(y_det, dim = N_det), dH_det = dimenH_det,
             H_det = H_det, H_det_sto = H_det_sto)
#initialise Stan model
DetHetmodel <- stan_model(file=StanLoc)

#Run optimiser
opt <- optimizing(DetHetmodel, data = data, as_vector = FALSE,
                          iter = 10000, verbose = TRUE)

#Extract MAP estimates
parameterestimates <- opt$par
#####################################################################

#make predictions
X_pred <-as.matrix(seq(0,1,len = 100)) #x locations
pred <- DetHetGPPredict(X_sto, y_sto, X_det, y_det, parameterestimates, X_pred)

#unstandardise predictions
pred$var <- pred$var*sdy^2
pred$mean <- pred$mean*sdy + meany


#plot the predictions:
ggplot() +
  geom_ribbon(aes(x=X_pred, ymin = (pred$mean -2*sqrt(pred$var)),
                  ymax = pred$mean + 2*sqrt(pred$var), fill = 'DetHet'),
              alpha = 0.15) +
```

```
geom_line(aes(x=X_pred, y=pred$mean, colour = 'DetHet')) +
geom_point(aes(x=X_sto, y=y_sto_clean, colour = 'True')) +
geom_point(aes(x=X_det, y=y_det_clean, colour = 'True'), shape = 3) +
theme_minimal(base_size = 16) +
ylab('y') +
xlab('x')+
xlim(0,1)+
coord_cartesian(ylim=c(-5.5, 3))+
scale_fill_manual(values = c('True' = '#000000', 'DetHet' = '#0072B2'),
                  guide=FALSE) +
scale_colour_manual(values = c('True' = '#000000', 'DetHet' = '#0072B2'),
                    guide=FALSE)
```

Bibliography

Al-Taweel, Y. (2018). *Diagnostics and Simulation-Based Methods for Validating Gaussian Process Emulators*. PhD thesis, University of Sheffield.

Andrianakis, I. and Challenor, P. G. (2012). The effect of the nugget on gaussian process emulators of computer models. *Computational Statistics & Data Analysis*, 56(12):4215–4228.

Andrianakis, I., Vernon, I., McCreesh, N., McKinley, T., Oakley, J., Nsubuga, R., Goldstein, M., and White, R. (2017). 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.

Andrianakis, I., Vernon, I. R., McCreesh, N., McKinley, T. J., Oakley, J. E., Nsubuga, R. N., Goldstein, M., and White, R. G. (2015). Bayesian history matching of complex infectious disease models using emulation: a tutorial and a case study on hiv in uganda. *PLoS Comput Biol*, 11(1):e1003968.

Ankenman, B., Nelson, B. L., and Staum, J. (2010). Stochastic kriging for simulation metamodeling. *Operations Research*, 58(2):371–382.

Azzimonti, D., Ginsbourger, D., Chevalier, C., Bect, J., and Richet, Y. (2021). Adaptive design of experiments for conservative estimation of excursion sets. *Technometrics*, 63(1):13–26.

Ba, S., Joseph, V. R., et al. (2012). Composite gaussian process models for emulating expensive functions. *The Annals of Applied Statistics*, 6(4):1838–1860.

Ba, S., Myers, W. R., and Brenneman, W. A. (2015). Optimal sliced latin hypercube designs. *Technometrics*, 57(4):479–487.

Baker, E., Challenor, P., and Eames, M. (2020a). Predicting the output from a stochastic computer model when a deterministic approximation is available. *Journal of Computational and Graphical Statistics*, 29(4):786–797.

Baker, E., Challenor, P., and Eames, M. (2021). Future proofing a building design using history matching inspired level-set techniques. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 70(2):335–350.

Baker, E., Barbillon, P., Fadikar, A., Gramacy, R. B., Herbei, R., Higdon, D., Huang, J., Johnson, L. R., Ma, P., Mondal, A., et al. (2020b). Analyzing stochastic computer models: A review with opportunities. *arXiv preprint arXiv:2002.01321*.

Bastos, L. S. and O'Hagan, A. (2009). Diagnostics for gaussian process emulators. *Technometrics*, 51(4):425–438.

BBC (2013). 10 ways the UK is ill-prepared for a heatwave. `https://www.bbc.co.uk/news/magazine-23341698`. Accessed: 2019-08-09.

Binois, M. and Gramacy, R. B. (2017). hetgp: Heteroskedastic gaussian process modeling and design under replication. *R package version*, 1(1).

Binois, M., Gramacy, R. B., and 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., and Ludkovski, M. (2019). Replication or exploration? sequential design for stochastic simulation experiments. *Technometrics*, 61(1):7–23.

Boukouvalas, A. and Cornford, D. (2009). Learning heteroscedastic gaussian processes for complex datasets. *Technical report*.

Boukouvalas, A., Cornford, D., and Stehlík, M. (2014a). Optimal design for correlated processes with input-dependent noise. *Computational Statistics & Data Analysis*, 71:1088–1102.

Boukouvalas, A., Sykes, P., Cornford, D., and Maruri-Aguilar, H. (2014b). Bayesian precalibration of a large stochastic microsimulation model. *IEEE Transactions on Intelligent Transportation Systems*, 15(3):1337–1347.

Bratley, P. and Fox, B. L. (1988). Algorithm 659: Implementing sobol's quasirandom sequence generator. *ACM Transactions on Mathematical Software (TOMS)*, 14(1):88–100.

Brynjarsdóttir, J. and O'Hagan, A. (2014). Learning about physical parameters: The importance of model discrepancy. *Inverse problems*, 30(11):114007.

Cantoni, E. and Hastie, T. (2002). Degrees-of-freedom tests for smoothing splines. *Biometrika*, 89(2):251–263.

Craig, P. S., Goldstein, M., Seheult, A. H., and 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*, pages 37–93. Springer.

Crawley, D. B., Lawrie, L. K., Pedersen, C. O., and Winkelmann, F. C. (2000). Energy plus: energy simulation program. *ASHRAE journal*, 42(4):49–56.

Cutajar, K., Pullin, M., Damianou, A., Lawrence, N., and González, J. (2019). Deep gaussian processes for multi-fidelity modeling. *arXiv preprint arXiv:1903.07320*.

Damblin, G., Barbillon, P., Keller, M., Pasanisi, A., and Parent, É. (2018). Adaptive numerical designs for the calibration of computer codes. *SIAM/ASA Journal on Uncertainty Quantification*, 6(1):151–179.

Damianou, A. and Lawrence, N. D. (2013). Deep gaussian processes. In *Artificial intelligence and statistics*, pages 207–215. PMLR.

Daniel, W. W. et al. (1990). Applied nonparametric statistics.

Davis, C. B., Hans, C. M., and Santner, T. J. (2020). Prediction of non-stationary response functions using a bayesian composite gaussian process. *Computational Statistics & Data Analysis*, page 107083.

Drovandi, C. C., Nott, D. J., and Pagendam, D. E. (2017). New insights into history matching via sequential monte carlo. *arXiv preprint arXiv:1710.03133*.

Eames, M., Kershaw, T., and Coley, D. (2011). On the creation of future probabilistic design weather years from ukcp09. *Building Services Engineering Research and Technology*, 32(2):127–142.

Eames, M. E. (2016). An update of the uk's design summer years: Probabilistic design summer years for enhanced overheating risk analysis in building design. *Building services engineering research and technology*, 37(5):503–522.

Eames, M. E., Ramallo-Gonzalez, A. P., and Wood, M. (2016). An update of the uk's test reference year: The implications of a revised climate on building design. *Building Services Engineering Research and Technology*, 37(3):316–333.

Epstein, E. S. (1969). A scoring system for probability forecasts of ranked categories. *Journal of Applied Meteorology*, 8(6):985–987.

Fadikar, A., Higdon, D., Chen, J., Lewis, B., Venkatramanan, S., and Marathe, M. (2018). Calibrating a stochastic, agent-based model using quantile-based emulation. *SIAM/ASA Journal on Uncertainty Quantification*, 6(4):1685–1706.

Field, K., Deru, M., and Studer, D. (2010). Using doe commercial reference buildings for simulation studies. Technical report, National Renewable Energy Lab.(NREL), Golden, CO (United States).

Forrester, A., Sobester, A., and Keane, A. (2008). *Engineering design via surrogate modelling: a practical guide*. John Wiley & Sons.

Frazier, P. I. (2018). A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*.

Gneiting, T. and 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., and Bishop, C. M. (1998). Regression with input-dependent noise: A gaussian process treatment. In *Advances in neural information processing systems*, pages 493–499.

Goldstein, M. and Rougier, J. (2009). Reified bayesian modelling and inference for physical systems. *Journal of statistical planning and inference*, 139(3):1221–1239.

Goldstein, M. and Wooff, D. (2007). *Bayes linear statistics: Theory and methods*, volume 716. John Wiley & Sons.

Gotovos, A., Casati, N., Hitz, G., and Krause, A. (2013). Active learning for level set estimation. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 1344–1350. AAAI Press.

Gramacy, R. B. and Lee, H. K. (2012). Cases for the nugget in modeling computer experiments. *Statistics and Computing*, 22(3):713–722.

Gramacy, R. B. and 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.

Haaland, B., Qian, P. Z., et al. (2011). Accurate emulators for large-scale computer experiments. *The Annals of Statistics*, 39(6):2974–3002.

Henderson, D. A., Boys, R. J., Krishnan, K. J., Lawless, C., and 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.

Herbei, R. and Berliner, L. M. (2014). Estimating ocean circulation: an mcmc approach with approximated likelihoods via the bernoulli factory. *Journal of the American Statistical Association*, 109(507):944–954.

Hersbach, H. (2000). Decomposition of the continuous ranked probability score for ensemble prediction systems. *Weather and Forecasting*, 15(5):559–570.

Holden, P. B., Edwards, N. R., Hensman, J., and Wilkinson, R. D. (2018). Abc for climate: dealing with expensive simulators. *Handbook of approximate Bayesian computation*, pages 569–95.

Imam, S., Coley, D. A., and Walker, I. (2017). The building performance gap: Are modellers literate? *Building services engineering research and technology*, 38(3):351–375.

Jalali, H., Van Nieuwenhuyse, I., and Picheny, V. (2017). Comparison of kriging-based methods for simulation optimization with heterogeneous noise. *European Journal of Operational Research*, 261(1):279–301.

Jarque, C. M. and Bera, A. K. (1980). Efficient tests for normality, homoscedasticity and serial independence of regression residuals. *Economics letters*, 6(3):255–259.

Jenness, S. M., Goodreau, S. M., and Morris, M. (2018). Epimodel: an r package for mathematical modeling of infectious disease over networks. *Journal of statistical software*, 84.

Johnson, L. R. (2010). Implications of dispersal and life history strategies for the persistence of linyphiid spider populations. *Ecological modelling*, 221(8):1138–1147.

Jones, D., Schonlau, M., and Welch, W. (1998). Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13(4):455–492.

Kennedy, J. C., Henderson, D. A., and Wilson, K. J. (2020). Multilevel emulation for stochastic computer models with an application to large offshore windfarms. *arXiv preprint arXiv:2003.08921*.

Kennedy, M. C. and O'Hagan, A. (2000). Predicting the output from a complex computer code when fast approximations are available. *Biometrika*, 87(1):1–13.

Kennedy, M. C. and O'Hagan, A. (2001). Bayesian calibration of computer models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(3):425–464.

Kersting, K., Plagemann, C., Pfaff, P., and Burgard, W. (2007). Most likely heteroscedastic gaussian process regression. In *Proceedings of the 24th international conference on Machine learning*, pages 393–400.

Kim, Y.-J. (2016). Comparative study of surrogate models for uncertainty quantification of building energy model: Gaussian process emulator vs. polynomial chaos expansion. *Energy and Buildings*, 133:46–58.

Kim, Y.-J., Ahn, K.-U., Park, C., and Kim, I.-H. (2013). Gaussian emulator for stochastic optimal design of a double glazing system. In *Proceedings of the 13th IBPSA Conference, August*, pages 25–28.

Kim, Y.-J. and Park, C.-S. (2016). Stepwise deterministic and stochastic calibration of an energy simulation model for an existing building. *Energy and Buildings*, 133:455–468.

Lawson, A., Goldstein, M., and Dent, C. (2016). Bayesian framework for power network planning under uncertainty. *Sustainable Energy, Grids and Networks*, 7:47–57.

Le Gratiet, L. (2013). Bayesian analysis of hierarchical multifidelity codes. *SIAM/ASA Journal on Uncertainty Quantification*, 1(1):244–269.

Lilliefors, H. W. (1967). On the kolmogorov-smirnov test for normality with mean and variance unknown. *Journal of the American statistical Association*, 62(318):399–402.

Loeppky, J. L., Sacks, J., and Welch, W. J. (2009). Choosing the sample size of a computer experiment: A practical guide. *Technometrics*, 51(4):366–376.

Lowe, J. A., Bernie, D., Bett, P., Bricheno, L., Brown, S., Calvert, D., Clark, R., Eagle, K., Edwards, T., Fosser, G., et al. (2018). UKCP18 science overview report. *Exeter, UK: Met Office Hadley Centre*.

Lyu, X., Binois, M., and Ludkovski, M. (2018). Evaluating gaussian process metamodels and sequential designs for noisy level set estimation. *arXiv preprint arXiv:1807.06712*.

Marrel, A., Iooss, B., Da Veiga, S., and Ribatet, M. (2012). Global sensitivity analysis of stochastic computer models with joint metamodels. *Statistics and Computing*, 22(3):833–847.

Matthews, A. G. D. G., Van Der Wilk, M., Nickson, T., Fujii, K., Boukouvalas, A., León-Villagrá, P., Ghahramani, Z., and Hensman, J. (2017). Gpflow: A gaussian process library using tensorflow. *The Journal of Machine Learning Research*, 18(1):1299–1304.

McKay, M. D., Beckman, R. J., and Conover, W. J. (2000). A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 42(1):55–61.

McKinley, T. J., Vernon, I., Andrianakis, I., McCreesh, N., Oakley, J. E., Nsubuga, R. N., Goldstein, M., White, R. G., et al. (2018). Approximate bayesian computation and simulation-based inference for complex stochastic epidemic models. *Statistical science*, 33(1):4–18.

Mohammadi, H., Challenor, P., and Goodfellow, M. (2019). Emulating dynamic non-linear simulators using gaussian processes. *Computational Statistics & Data Analysis*, 139:178–196.

Nadarajah, S. (2005). A generalized normal distribution. *Journal of Applied statistics*, 32(7):685–694.

Neal, R. M. (1997). Monte carlo implementation of gaussian process models for bayesian regression and classification. *arXiv preprint physics/9701026*.

Oakley, J. and O'Hagan, A. (2002). Bayesian inference for the uncertainty distribution of computer model outputs. *Biometrika*, 89(4):769–784.

O'Hagan, A. (2006). Bayesian analysis of computer code outputs: A tutorial. *Reliability Engineering & System Safety*, 91(10-11):1290–1300.

O'Hagan, A. and Leonard, T. (1976). Bayes estimation subject to uncertainty about parameter constraints. *Biometrika*, 63(1):201–203.

Peleg, N., Fatichi, S., Paschalis, A., Molnar, P., and Burlando, P. (2017). An advanced stochastic weather generator for simulating 2-d high-resolution climate variables. *Journal of Advances in Modeling Earth Systems*, 9(3):1595–1627.

Picheny, V., Ginsbourger, D., Roustant, O., Haftka, R. T., and Kim, N.-H. (2010). Adaptive designs of experiments for accurate approximation of a target region. *Journal of Mechanical Design*, 132(7).

Plumlee, M. and Tuo, R. (2014). Building accurate emulators for stochastic simulations via quantile kriging. *Technometrics*, 56(4):466–473.

Pukelsheim, F. (1994). The three sigma rule. *The American Statistician*, 48(2):88–91.

Qian, P. Z. G., Wu, H., and Wu, C. J. (2008). Gaussian process models for computer experiments with qualitative and quantitative factors. *Technometrics*, 50(3):383–396.

Rasmussen, C. and Williams, C. (2006). *Gaussian Processes for Machine Learning*. Adaptive computation and machine learning. MIT Press.

Richardson, C. W. (1981). Stochastic simulation of daily precipitation, temperature, and solar radiation. *Water resources research*, 17(1):182–190.

Sacks, J., Welch, W. J., Mitchell, T. J., and Wynn, H. P. (1989). Design and analysis of computer experiments. *Statistical science*, pages 409–423.

Salter, J. M. and Williamson, D. (2016). A comparison of statistical emulation methodologies for multi-wave calibration of environmental models. *Environmetrics*, 27(8):507–523.

Salter, J. M., Williamson, D. B., Scinocca, J., Kharin, V., et al. (2019). Uncertainty quantification for computer models with spatial output using calibration-optimal bases. *Journal of the American Statistical Association*, 114(528):1800–1814.

Schnieders, J. and Hermelink, A. (2006). Cepheus results: measurements and occupants' satisfaction provide evidence for passive houses being an option for sustainable building. *Energy Policy*, 34(2):151–171.

Shapiro, S. S. and Wilk, M. B. (1965). An analysis of variance test for normality (complete samples). *Biometrika*, 52(3/4):591–611.

Srinivas, N., Krause, A., Kakade, S. M., and Seeger, M. W. (2012). Information-theoretic regret bounds for gaussian process optimization in the bandit setting. *IEEE Transactions on Information Theory*, 58(5):3250–3265.

Stan Development Team (2016). Stan modeling language users guide and reference manual.

Stuart, A. and Ord, J. K. (1994). *Kendall's advanced theory of statistics*. Wiley.

Sullivan, T. J. (2015). *Introduction to uncertainty quantification*, volume 63. Springer.

TM52 (2013). The limits of thermal comfort : avoiding overheating in European buildings. Technical report, CIBSE.

Tuo, R., Wu, C. J., et al. (2015). Efficient calibration for imperfect computer models. *The Annals of Statistics*, 43(6):2331–2352.

Vernon, I., Goldstein, M., and Bower, R. (2014). Galaxy formation: Bayesian history matching for the observable universe. *Statistical science*, pages 81–90.

Vernon, I., Goldstein, M., Bower, R. G., et al. (2010). Galaxy formation: a bayesian uncertainty analysis. *Bayesian analysis*, 5(4):619–669.

Volodina, V. and Williamson, D. (2020). Diagnostics-driven nonstationary emulators using kernel mixtures. *SIAM/ASA Journal on Uncertainty Quantification*, 8(1):1–26.

Vysochanskij, D. and Petunin, Y. I. (1980). Justification of the 3σ rule for unimodal distributions. *Theory of Probability and Mathematical Statistics*, 21(25-36).

Wate, P., Iglesias, M., Coors, V., and Robinson, D. (2020). Framework for emulation and uncertainty quantification of a stochastic building performance simulator. *Applied Energy*, 258:113759.

Williamson, D. B., Blaker, A. T., and Sinha, B. (2017). Tuning without over-tuning: parametric uncertainty quantification for the nemo ocean model. *Geoscientific Model Development*, 10(4):1789–1816.

Xie, G. and Chen, X. (2017). A heteroscedastic t-process simulation metamodeling approach and its application in inventory control and optimization. In *Simulation Conference (WSC), 2017 Winter*, pages 3242–3253. IEEE.

Xu, W., Williamson, D. B., and Challenor, P. (2021). Local voronoi tessellations for robust multiwave calibration of computer models. *International Journal for Uncertainty Quantification*, 11(5).

Yan, J., Kim, Y.-J., Ahn, K.-U., and Park, C.-S. (2013). Gaussian process emulator for optimal operation of a high rise office building. In *Proceedings of 13th International Building Performance Simulation Association Conference*.

Zhang, B., Gramacy, R. B., Johnson, L., Rose, K. A., and Smith, E. (2020). Batch-sequential design and heteroskedastic surrogate modeling for delta smelt conservation. *arXiv preprint arXiv:2010.06515*.