Systematic Comparison of Designs and Emulators for Computer Experiments Using a Library of Test Functions

by

Dylan Maciel

B.Sc., Dalhousie University, 2017

Project Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

in the
Department of Statistics & Actuarial Science
Faculty of Science

Declaration of Committee

Name:	**Dylan Maciel**
Degree:	**Master of Science**
Thesis title:	**Systematic Comparison of Designs and Emulators for Computer Experiments Using a Library of Test Functions**
Committee:	**Chair:** Jinko Graham Professor, Statistics and Actuarial Science
	Derek Bingham Supervisor Professor, Statistics and Actuarial Science
	Boxin Tang Committee Member Professor, Statistics and Actuarial Science
	Richard Lockhart External Examiner Professor, Statistics and Actuarial Science

Abstract

As computational resources have become faster and more economical, scientific research has transitioned from using only physical experiments to using simulation-based exploration. A body of literature has since grown aimed at the design and analysis of so-called computer experiments. While this literature is large and active, little work has been focused on comparing methods. This project presents ways of comparing and evaluating both design and emulation methods for computer experiments. Using a suite of test functions — in this work we introduce the Virtual Library of Computer Experiments — a procedure is established which can provide guidance as to how to proceed in simulation problems. An illustrative comparison is performed for each context; putting three emulators, then four experimental designs up against each other; while also highlighting possible considerations for test function choice.

Keywords: Gaussian process; Computer experiments; Emulation; Design of experiments

Dedication

To Mom,

look at us now.

Acknowledgements

The first acknowledgement goes to my supervisor Derek Bingham; I appreciate you taking a chance on someone with limited background in statistics and guiding with patience. I came out of our talks not only feeling reinvigorated but frequently with a bit of life advice that was always relevant.

I extend my gratitude to those on my examining committee Richard Lockhart and Boxin Tang, as well as the chair Jinko Graham for taking their time and providing feedback on this project. To the professors whose courses I took, thank you for being great examples of openness and work ethic. A special thanks goes to Marie Loughin for managing the workshop and all the frustration it comes with.

To those in my cohort; I'll look back at our, admittedly overlong, coffee breaks with fondness. Thanks for filling these past couple years with great conversation.

Most importantly, I wouldn't be here without my family, I've spent too much time thinking about what to write here but words are not enough; I am forever grateful for your presence.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

As computers have become more capable of running many complicated calculations, there has been a transition in the way scientific experiments are performed. Decades ago the vast majority were real-world experiments, but recently many scientists have transitioned to modeling the processes of the real-world with mathematical models run on computers; these are often referred to as computer experiments or simulator experiments. As the processes become more complex the computational cost of running a simulator can become too large; presenting two significant problems. First, since there is only limited resources, which initial set of input values do we run the simulator? Secondly, given this set of initial simulator runs, how do we predict the simulator for new sets of input values? These are known as the design problem and the emulation problem, respectively.

The literature for the design and emulation of computer experiments is vast. These papers generally follow the same structure; a method is proposed, then applied to a test function or two for comparison. Comparison is usually done in two ways; theoretical results are derived and/or empirical comparison is done in which performance is compared on single, large test datasets for each test function. This can be done because test functions are typically chosen to be fast running simulators. The issue here is that comparisons and conclusions are being made over too few examples. There is also a small literature for validating and comparing emulators – but again, as of yet there has not been a way to systematically evaluate the performance of emulators, though its need has been acknowledged [12, 46].

With this project we aim to address this problem; the absence of systematic evaluation of methods for the design and analysis for computer experiments. This work started with the creation of the Virtual Library of Simulator Experiments (VLSE) [63], which provides a suite of test functions that can be used for evaluations of new methods. We then outline our recommendation for the use of this library. The end goal is two-fold. First; to have a creator of a new method access the VLSE, apply their method to many test functions, then compare their method to others that have also been evaluated using the VLSE. Second, is to translate the academic approaches of researchers into practical guidance for experimenters in the future.

The rest of this project is organised as follows; in chapter 2 we provide brief history of the design and emulation for computer experiments, along with an introduction to some of the test functions found in the VLSE and how we can classify these test functions. In chapter 3 we define the emulation problem and describe three possible choices for emulators; the conventional choice Gaussian process (GP) model, along with the Bayesian Treed Gaussian Process (TGP) model, and Bayesian Adaptive Spline Surfaces (BASS). Chapter 4 contains a description of the design problem and briefly describes a few types of designs for computer experiments. In chapter 5 we outline the performance metrics that we recommend for the evaluation and comparison of both emulators and designs for simulations. An outline of how to use the VLSE for comparisons is presented in chapter 6. Several measures of relative performance are introduced, and illustrative comparisons are proposed for both the design and emulation methods described in earlier in the work. Lastly, chapter 7 provides a brief summary of the project, along with some future considerations.

Chapter 2

Background

Mathematical models created by physicists and engineers are often implemented through large-scale computer codes. These models are used to gain a better understanding of physical processes, and regularly replace or supplement the use of real-world experiments. This leads to highly complex codes that often take minutes, hours, days, or more per evaluation. To explore the response surface for one of these simulators it is customary to run an initial set of trials and model the response surface as a function of the inputs. This activity is called computer model emulation and has been done in applications ranging from cosmology [39], to radiative shock experiments [35], to water flow through a borehole [32], to thermodynamic models of sea ice [11], etc.

Building an emulator, or statistical surrogate, for a complex computer code is an essential task due to the simulator's computational cost. With a fast emulator, scientists can tackle many problems such as calibrating the computer experiment to real-world data [40, 34, 5, 62], finding locations of optima [37, 51], conducting sensitivity analysis [56, 66, 24], etc. There are two crucial steps to building an emulator: (i) specifying the statistical model; and (ii) choosing the experimental design.

The main goal of the emulator is to predict the code's output $y \in \mathcal{R}$ at input values $\boldsymbol{x} \in \mathcal{R}^d$. It is also of interest to accompany these predictions with a measure of uncertainty - we want a relatively fast way to estimate the probability distribution for $\hat{y}(\boldsymbol{x})$. The problem is to build the surrogate given an initial set of n input values $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n]^T$ and corresponding observed output values $\boldsymbol{y} = [y(\boldsymbol{x}_1), y(\boldsymbol{x}_2), \ldots, y(\boldsymbol{x}_n)]^T$ from the simulator. In this work, only deterministic simulators are considered (i.e. for a set of inputs the simulation always results in the same observed output).

The first computer experiment appears to be conducted by Metropolis et al. [50]. The initial works in the statistical design and analysis of computer experiments dates back to the late eighties [58, 18]. These works appear to be the first to implement and gain traction with the now conventional GP as the choice statistical model for computer experiments. In short, GP methodology places a distribution over the function space for the computer model $y(\boldsymbol{x})$ and treats it as a single realization from this distribution. GP regression is attractive in this setting as it is a flexible nonparametric method with a couple of key attributes: (i) it provides a foundation for uncertainty quantification (UQ) in deterministic settings; (ii) model predictions at sampled inputs are exactly the observed outputs, i.e. the GP interpolates the responses without error. A more in depth consideration can be found in section 3.1.

Once a statistical model is chosen, one must test its ability before applying it in a real world context. Using GP theory is a good starting point to assess the efficacy of design and analysis approaches for computer models. However, while GPs have favourable properties, in truth simulators are not realizations of a GP. Comparisons that do not rely on GP theory are necessary. For this reason when comparing GPs or other emulation methods, it is common to use fast-running simulators to create large test data sets. This allows for quantification of performance on more realistic examples whilst keeping the computational expenses low.

Before constructing an emulator for a complex computer code, one must first choose the input settings at which the computer code will be run. Due to the high computational cost the choice of these inputs, or design, is an essential task. Classical experimental design aims to control for random error and model bias through replication, blocking, and randomization [69]. However, due to the deterministic nature of computer codes these principles are deemed unnecessary. In this context we require a design that contains n unique points in the input space. For computer model emulation, the work has largely focused on model-free designs, $\boldsymbol{X}$, which commonly aim to fulfill some geometric criteria for the design space [45].

The literature for the design of computer experiments generally seeks designs that are space-filling - that is, designs that spread the observations of $y(\boldsymbol{x})$ throughout the entire input space as good as possible. The thought; since we have limited knowledge about the behaviour of $y(\boldsymbol{x})$, this should optimize the interpolation capabilities of the chosen model in the end. As we shall see, prediction using a GP is a weighted average of observation, with weights based on the distance from the input where the prediction is to be made. Thus local design points are beneficial.

Two prominent space-filling designs include Latin hypercube designs (LHDs) and maximin distance designs. LHDs were introduced by McKay et al. in 1979 as a way to ensure that random samples are uniformly distributed in each dimension. Maximin distance designs, introduced by Johnson et al. in 1990, are attractive as they were shown to minimize the predictive variance of GP models. There are various other approaches to design in this context, many of which build upon the ideas established in these works and claim to be optimal in some sense.

Typically, designs are compared with a metric computed on the input space; e.g. measures based on distance between design points [27]. These metrics have, on occasion, been presented without much concern for the implications on the performance of the emulator. An important question is to ask whether or not one design approach is practically different from another. If comparisons of designs are done by way of their effect on the performance of a GP, it is usually done on too few examples. This leads us to question whether the small set of test problems is sufficient for selecting an appropriate design in a future analysis.

In a similar vein, emulation methods are sometimes compared on single data sets using a variety of different metrics for only a few test functions – the Borehole function seems to be the most popular choice [32]. One must again ask; do the chosen test problems provide enough information to generalise the results to a new problem? Or, has the limited choice led to the conclusions being overconfident about the performance of one approach over another. For the end user of these designs and emulators, the ways of comparing methods up until now have left much to be desired in way of pragmatism; a comparison done over a small set of problems may be insufficient for selecting how to proceed in a real-world context.

Does choice of design truly matter? Which emulation method is "best" for their context? Ideally, these questions can be easily answered through the literature, but to do this, comparisons need to be done over numerous examples. Taking inspiration from the global optimization literature [20]; we introduce the VLSE which researchers and practitioners alike can use to rigorously put methods up against each other over many test functions and come to informed conclusions.

2.1 Test Functions

The VLSE aggregates many fast-running test functions found in the literature; not just the computer experiment literature, but others such as optimization. Ideally this library aims to capture realistic design and emulation situations that one might come across in practice – the library should continually be growing. That way, when a practitioner is presented with a novel problem, they can access the VLSE and compare the performance of candidate methods across many test functions; leading to an informed decision on how to proceed. Or, if a researcher comes up with a novel emulation or design method they can make extensive comparisons with benchmark methods. It is therefore useful to try to classify the test functions contained within the VLSE.

There are many possible ways to classify test functions; here we outline a few. First, we can separate test functions by the dimension of their input space d. We can then consider separability by input dimension. A test function $f(\boldsymbol{x})$ is considered additively separable if there exists functions $g_1(x_1),\ g_2(x_2), \ldots,\ g_d(x_d)$, such that

$$f(\boldsymbol{x}) = \sum_{i=1}^{d} g_i(x_i). \tag{2.1.1}$$

Similarly, a test function $f(\boldsymbol{x})$ is considered multiplicatively separable if there exists functions $g_1(x_1),\ g_2(x_2), \ldots,\ g_d(x_d)$, such that

$$f(\boldsymbol{x}) = \prod_{i=1}^{d} g_i(x_i). \tag{2.1.2}$$

We can also differentiate test functions by their behaviour over the input space; these include but are not limited to: continuity and differentiability over the entire domain of interest, and modality. A test function $f(\boldsymbol{x})$ is considered to be multimodal if it has many local optima.

For an example classification we consider the exponential function found in Currin et al. [17];

$$f(\boldsymbol{x}) = \left[1 - \exp\left(-\frac{1}{2x_2}\right)\right] \frac{2300x_1^3 + 1900x_1^2 + 2092x_1 + 60}{100x_1^3 + 500x_1^2 + 4x_1 + 20}.$$

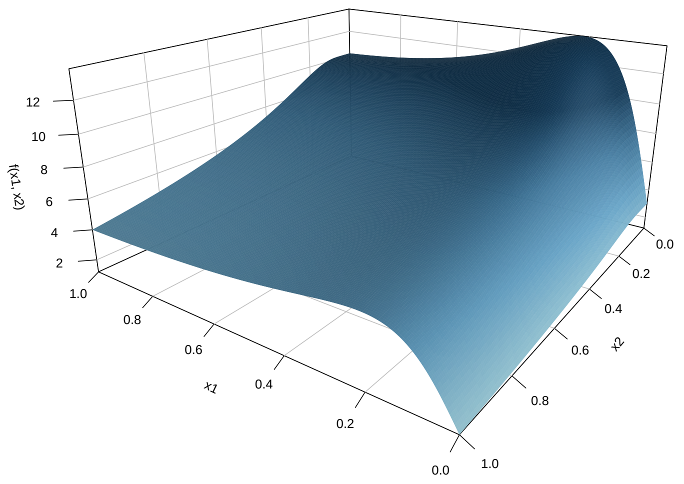

Figure 2.1: Two-dimension test function found in Currin et al. [17].

This is a two-dimensional test function that is multiplicatively separable, differentiable, and continuous on the unit square. A plot of this function can be found in figure 2.1, where we see the function is smooth and has one optimum in the design space.

Classifying test functions in this way is useful if an emulation method is designed to perform in certain situations. For example, TGPs are expected to perform better for data with discontinuities when compared to a stationary GP; we can therefore pick only test functions that allow us to test this hypothesis. The example test function given in Gramacy et al. [30] is:

$$f(x) = \begin{cases} \sin(\frac{\pi x}{5}) + 0.2\cos(\frac{4\pi x}{5}), & \text{if } x < 10 \\ \frac{x}{10} - 1, & \text{otherwise.} \end{cases} \tag{2.1.3}$$

A plot of this function can be found in figure 2.2. This is a piecewise function which does not have the same level of smoothness throughout $\boldsymbol{X}$ and is clearly not a realization of a GP.

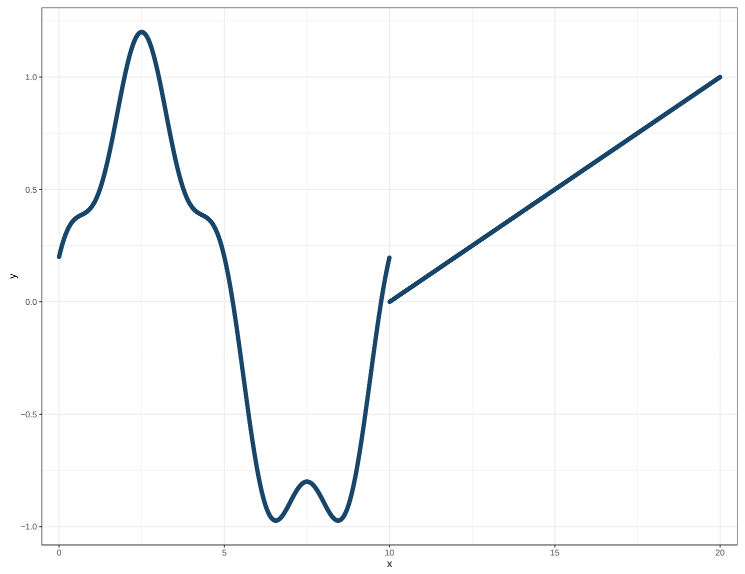

Figure 2.2: One dimensional piecewise test function found in [33] and multiple papers by Gramacy and Lee.

In what follows, we propose and show how to compare emulation and design approaches using the VLSE. The point is not to declare a winner amongst those we will illustrate, but instead to show how methods can be compared in practice by researchers and users of computer models. It is important to note that the VLSE is in and of itself not the take away of this work. What is key, is that approaches for the design and analysis of computer experiments be compared and contrasted in ways that provide enough information for experimenters to make sensible choices in their settings.

Chapter 3

Emulation

In this chapter we outline three approaches for the emulation of computer experiments; GPs, TGPs, and BASS. The idea is to propose the steps one should follow when selecting an emulation approach or evaluating new methods; this is illustrated in section 6.2. This is by no means an exhaustive list of emulation methods, and instead serves as an example of how one may proceed to compare methods.

3.1 Gaussian Processes

The most often used model for simulator emulation is a stationary, anisotropic Gaussian process. A GP is the extension of the univariate normal distribution where curves are sampled. Thus, it is a distribution over functions, where for a single point it is marginally normal. To define a GP we need to specify two things; a mean function $\mu = \mu(\boldsymbol{x})$ and a covariance matrix $\boldsymbol{\Sigma} = \sigma^2 \boldsymbol{R}$, where $\boldsymbol{R}$ is the correlation matrix. There are two common ways to specify the mean function, with a constant mean or with linear basis functions. In cases where a constant mean is used, it is common to center and scale the response values by their mean and standard deviation, respectively, and view the GP as having a mean zero. When prior knowledge about the behaviour of the simulator is available, this can be used to specify the form of the basis functions.

A kernel or correlation function is used to define the correlation of two output values,

$$Corr(\boldsymbol{y}(\boldsymbol{x}), \boldsymbol{y}(\boldsymbol{x}')) = \rho(\boldsymbol{x}, \boldsymbol{x}'; \boldsymbol{l}).$$

Here $\boldsymbol{l}$ is a vector of parameters for ρ that govern the correlation between responses in each coordinate dimension. The most common choice for the correlation function is the squared exponential (SE) kernel given in equation (3.1.1). Here the l_js allow control the degree of the correlation and are constrained to be positive. The true function in figure 3.1 is an example realisation from a GP with zero-mean and SE kernel;

$$\rho(\boldsymbol{x}, \boldsymbol{x}'; \boldsymbol{l}) = \prod_{j=1}^{d} \exp\left\{\frac{-|x_j - x'_j|^2}{l_j}\right\}. \tag{3.1.1}$$

In equation (3.1.1) we have the separable form of the SE kernel. If it is believed that the function is isotropic we can set all l_j to the same value l, therefore making the correlation between two points only rely on their Euclidean distance.

Another property of the stationary covariance model is that the correlation between observations at any two points relies only on the distance between those points in the input space. Other common examples of kernels that lead to stationarity are the Matern kernel and the periodic kernel [68]. If instead, we believe that the correlation between two process values should depend on their location in the input space, the GP is called non-stationary. An example of a non stationary kernel is the polynomial kernel;

$$\rho(\boldsymbol{x}, \boldsymbol{x}'; p) = (\boldsymbol{x} \cdot \boldsymbol{x}')^p. \tag{3.1.2}$$

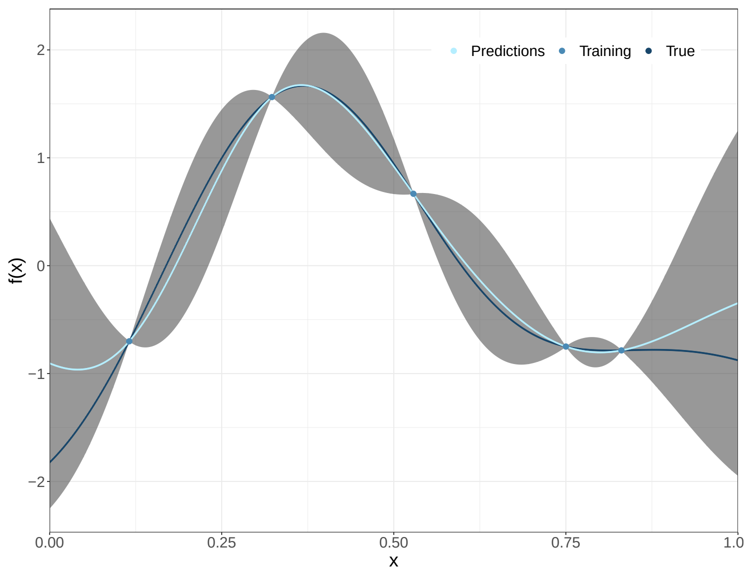

Figure 3.1: One-dimensional GP model fit to five training points (middle blue). Model predictions (light blue) along with 95% uncertainty bounds (grey) are given. The test function (dark blue) used is a random draw from a zero-mean GP with SE kernel.

Here, we see that the correlation relies on the inner product between the two input variables and is therefore not invariant to translation. In this work we implement GPs with a constant mean and the separable SE kernel.

Having specified the mean and correlation functions, we can turn our focus to estimating the model parameters and hyperparameters. The likelihood is given in equation (3.1.3). Parameters can be estimated through maximum likelihood or Bayesian methods. Either way we run into a drawback of the GP model - the requirement of inverting the $n \times n$ correlation matrix $\boldsymbol{R}$, whose computational time scales with $\mathcal{O}(n^3)$. Due to this, GP models are preferred when the size of the data is not too large (currently $n < 20,000$).

$$\mathcal{L}(\boldsymbol{y}|\boldsymbol{X};\sigma^2,\boldsymbol{l}) = \frac{1}{(2\pi\sigma^2)^{n/2}|\boldsymbol{R}|^{1/2}} \exp\left\{ -\frac{(\boldsymbol{y}-\boldsymbol{\mu})^T\boldsymbol{R}^{-1}(\boldsymbol{y}-\boldsymbol{\mu})}{2\sigma^2}\right\} \tag{3.1.3}$$

Once we have estimated parameters we can consider prediction at new input values $\boldsymbol{x}^*$. Again, $\{\boldsymbol{y}, y(\boldsymbol{x}^*)\}$ follow a MVN distribution, so we can derive the predictive distribution as the conditional MVN distribution;

$$y(\boldsymbol{x}^*)|\boldsymbol{y} \sim MVN(\hat{\mu}(\boldsymbol{x}^*), \hat{\Sigma}(\boldsymbol{x}^*)). \tag{3.1.4}$$

Here the mean and covariance are given by the conditional MVN formulas in equations (3.1.5) and (3.1.6); these are sometimes called the kriging mean and variance due to their use in geostatistics literature [42].

$$\hat{\mu}(\boldsymbol{x}^*) = \hat{\boldsymbol{\mu}} + \hat{\boldsymbol{r}}(\boldsymbol{x}^*)^T\hat{\boldsymbol{R}}^{-1}(\boldsymbol{y}-\hat{\boldsymbol{\mu}}) \tag{3.1.5}$$

$$\hat{\Sigma}(\boldsymbol{x}^*) = \hat{\sigma}^2\Big[1 - \hat{\boldsymbol{r}}(\boldsymbol{x}^*)^T\hat{\boldsymbol{R}}^{-1}\hat{\boldsymbol{r}}(\boldsymbol{x}^*)\Big] \tag{3.1.6}$$

Here, $\hat{\boldsymbol{r}}(\boldsymbol{x}^*)$ is the vector of correlations between previously sampled points and a new point of interest. In the maximum likelihood context it is common to use the mean of the predictive distribution as our predictions for $y(\boldsymbol{x}^*)$. Looking at the second term in equation (3.1.5), we see that this mean is a weighted sum of the training data, where the weights are a function of the correlations between training and testing points.

Figure 3.1 illustrates a one-dimensional GP emulator. Five training data points (middle blue) are sampled from the true function (dark blue) and used to construct the GP emulator; predictions (light blue) given by equation (3.1.5) are presented with

95% prediction intervals (grey). Here we see that predictions at a new location $\boldsymbol{x}^*$ far away from the training data lead to high predictive variance (particularly near the edges). And, as $\boldsymbol{x}^*$ approaches previously sampled points the variance decreases until we have absolute certainty of the simulator output where have run it. This behaviour can be identified in equation (3.1.6) where the values in $\hat{\boldsymbol{r}}$ increase as $\boldsymbol{x}^*$ gets closer in distance to inputs in $\boldsymbol{X}$, causing greater reduction from estimated process variance $\hat{\sigma}^2$. This leads to the characteristic football shaped intervals associated with GPs.

3.2 Bayesian Treed Gaussian Processes

For emulation of deterministic computer models, a common issue is modeling simulators that are not well represented by a stationary model. TGP is a method developed by Gramacy and Lee [29] to tackle this issue. The approach has the added benefit in reducing the computational demand of inverting the correlation matrix and fitting models to data which have multiple regimes of noise. This is done by combining Bayesian regression trees [16] and GPs – that is, the input space is partitioned into regions in which independent stationary GPs are fit. This results in each GP being fit to less data in each partition, minimizing the computational cost, and adding flexibility to the overall model.

Treed partitioning is the mechanism which allows TGP to be more efficient and flexible. Classification and regression trees (CART) were first developed by Breiman et al. [9]. This model makes recursive binary splits to the input space on single values of predictors. The result is a tree like structure that begins with a stump (the whole predictor space) and ends with leaves/terminal nodes; splits that occur between the two are know as internal nodes. In the case of CART, a constant is usually fit to each leaf of the tree, $\mathcal{T}$. Chipman et al. [16] later adapted the model by fitting separate linear models in each leaf and altering the specification of the model so that it is parametric. This allows for a hierarchical Bayesian analysis of the tree and the models within its terminal nodes.

Growing a regression tree in the Bayesian context requires placing a prior distribution on the size of the tree, i.e. we put a distribution on the class of trees that do well in the context of the presented data. Gramacy and Lee maintain the approach of Chipman et al. [15] in their prior distribution specification for the tree, with a few alterations. An important function of this prior distribution is to control the minimum amount of data points in the leaves of the tree. The tree-generating process starts with a stump and probabilities are assigned to both the choices of whether to split and where to split each subsequent node. Given a tree, the model then fits a GP in each terminal node. All parameters and hyperparameters are given priors and the posterior is explored with a combination of Gibbs and Metropolis-Hastings (MH) sampling.

Chipman et al. [15] proposed methods to modify the current tree; grow, prune, swap, and change. This is where Gramacy and Lee deviates a little from Chipman et al. as they add an extra method of altering the tree with what is called a rotation; this makes proposed trees more diverse and therefore allows for better mixing. Both the swap and change modifications do not effect the number of leaves in a tree, but the others do and therefore either increase or decrease the dimension of the parameter space for GP estimation. So, as the tree is grown, GP parameters are estimated, and in the end we have a posterior for both the tree and the GPs.

Once the posteriors have been sampled, prediction for the TGP model is straight forward and can be done in two main ways. First, we can take the maximum a posteriori (MAP) tree and apply the kriging equations ((3.1.5) and (3.1.6)); resulting in a model that is discontinuous at the boundaries of the trees regions and does not fully capture uncertainty. The second prediction method is to sample from the posterior of the trees and parameters for the GPs, fully capturing uncertainty in the predictive distribution and smoothing the surface at boundaries.

3.3 Bayesian Adaptive Spline Surfaces

BASS is a Bayesian adaptation of the multivariate adaptive regression splines (MARS) model [23]. The BASS model builds upon MARS making a few key changes that stabilize computation of the model. The original intent of the BASS model is to emulate complex computer codes, while allowing for sensitivity analysis and uncertainty quantification; especially in the context of high input dimension and larger data sets. Due to the piecewise nature of MARS based models they are not actually an interpolator of the simulator; however in certain contexts they provide a good alternative to GPs. We choose to incorporate this model into our comparison to show that comparisons can be done with non-GP based emulation methods.

MARS is a nonparametric regression method that fits a predictor of the form

$$\hat{y}(\boldsymbol{x}) = \sum_{m=1}^{M} a_m B_m(\boldsymbol{x}), \tag{3.3.1}$$

where there are M basis functions, $B_m(\boldsymbol{x})$, with a coefficients a_m. If we were to make the basis functions as indicators for regions in the input space the model would be similar to regression trees, in that the resulting model includes splits on inputs and a constant is used to predict unknown values of the response. The two main differences in the MARS model is the choice of basis function being linear and allowance of interactions between the effects of inputs.

The basis functions used in MARS are step-wise linear, with reflection points on single input values - these are known as knots,

$$B_m(\boldsymbol{x}) = \begin{cases} 1, & i = 1 \\ \prod_{k=1}^{K_m} \left[s_{km}(x_{v_{km}} - \ t_{km}) \right]_+, & i = 2, 3, \ldots \end{cases} \tag{3.3.2}$$

Here, t_{km} is the value on which the knot occurs on $x_{v(km)}$, where $v(km)$ indicates the input dimension of x while also constraining each input to only occur once in the basis. The term K_m defines the number of splits in the basis, $s_{km} \in \{-1, 1\}$ is a sign indicator, and $[\cdot]_+ = \max(0, \cdot)$. If a smoother prediction surface is desired, higher order splines can be used in the bases.

Knots, and therefore interactions, are chosen in a forward stepwise manner such that each newly included term minimizes the generalized cross-validation error. This is done until a predetermined number of knots is reached, resulting in a model that is intentionally overfit to the training data set. To improve prediction accuracy the algorithm then removes (or prunes) knots that are deemed to not significantly affect the prediction accuracy of the model.

The Bayesian implementation of MARS, also known as BMARS, places a probability distribution over the space of all possible MARS models. This is done by treating all unknown parameters as random and assigning prior distributions. An essential part of the prior specification is that on the number of basis functions included in the model M; placing preference on models with only a few basis functions. Exploration of the posterior distribution proceeds from an initial MARS model by adding, deleting, or modifying a spline. Further specification of prior distributions and Monte Carlo implementation can be found in Denison et al. [19].

The BASS model makes some changes to BMARS in order to improve posterior exploration of the model space. The first change is to the basis functions of the model:

$$B_m(\boldsymbol{x}) = \begin{cases} 1, & i = 1 \\ \prod_{k=1}^{K_m} g_{km}\Big[s_{km}(x_{v_{km}} - \ t_{km})\Big]_+, & i = 2, 3, \ldots \end{cases} \tag{3.3.3}$$

Here the added coefficient, $g_{km} = (s_{km} + 1)/2 - s_{km}t_{km}$, allows for the basis function coefficients to be on the same scale resulting in improved posterior sampling. BASS also introduces parallel tempering to improve mixing over the modes of the posterior in the MCMC scheme. Further information about prior distribution specification and Monte Carlo implementation can be found in Francom and Sansó [23].

Due to the fully Bayesian approach, we can utilize predictions from each MARS function contained in the posterior to get a predictive distribution for the simulator output at new $\boldsymbol{x}^*$. An example one-dimensional BASS model fit can be seen in figure 3.2, where we see the piecewise linear nature of the emulator.

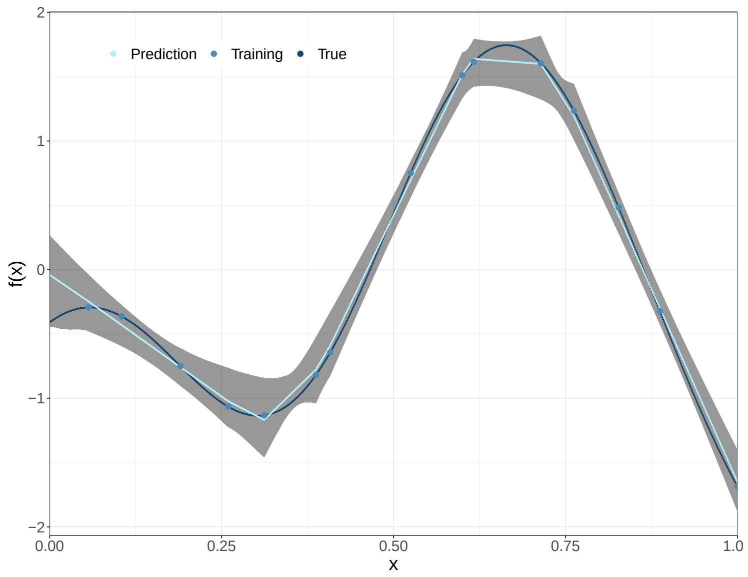

Figure 3.2: BASS model fit to 15 training points (middle blue). Posterior mean of predictions (light blue) along with 95% uncertainty bounds (grey) are given. The test function (dark blue) used is a random draw from a zero-mean GP with SE kernel.

Chapter 4

Design of Computer Experiments

In this chapter we describe five methods for designs of computer experiments; random uniform designs [60], rLHDs [49], maximin LHDs [36], s-optimal LHDs [61], and maximum projection (MaxPro) designs [2]. This is by no means an exhaustive list of possible designs, but will allow us to demonstrate how to compare and contrast designs. Here, we will be considering designs of size n on the unit hypercube $[0, 1]^d$. Each column of the $n \times d$ design matrix represents factors or inputs for the simulator. While each row corresponds to a set of input values for a single trial.

4.1 Random Uniform Designs

Acting as a baseline, one can select n design points at random from the design region. This is done by selecting n design points from a d-dimensional uniform distribution on the unit cube $[0, 1]^d$. While this may seem an obvious choice, it cannot ensure that the design is space-filling. In fact, there is a high probability that a design obtained in this way may lead to clumps of observations in the design region (i.e. there will be regions containing high densities of points, leaving other regions relatively unexplored). Hence, other designs are favoured in the computer experiment literature.

4.2 Latin Hypercube Designs

The use of LHDs was introduced as a way to ensure random sampling throughout the entire input space whilst avoiding the clumping of purely random designs. This is done by splitting the design space into n^d cubes and methodically choosing which cubes to place the n design points in.

To do this we first create a Latin hypercube $\boldsymbol{L} = (l_{ij})$ by splitting each input of the d-dimensional space into n equally spaced partitions. Each column in $\boldsymbol{L}$ is then given some random permutation of the numbers $1, \ldots, n$, placing the design points into one of the partitions. The setting for the input is placed within the interval, e.g. the midpoint of the interval or a randomly selected value in the interval; the latter is known as a randomized LHD. To change $\boldsymbol{L}$ into a design $\boldsymbol{X}$, we map each column to the unit interval:

$$\boldsymbol{x}_j = \frac{l_{ij} - \min(\boldsymbol{l}_j)}{\max(\boldsymbol{l}_j) - \min(\boldsymbol{l}_j)}. \tag{4.2.1}$$

LHDs became favoured in the design of computer experiments field due to theoretical reduction of variance in numerical integration when compared to random uniform designs [49]. Creating a design in this way also guarantees one-dimensional stratification, ensuring that each stratum has one sample [45]. This may be especially desirable if the simulator is believed to be additive.

However, not all LHDs are equally good because, while maintaining 1-d stratification, the space-filling properties in two or more dimensions can be quite poor. For example, if we consider a two dimensional input space, a valid LHD is one in which all design points fall along the diagonal of the two-dimensional grid. This design clearly leaves large regions in the design space unexplored, and would lead to poor performance by emulators. For this reason other methods of constructing designs have been developed to better ensure the desired space-filling property.

4.3 Maximin Latin Hypercube Designs

Maximin designs gained traction after Johnson et al. [36] showed that under certain assumptions maximin designs are asymptotically D-optimal, i.e. optimal with respect to minimizing the variance found in equation (3.1.6). These designs attempt to spread the points throughout the input region by maximizing the minimum distance between them. Maximin LHDs [54] combine the design method of the previous section with maximin distance designs by choosing the design that optimizes the maximin distance criteria among the class of LHDs. Doing this tackles problems that arise when using each design method on its own. Namely with LHDs, there is no guarantee that the design will effectively fill the design space. And, the projections of maximin designs onto each input tend to include clumps.

Conceptually maximin distance designs are quite straight forward and are by definition space-filling. It's all in the name; first we need to define some measure of distance between the points in the design, then the smallest distance between two points in the design is made to be as large as possible. So, if Euclidean distance is used

$$d(\boldsymbol{x}_i, \boldsymbol{x}_j) = ||\boldsymbol{x}_i - \boldsymbol{x}_j|| = \sqrt{\sum_{k=1}^{d} (x_{ik} - x_{jk})^2}, \tag{4.3.1}$$

we want to find the design $\boldsymbol{X}$, such that

$$\boldsymbol{X} = \text{argmax}_{\boldsymbol{X}} \min \Big\{ d(\boldsymbol{x}_i, \boldsymbol{x}_j) \ : \ i \neq j = 1, \ldots, n \Big\}. \tag{4.3.2}$$

A good intuition for this design was given in Pronzato and Müller [57] which likened maximin designs to setting up tables in a restaurant so that we minimize the chance of eavesdropping between parties. To fulfill this requirement, this type of design tends to place points on the borders of the region. While the concept for this design is quite simple, the required $n \times d$ optimization can be quite difficult.

There are many available algorithms for finding maximin LHDs. Morris and Mitchell [54] use a simulated annealing algorithm which is based on the well known Metropolis algorithm. Since their initial paper many other algorithms have been proposed, including but not limited to: Van Dam et al. [65] who use a branch-and-bound algorithm, Kenny et al. [41] who propose a columnwise-pairwise algorithm, and Chen et al. [13] who use a particle swarm algorithm.

The literature contains numerous alternatives to using maximin distance criteria. For example minimax designs [45], which seek to minimize the maximum distance between the design and other points in the input space, orthogonal array-based LHDs [64], and nearly orthogonal designs [7]. The performance of an emulation method for a given simulator may depend on the geometric criteria used. Thus, one should not limit their consideration to any one method but attempt to cater their design to the problem at hand.

4.4 S-Optimal Latin Hypercubes

There are many variants of space-filling designs. Here we investigate s-optimal designs [61]. Like the maximin LHDs of the previous section, the s-optimal design method starts with an initial LHD but instead minimizes the inverse of the squared pairwise Euclidean distances amongst all design points.

The criterion used by Stocki, sometimes referred to as the Audze-Eglais criterion, was first developed by Audze [1].

$$G(\boldsymbol{X}) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{1}{||\boldsymbol{x}_i - \boldsymbol{x}_j||^2}. \tag{4.4.1}$$

To find the optimal design the function G in equation (4.4.1) is minimized. Doing this maximizes the mean distance between each point and all other points, effectively placing the design points as uniformly as possible throughout the design space. The analogy given to justify this metric is based in the physics of charged particles exerting repulsive forces on each other; if the force is proportional to the inverse of their squared distances, they will reach maximal potential energy when the function in equation (4.4.1) is minimized [4].

Like maximin LHDs, the s-optimal LHD method creates designs that have favourable spread in the d-dimensional design space, while also holding the uniformity property in its one-dimensional projections. However, with the design methods considered until now, the space-filling properties into other lower-dimensional spaces of the design region have not been considered. In some contexts, not all inputs of a simulator have an effect on its output – it may beneficial to fill the space of the active inputs. This idea is maintained by the MaxPro designs of the next section.

4.5 Maximum Projection Designs

MaxPro designs were introduced by Joseph et al. [38] with the goal of further improving the projection properties of maximin LHDs. The method aims to generate designs that are not only space-filling in the input space, but also space-filling in its projections on to the subspaces of dimension $2, \ldots, d-1$. To do this, a weighted

Euclidean distance is defined to facilitate measuring distance in the subspaces;

$$d(\boldsymbol{x}_i, \boldsymbol{x}_j; \boldsymbol{\theta}) = \left\{ \sum_{k=1}^{d} \theta_j (x_{ik} - x_{jk})^2 \right\}^{1/2}. \tag{4.5.1}$$

If an input is part of the subspace of interest we set $\theta_j = 1$, otherwise $\theta_j = 0$. In the full design space we treat the θ_js as measures of importance; setting $\theta_j \in [0, 1]$ and $\sum_{i=1}^{d} \theta_j = 1$. Substituting this distance into the maximin distance from equation (4.3.2) gives the following criteria:

$$\underset{\boldsymbol{X}}{min} \quad \phi_p(\boldsymbol{X}; \theta) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} d(\boldsymbol{x}_i, \boldsymbol{x}_j; \boldsymbol{\theta})^{-p}. \tag{4.5.2}$$

To deal with the unknown true importance of each input Ba and Joseph [2] adopt a Bayesian approach, placing an equal probability prior distribution over $\boldsymbol{\theta} = \{\theta_1, \ldots, \theta_d\}$. So, we now seek the design that minimizes the expectation of $\phi_p(\boldsymbol{X}; \boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$. Setting $p = 2d$ allows for an analytic solution when integrating out $\boldsymbol{\theta}$, and the final form of the criteria for the design becomes;

$$\underset{\boldsymbol{X}}{min} \quad \psi(\boldsymbol{X}) = \frac{1}{\binom{n}{2}} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{\prod_{l=1}^{d} (x_{il} - x_{jl})^2}. \tag{4.5.3}$$

A couple of properties of MaxPro designs can be deduced from equation (4.5.3). First, each point in the design must have unique values for each input, otherwise (4.5.3) goes to ∞. Second, the product in the denominator ensures preference for dispersed points in the projections of the design in all subspaces.

The algorithm to generate MaxPro designs takes advantage of its similarities to maximin LHDs. First, Ba and Joseph use the simulated annealing algorithm developed by Morris and Mitchell [54] to generate a LHD. The resulting design is then optimized with respect to the maximum projection criterion with a derivative based optimization algorithm.

Chapter 5

Performance Metrics

Below we outline the performance metrics that we propose for use when comparing and evaluating emulation methods or design methods with test functions found in the VLSE. The metrics included fall into two categories; measures of predictive accuracy and measures of prediction interval performance. This is not a comprehensive list of possible metrics; others were investigated, but were either found to be non-informative or highly correlated with the included metrics. For more possible performance metrics we refer the reader to Bastos and O'Hagan [3]. When comparing emulators it may also be of interest to include run time as a comparative metric.

5.1 Root Mean Squared Prediction Error

A benefit of using fast test functions is that the simulator output is readily available to create training and validation sets. It is common to evaluate the predictive ability of a model by comparing predictions with the output from the model. One metric that compares model outputs at predictive locations x^* to predictions from the emulator is the root mean squared prediction error (RMSPE).

$$RMSPE = \sqrt{\frac{1}{n} \sum_{\boldsymbol{x}^* \in \mathcal{X}_{pred}} (y(\boldsymbol{x}^*) - \hat{y}(\boldsymbol{x}^*))^2} \tag{5.1.1}$$

This can be interpreted as roughly the average distance between the predictions and true model output. Since the value of the RMSPE is dependent on the range of the response it cannot be directly compared across different test functions but can still be used for ranking of methods for each test function.

5.2 Nash-Sutcliff Efficiency

The Nash-Sutcliff efficiency (NSE) first appeared in literature for the modeling hydrological experiments with the intent of providing a basis for discussion of modelling techniques [55].

$$NSE = 1 - \frac{\sum_{\boldsymbol{x}^* \in \mathcal{X}_{pred}} (y(\boldsymbol{x}^*) - \hat{y}(\boldsymbol{x}^*))^2}{\sum_{x^* \in \mathcal{X}_{pred}} (y(\boldsymbol{x}^*) - \bar{y}(\boldsymbol{x}^*))^2} \tag{5.2.1}$$

It is bounded by 1 above and $-\infty$ below and is comparable to R^2 in the typical linear regression setting. That is, if we look at the second term in the formula we see that it is a measure of the proportion of the variance in the response explained by the emulator predictions. Also, like R^2, NSE values close to 1 indicate that the emulator has performed well in terms of prediction accuracy. Observing a NSE below 0 indicates that the predictions from the emulator perform worse than using the mean of the response as a predictor.

5.3 Empirical Coverage Probability of Prediction Intervals

We now move from purely predictive measures of performance to those that rely on the variability of the predictions. When using any of the emulation methods of chapter 3, $100(1-\alpha)\%$ pointwise prediction intervals can be found. The empirical coverage probability for these intervals over a set of prediction sites is given by:

$$ECP(\boldsymbol{x}^*; \alpha) = \frac{1}{m} \sum_{\boldsymbol{x}^* \in \mathcal{X}_{pred}} \mathbf{1}\Big[y(\boldsymbol{x}^*) \in I(\boldsymbol{x}^*; \alpha)\Big] \tag{5.3.1}$$

Where $\mathbf{1}[\,]$ is the indicator function and $I(\boldsymbol{x}^*; \alpha)$ is the $100(1-\alpha)\%$ prediction interval at $\boldsymbol{x}^*$.

On average, one would like the empirical coverage probability to be close to the theoretical coverage probability. This is an essential check as we want to ensure that for future emulation of computer experiments the reported prediction intervals are reliable.

5.4 Mean Relative Interval Length

Due to the nature of prediction intervals (e.g excessively large intervals will capture the truth 100% of the time) we suggest looking at the length of the prediction intervals along with the ECP. We use the mean relative interval length (RIL), given by:

$$L_I(\boldsymbol{x}^*;\alpha) = \frac{1}{m} \sum_{\boldsymbol{x} \in \mathcal{X}_{pred}} \frac{U(\boldsymbol{x}^*;\alpha) - L(\boldsymbol{x}^*;\alpha)}{|\hat{y}(\boldsymbol{x}^*)|}. \tag{5.4.1}$$

Here $U(\boldsymbol{x}^*;\alpha)$ and $L(\boldsymbol{x}^*;\alpha)$ are defined to be the upper bound and lower bound of the $100(1-\alpha)\%$ prediction interval, respectively. This form of prediction interval length is recommended as it scales the interval by the magnitude of the response value, allowing for direct comparison of the interval length over different test functions and multiple test sets.

With the performance metrics defined we now turn our focus to comparing the emulation and design methods outlined in previous chapters.

Chapter 6

Comparison of Designs and Emulators

6.1 Method for Comparison

We now outline the proposed method to compare and evaluate the performance of design and emulators for computer experiments using the suite of test functions found in the VLSE. The approach for both contexts is very similar; create many training datasets and large validation sets, apply the methods of interest, then compute performance metrics. This allows for a robust way of making comparisons which can be applied to any situation that a practitioner or researcher may face.

When considering the statistical surrogate the primary interest is to account for the variability in performance metrics caused by the choice of design through accounting for both design size and design type. For design type we suggest using multiple LHDs. The reason for this is two-fold; first, for some simulators the choice of design may have a greater effect on the performance of emulation methods than for others; second, randomized LHDs should capture the range of good and bad designs that may be used by experimenters.

Implementations of the design methods considered in this work (and many others) are not deterministic, producing equivalent but distinct designs which result in varying emulator performance. Therefore the goal of the method for comparison of designs differs slightly from that for comparing emulators - that is, creating data for comparison which captures the aforementioned variation in performance metrics whilst controlling for variation in performance that is not due to the designs themselves. In this work we consider designs when the emulator of choice is a GP. Of course, the same ideas will apply if a different emulation approach is used.

To facilitate the comparison, test functions found in the VLSE are fast running so that the user can produce many training sets and large test sets per test function for the comparison. A key for a practitioner is that the test functions should represent the function space in terms of complexity and dimensionality. For a creator of a new design method the goal will be to explore the settings where their design improves on established methods and where the method is less successful; it may be that a certain design fills the input space better for additive functions than for multiplicative functions. Thus, the effect of design on performance for a given setting can be explored by selecting appropriate test functions from the VLSE.

Similarly, it may also be of interest in the comparison of emulators to control for the variation in performance metrics that is due to the type of test function; e.g if we have additive functions, multiplicative functions, or functions with multiple regimes, etc. we may find that some methods do better at emulation in certain settings. An example of this is the Treed Gaussian Process (TGP), which was created with the intent of performing well in contexts where a stationary Gaussian Process (GP) does not adequately capture the functional response of a computer experiment. Consequently, it may be of interest to then stratify the comparisons by function type – this may be informed by users own problem.

With the simulation context specified; the next thing to consider when comparing designs and emulators is the run size and number of factors. In practice, one typically has a limit to the number of runs they can make of the simulator. For a practitioner this may be predetermined, thus constraining the scope of the comparison. But, to assess relative performance of methods in general, one should ideally consider many design sizes ranging from sparse to dense. We suggest that for each test function - design size combination a sufficient number of designs, n_D, are produced and responses obtained for each design size. The choice of n_D is somewhat arbitrary; the intent is to generate enough designs so that distributional comparisons of performance metrics can be performed later. The user can train their emulator(s) on each of the designs and compute performance metrics on the accompanying test sets.

Algorithm 1 lays out the process for generating comparison data for emulators. The key here is to avoid extrapolation; we want to compare the interpolation abilities of the emulators, nothing else. Before fitting models it is good practice to scale the

Algorithm 1 Data Generation for Emulator Comparison

```
1. Choose a suite of test functions to use for comparison.
2. Choose the set of design sizes to be used.
3. For each test function - design size combination create $n_d$ sets of training data
   and test data, insuring that the test set design lies within the training design.
     • For each design $(\boldsymbol{X}_1, \ldots, \boldsymbol{X}_{n_d})$, train each model and compute performance
       diagnostics on the test set.
```

inputs to lie on the unit interval and standardize the response values by subtracting by their mean and dividing by the standard deviation. Performance metrics can be computed on this scale.

Algorithm 2 then outlines our proposed procedure for producing data for the comparison of experimental designs. It differs from the emulator algorithm in two main aspects; first, since the space-filling abilities of the designs are of interest and the same emulator will be employed, predictions outside of any specific design are allowed. Second, the inputs and outputs should still be scaled before fitting the models, but performance metrics should be computed on the original scale of the data as the scaling done for each design method will naturally use different values.

Once the comparison data is created evaluation of performance on single test functions can be done by comparing metric distributions for the design types visually as seen in the following sections. If desired, one can perform a hypothesis test for a difference between the median of the distributions using an ANOVA or Kruskal-Wallis test depending on perceived adherence to the normality assumption of the ANOVA. Though we recommend to not make conclusions about relative performance of designs based on single test functions.

Comparing results across test functions is less straight forward but is the primary concern of this work. To facilitate this comparison we first introduce a couple measures of relative performance between the distributions of performance metrics of each design or emulation method. The first measure; the proportion of times a method performs worse than the best median, p_{worse}. Here the 'best median' is defined to be either the minimum or maximum (depending on the metric) median of the distributions associated with the compared methods. It can be seen that $p_{worse} \in [0.5, 1]$, such that its value associated with best performing method (according to the median of

Algorithm 2 Data Generation for Design Comparison

```
1. Choose a suite of test functions to use for comparison.
2. Choose the set of design sizes to be used.
3. For each test function - design size combination create $n_d$ sets of training data
   for each design type of interest and a test set.
   (a) For each design $(\boldsymbol{X}_1, \ldots, \boldsymbol{X}_{n_d})$, build a GP model, scaling each $\boldsymbol{x}_j \in [0, 1]$,
       $j = 1, 2, \ldots, d$ and centering the response by its mean and scaling by its
       standard deviation.
   (b) Make predictions and compute performance metrics on the original scale
       of the response.
```

its distribution) is 0.5. This proportion can be found for each test function and the resulting distributions can be used to compare the methods.

The next measure aims to balance rewarding a method for performing well with penalization for performing relatively poorly. To do this we first compute the average reverse rank of each method, $\bar{r}_m$; here m indicates the method. This is done by finding the reverse ranking of the methods for each iteration of the design comparison, then taking the mean over all iterations. This average is then divided by p_{worse}, resulting in the scaled average reverse rank:

$$SRR_m = \bar{r}_m / p_{worse}. \tag{6.1.1}$$

SRR_m reaches its maximum of two times the number of methods being compared when a method always performs best, i.e. when the average reverse rank of method m is equal to the number of methods being compared and p_{worse} is equal to 0.5. The minimum of SRR_m is 1 and occurs in the opposite case.

Note that the above two measures are relative. When performing a comparison it is imperative that one considers whether the differences seen have a practical effect on the analysis; rather than being different in the sense of error chasing or choosing a best poor performer. We therefore advise that these measures are always paired with thorough examination of the performance of the methods for individual test functions. The next section provides an illustration of this process for both contexts.

6.2 Emulator Comparison

We now go through the exercise of comparing the emulation methods in chapter 3 to illustrate the use of the VLSE. We implement the emulators on twenty test functions and consider design sizes of $n = 5d,\ 10d,\ 20d,$ and $50d$ for each. For each test function - design size combination we generate twenty training data sets using rLHDs for the design, each with an accompanying test set of size $50d$. To keep the comparison simple we do not add noise to any observations of the test functions. All simulations are done in `R`.

Implementation of GPs with a SE kernel is done with the `GPfit` package [48]. It uses a maximum likelihood for parameter estimation, making a couple reparameterizations from what is outlined in section 3.1 to aid optimization of the multimodal likelihood surface. A multi-start limited-memory Broyden-Fletcher-Goldfarb-Shanno boxed-constraints (L-BFGS-B) algorithm is used to estimate model parameters.

To implement TGPs we use the `tgp` package [30]. Default settings are used except for a few key areas. First, for MCMC we increase the number of restarts and samples to ensure convergence of the chain. We also change the nugget prior distribution for the GPs so that their purpose is solely to prevent numerical singularity when inverting the correlation matrix.

The BASS model is built with the `BASS` package [23]. Default parameters are used for the prior distributions placed on the model parameters. For MCMC settings; we increase the number of samples, include some thinning, and add temperatures to the parallel tempering to improve mixing and ensure convergence.

In the next sections we follow the steps of evaluation outlined in section 6.1; a comparison is done at the individual level then on the group level of test functions. The goal is to make recommendations as to what emulator one should proceed with when faced with a novel computer experiment. We note that three emulators is a limited number to choose from. However, that's the advantage of having a library of test functions such as the VLSE and an agreed upon process for comparison – if in the future someone wanted to put another emulation method up against these, accessing and generating data for the test functions will be quick and easy.

6.2.1 Evaluation on Individual Test Functions

Here we present some of the results for emulator comparison. For medians and interquartile ranges of the performance metrics obtained on the other test functions we direct the reader to the tables located in appendix A. We'd present those in full, but the process becomes quickly repetitive, and thus would not add much value.

First, we look at results for the two-dimensional Franke's function [25];

$$f(x) = 0.75\exp\left(-\frac{(9x_1-2)^2}{4}-\frac{(9x_2-2)^2}{4}\right)+0.75\exp\left(-\frac{(9x_1+1)^2}{49}-\frac{(9x_2+12}{10}\right)$$
$$+0.5\exp\left(-\frac{(9x_1-7)^2}{4}-\frac{(9x_2-3)^2}{4}\right)-0.2\exp\left(-(9x_1-4)^2-(9x_2-7)^2\right). \tag{6.2.1}$$

A visualization of the response surface for this function can be found in figure 6.1. Figure 6.2 contains the obtained performance metric distributions for each the three emulation methods. Within each panel of the figure the metrics are stratified by design size. Panels A and B contain the RMSPE and NSE, respectively. Conclusions from these two panels will be the same as the metrics themselves are highly related (both are functions of the squared prediction error). Observe that the GP and TGP are undoubtedly more accurate predictors of this surface when compared to the BASS model. Panels C and D contain the measures of interval performance; 90% prediction/credible intervals are used. Once the design size has reached $10d$ we see that

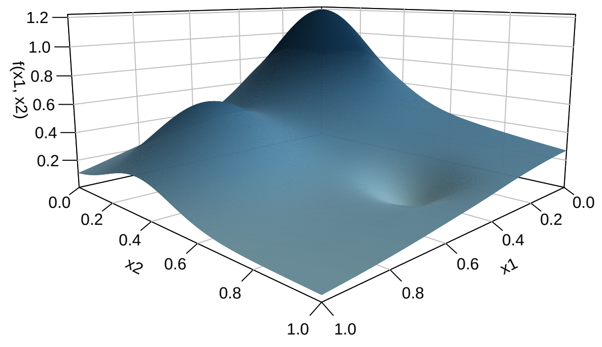

Figure 6.1: Two-dimensional Franke test function [25]. The response surface is smooth with two gaussian peaks contained in the input space.

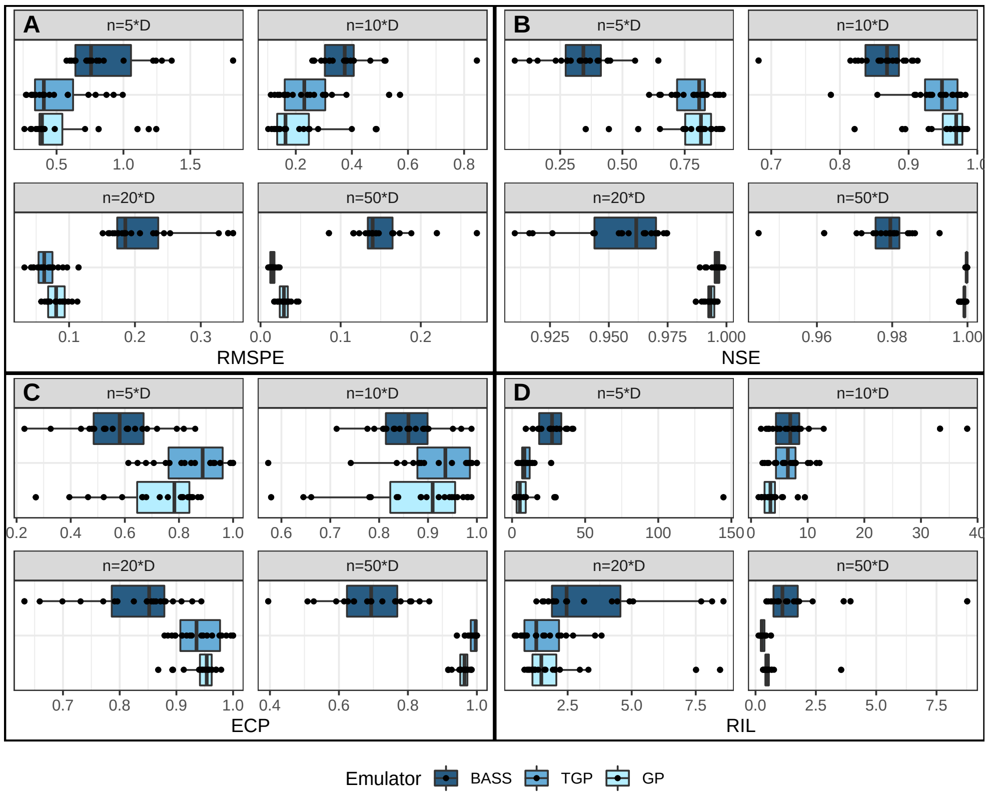

Figure 6.2: Franke's function performance metric distributions for emulator comparison with (A) RMSPEs, (B) Nash-Sutcliff efficiencies, (C) empirical coverage probabilities, and (D) relative interval lengths.

proportion of true output values that lie in their corresponding intervals for the two GP based methods agree well with the theorized coverage probabilities; the BASS model does not do well in this aspect either.

One thing to consider when making these comparisons is whether or not the observed difference in performance between emulation methods is practical. If we look at the NSE values obtained for Franke's function and consider their interpretation as the proportion of the variance in the response explained by the emulator, we notice that at $n = 10d$ the median NSE for all three methods is quite good. At $n = 20d$ and $n = 50d$ this lack of practical difference is even more exaggerated. When this occurs the practitioner might consider other factors such as run time and/or ease of implementation when choosing a statistical surrogate.

The BASS model does not perform relatively poorly in all situations. Figure 6.3 consists of boxplots for the performance metrics obtained on the corner peak integrand

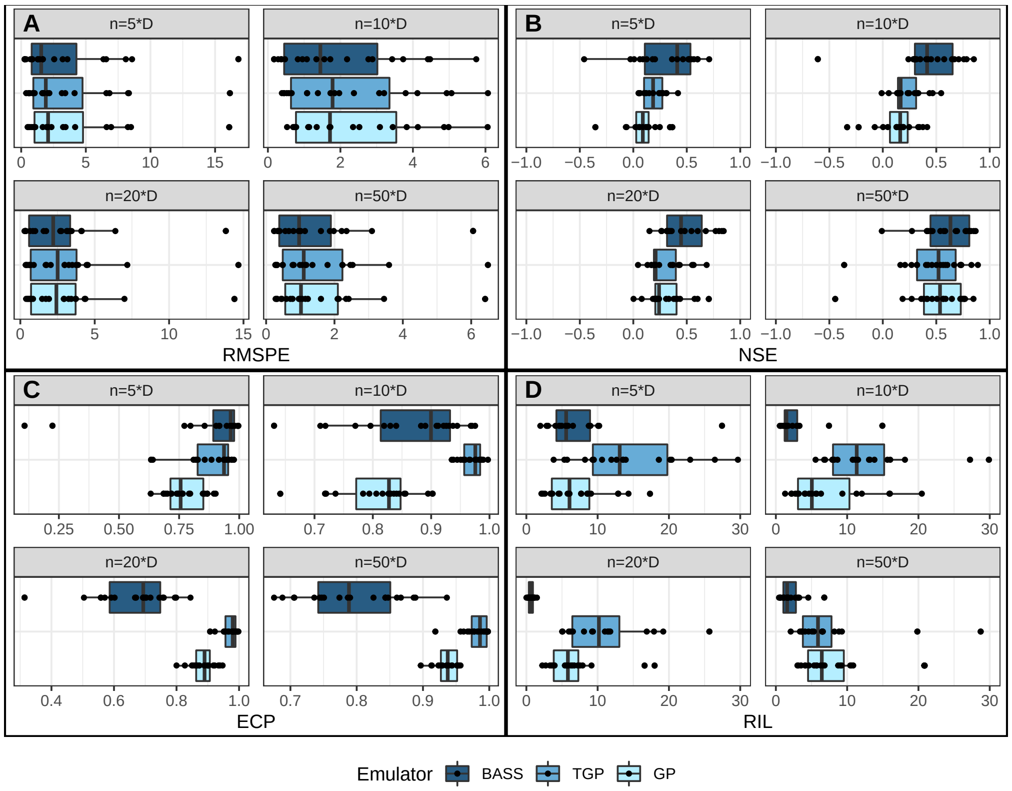

Figure 6.3: Corner Peak function performance metric distributions for emulator comparison with (A) RMSPEs, (B) Nash-Sutcliff efficiencies, (C) empirical coverage probabilities, and (D) relative interval lengths. Observations of the NSE that are less than -1 were removed from the plot to allow for better visual comparison.

function [28]. The functional form is

$$f(x) = \left(1 + \sum_{i=1}^{d} a_i x_i\right)^{-(d+1)}, \tag{6.2.2}$$

where the a_i are constants that control the prominence of a sharp peak in the corner of an otherwise flat response surface. For our simulation d is set to 3 and each a_i to 5. Looking at the RMSPE values in panel A for each design size we see that some designs do result in bad performance for all emulators. For the NSEs in panel B we removed eight outliers across the four design sizes whose NSE values were less than -1. It is observed that the BASS model has the highest median for all design sizes, i.e. it does best with respect to explaining the variation found in the response surface. We do however note that the prediction intervals (panel D) are short compared to other methods and this consequently led to poor performance with regards to ECP (panel C).

One last result we would like to make note of is the predictive performance of all three methods on the Borehole function. We point this out because of the the widespread use of the Borehole function in the computer experiments literature. In table A.1, we see that the median NSE for three methods is 0.998 (GP) 0.983 (TGP) and 0.994 (BASS) for a design size of n = 5d. This function is regularly employed to compare the predictive ability of new methods to those already established in the literature. However, looking at the NSE, we see that any improvement over a GP emulator would provide no practical gains.

6.2.2 Evaluation Over the Suite of Test Functions

After evaluating the effectiveness of the emulation methods on individual test functions we move to comparisons aggregated over test functions. First we look at the boxplots of p_{worse} found in figure 6.4. RMSPE is utilized here to focus on the predictive accuracy of the chosen emulators. Comparisons at design size of $5d$ are in the top right box. For this amount of training data the GP frequently led to the lowest median RMSPE for the investigated test functions. What is observed in the boxplots for $n = 20d$ and $n = 50d$ (left and right on the bottom of of the figure) alludes

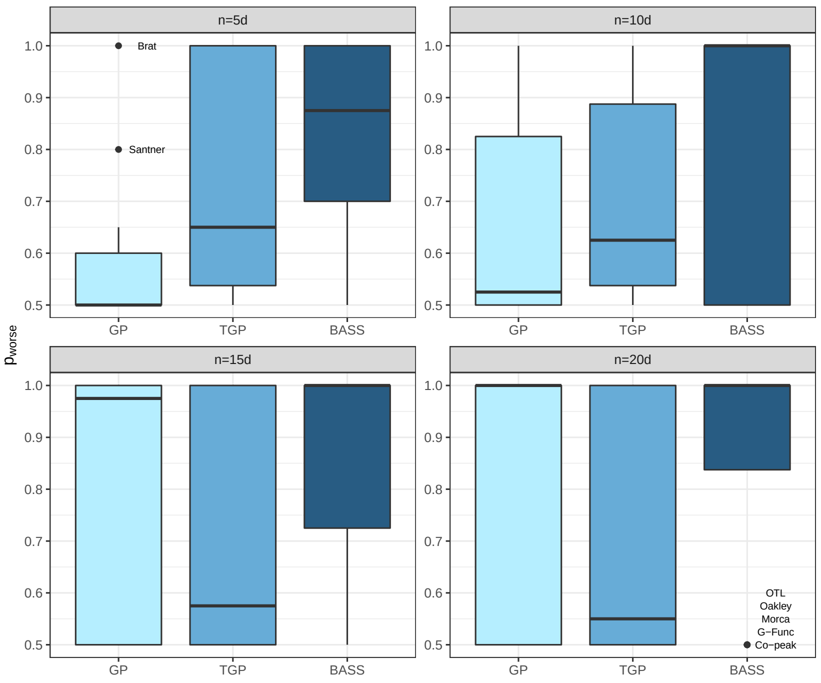

Figure 6.4: Boxplot of proportion of times a method does worse than the best median for each test function stratified by initial design size. Here RMSPE is used to compare predictive accuracy.

to emulator performance that is more disparate, as the distribution of p_{worse} for the BASS model is concentrated toward one. It also highlights the need for our second measure SRR_m, as the comparisons between the GP and TGP emulators are difficult here.

Boxplots for the scaled average reverse rank measure are contained in figure 6.5. Here $SRR_m \in [1, 6]$ as there are three methods being compared. We first dissect the relative performance of TGPs to GPs. In the simulations, TGP led to greater predictive accuracy for $n = 20d$ and $n = 50d$. However, this difference is not observed for $n = 10d$, and for $n = 5d$ the GPs outperform the TGPs. This is most likely due to the amount of information available to estimate the parameters of the model. For low n the fixed number of parameters for the GPs makes for easier estimation, whereas the modular nature of TGPs struggles in the presence of little data.

Now consider the comparative performance of BASS to the two other models. For the most part we see that it does not do as well in terms of predictive accuracy, but there are a few test functions for which it is competitive. The Bratley, corner peak, Oakley & O'Hagan, OTL, and Schwefel test functions show up as outliers for the BASS distribution of SRR_m. The response surfaces for first three of these test

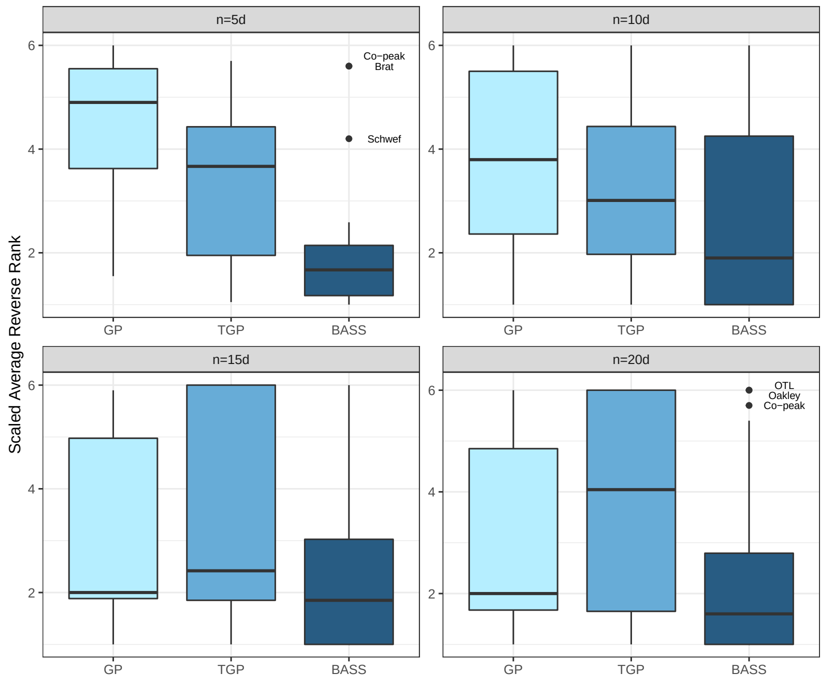

Figure 6.5: Boxplot of scaled average reverse rank measure for emulator comparisons.

functions are characterized as being predominantly flat, allowing for the less flexible BASS model to do well.

Pairing the comparative results from these two measures with results on individual test functions found in tables A.1 through A.4 leads to some key insights. While comparative differences are observed for large design sizes, most can be classified as non-practical as the three emulators capture much of the variation found in the data; this is especially true for $n = 50d$ where many of the median NSE values are one. The BASS model frequently leads to the smallest RIL, this due to the piecewise-linear nature of the model constraining the function space more than the other two methods. The result is that when BASS is a good choice for an emulator, it not only performs well with respect to predictive accuracy but also prediction interval performance.

The above results help substantiate our advice for practitioners to choose test functions which resemble their problem. For other takeaways from our comparison of these three emulators, we would suggest the use of the GP based methods. Greater predictive accuracy was achieved by TGPs for larger designs, and for smaller designs GPs performed better – the threshold in design size was not explored. Though an analysis of the interaction between emulator type and design size and type would be beneficial for the end user of these methods.

6.3 Comparison of Designs

In this section we demonstrate how to use the VLSE through a comparison of the designs outlined in chapter 3. Twenty test functions of varying types were used. Regularly, the literature for the design of computer experiments advocates for the use of ten design points per dimension of the problem [47, 12, 37, 11], so for our comparison we consider $n = 5d,\ 10d,\ 15d,$ and $20d$. All simulations are done in `R`.

For each test function and design size combination we go through the process of generating designs, building the emulators, then computing the performance metrics. Test sets are of size 1000 per dimension and are generated via randomized latin hypercube sampling. Our initial comparisons included the random uniform designs outlined in section 4.1, but we found that they performed consistently worse than all other designs – so bad that comparisons between others designs were made difficult. For this reason we choose not to include them in what follows.

Implementations for the remaining four designs use default settings. For randomized LHDs we use the `lhsDesign` function from the `DiceDesign` package in R [21]. The `lhs` package [10] is used to implement maximin LHDs and s-optimal LHDs. This maximin LHD implementation sequentially adds points to the design that satisfy the maximin criteria. The implementation for s-optimal LHDs utilizes the columnwise pairwise algorithm outlined in Stocki [61]. Lastly, implementation of maximum projection designs is done with the `MaxProLHD` function of the `MaxPro` package [2]. Next we will compare the performance of these design methods on individual test functions, then across test functions ass outlined in section 6.1. Overall, MaxPro seems to perform best most often, but there is little practical difference between design types in most cases.

6.3.1 Evaluation on Individual Test Functions

Figure 6.6 presents archetypal result from generating design comparison data for a test function. Here we have the five-dimensional Friedman test function [26]:

$$f(\boldsymbol{x}) = 10\sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5. \tag{6.3.1}$$

Panel A shows boxplots of the RMSPE for the design methods stratified by design size. Panel B shows the NSE with the same stratification. With these two metrics we see that the GP struggles to accurately capture the nature of the response surface at $n = 5d$. This is confirmed by distributions of the ECP and RIL found in panels C and D, respectively, where the relatively large prediction intervals struggle to achieve the theorized coverage probability. Moving to larger design sizes, we observe good performance in terms of prediction and interval performance. In particular, the NSE values approach their maximal value of 1 as n increases. Interestingly, MaxPro has the minimum median RMSPE and RIL and maximum median NSE and ECP for all design sizes – this was the case for many of the test functions. However, we do note that the observed differences may not be practically different.

For a couple test functions we found that the $n = 10d$ rule, and even $n = 20d$ was insufficient to build a GP emulator with adequate predictive accuracy. For example, figure 6.7 contains the distributions for performance metrics computed on the Schwefel

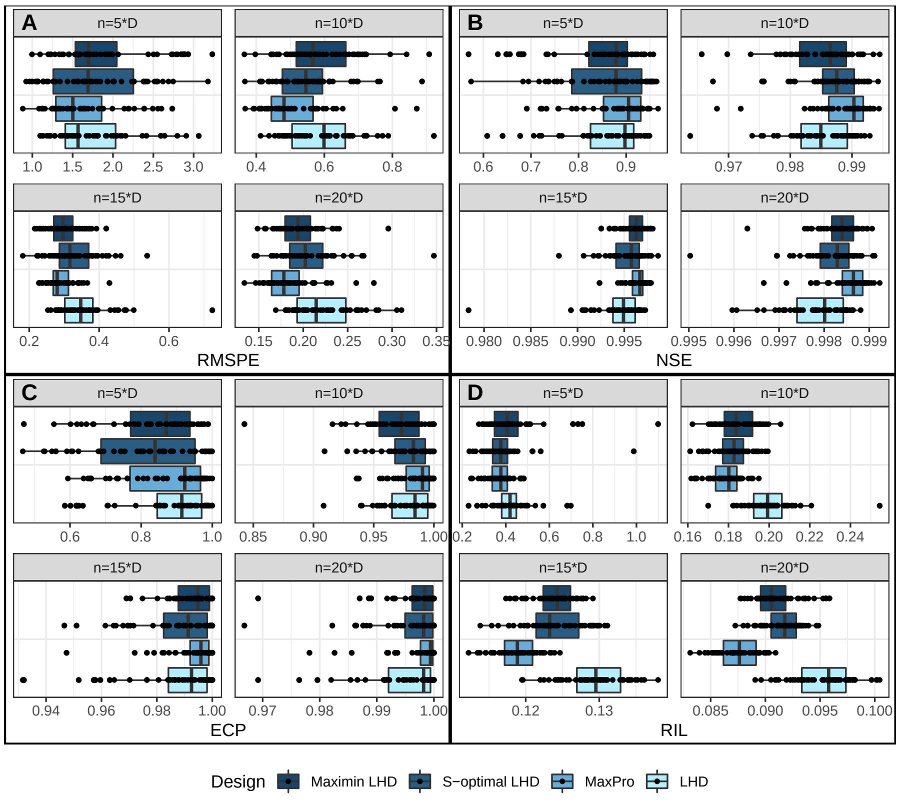

Figure 6.6: Friedman function performance metric distributions for (A) RMSPEs, (B) Nash-Sutcliff efficiencies, (C) empirical coverage probabilities, and (D) relative interval lengths.

test function;

$$f(\boldsymbol{x}) = 418.9829d - \sum_{i=1}^{d} x_i \sin\left(\sqrt{|x_i|}\right). \tag{6.3.2}$$

this is an additively separable function whose inputs $x_i \in [-500, 500]$; we set $d = 5$. The response surface of the Schwefel function is complex. Through the NSE values, we see that the emulator struggles to perform better than using the mean response value as a predictor for all design sizes used.

The other test function for which the GP emulator performed in this way, though the poor performance was not as severe, was the multiplicatively separable product peak integrand function [28] with 5 five inputs. For this test function when the design size is larger than $5d$, the s-optimal design is the only design to guarantee a majority of the NSE values are positive. For other design types the NSE distributions have long tails that reach below zero.

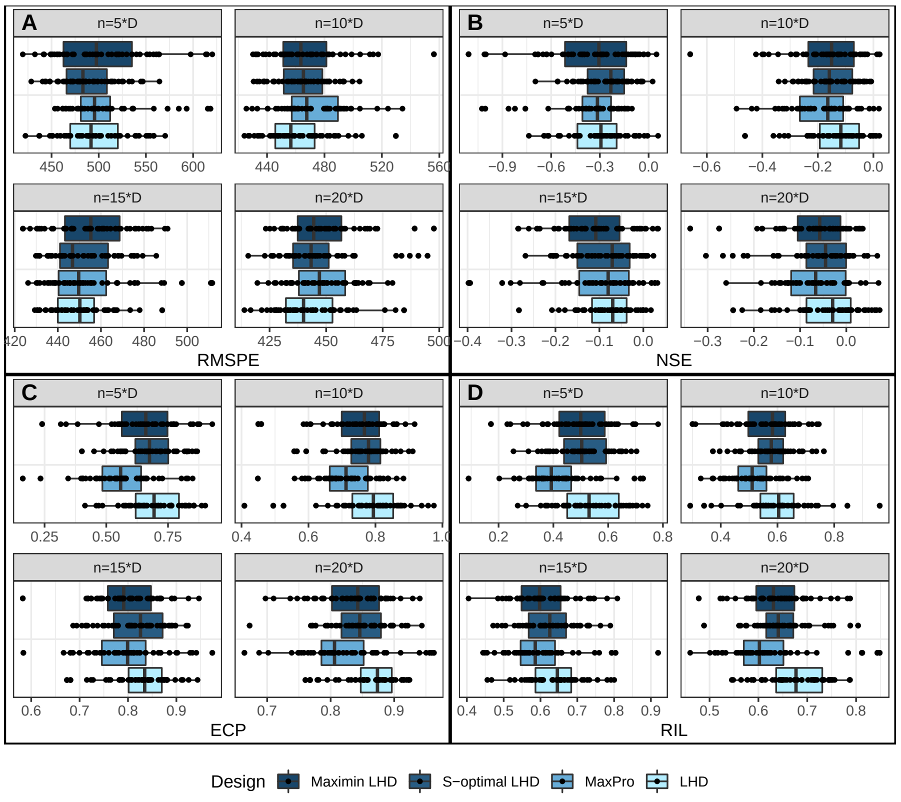

Figure 6.7: Schwefel function performance metric distributions for (A) RMSPEs, (B) Nash-Sutcliff efficiencies, (C) empirical coverage probabilities, and (D) relative interval lengths. For all design sizes and methods the GP emulator does not capture the true nature of the response surface accurately.

The results for these two test functions brings question to the $n = 10d$ rule as a general piece of advice. Whilst a reasonable choice for many contexts; guidance should be available for practitioners concerning appropriate design sizes for simulators whose response surfaces are complex. One work that aims to answer this question is Harari et al. [31] in which they explore the relationship between sample size, model complexity, and prediction accuracy.

If the reader is interested in comparisons for other specific test functions we direct them to tables B.1 through B.4 which provide the medians and interquartile ranges of performance metrics for each test function - design size combination. A couple highlights: LHDs led to much higher variance in the metrics for a few test functions, namely the Bohachevsky 1, Branin, Bratley, Cheng & Sandu, Currin Exponential, Franke, Lim, and Moon test functions. Maximin LHDs perform best, with respect to metric medians, for the Trid function and G-function.

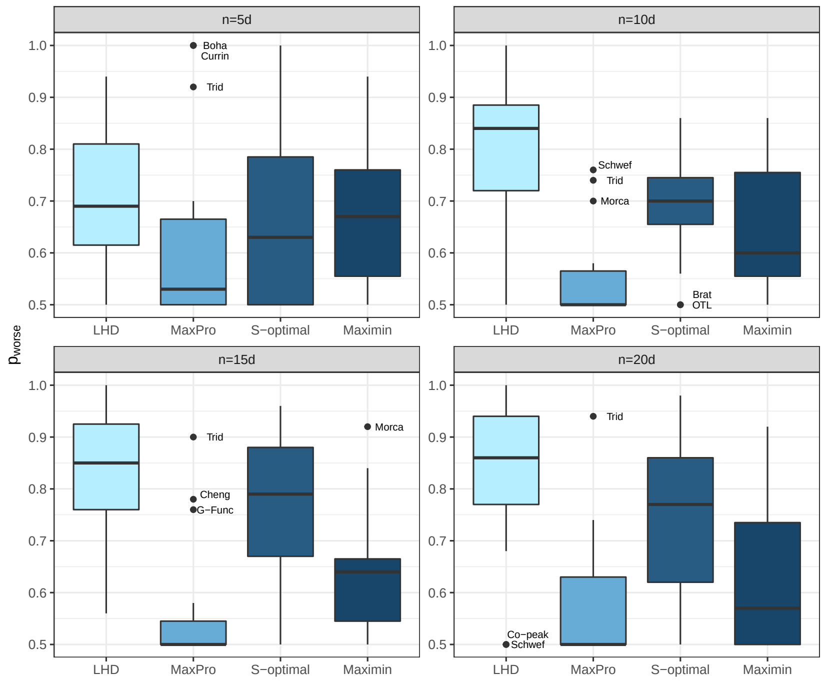

Figure 6.8: Boxplot of proportion of times a method does worse than the best median for each test function stratified by initial design size. Here RMSPE is used to compare predictive accuracy.

6.3.2 Evaluation Over the Suite of Test Functions

We now turn our attention to comparing the design methods across test functions. The focus here is on the effect of design on predictive performance through RMSPE with the two measures outlined in section 6.1. A boxplot of p_{worse} for RMSPE can be found in figure 6.8 below. It is observed that for design sizes greater than $5d$ the median of the p_{worse} distribution is 0.5 for the MaxPro designs; indicating that they frequently lead to the a GP emulator with the highest predictive accuracy. This confirms our findings from our independent analysis of the test functions above. Also note that the poor performance for the Schwefel and Product Peak test functions is not highlighted in this plot, thus emphasizing the need to pair this form of comparison with an understanding of the performance for each individual test function.

Next we consider the scaled average reverse ranking of the methods, a boxplot of which can be found in figure, 6.9. For this comparison $SRR_m \in [1, 8]$. Recalling that higher values of this measure indicate better relative performance; we come to the same conclusion as above - in that MaxPro designs generally lead to better GP predictive accuracy when there is sufficient data. For $n = 5d$, observe that the choice

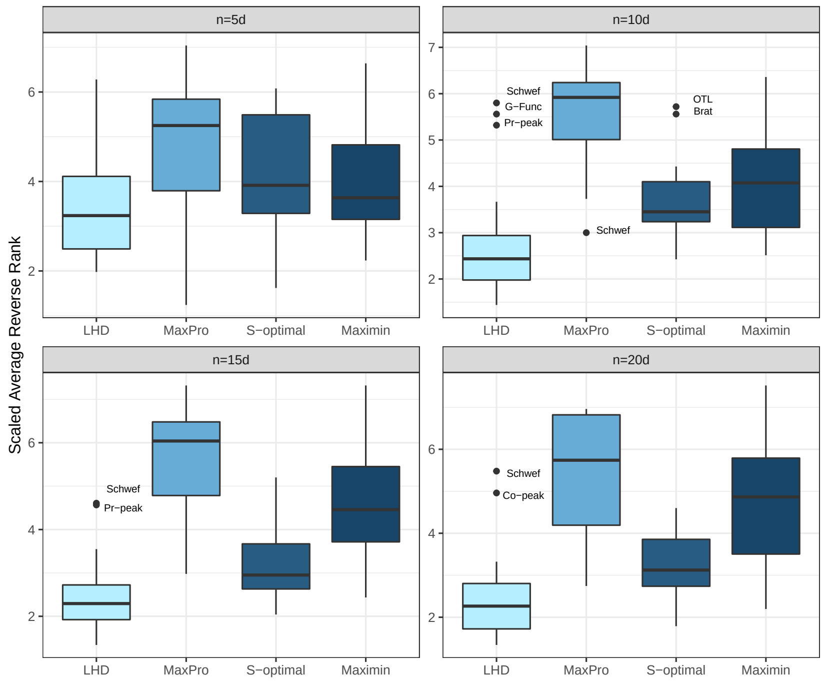

Figure 6.9: Comparison of designs boxplot of the scaled average reverse rank metric for each test function stratified by initial design size. The metric used for this comparison is RMSPE.

of design is less important as the lack of information about the response surface drives the poor emulator performance. We also point out that for $n = 20d$, the medians of the distributions have begun to converge again compared to $n = 10d$ and $n = 15d$. This trend would continue until large enough n, where choice of design becomes will again be less important.

As for general takeaways from our comparison; using a design based on some space-filling criteria can ensure improvement of GP emulator performance over a randomized LHD in many cases, but the choice of space-filling criteria (from our simulations) may not be as clear in a practical sense. By that we mean that a practitioner using these methods must also consider the time it takes to implement a new method, if one is already available, and the cost that accrues; it may be more cost effective to use a method with marginally worse performance that has existing code, than to spend the time coding up the optimal design method.

Setting practical difference aside, from the above analysis we would recommend that a practitioner use MaxPro designs if they want to ensure best performance of a GP emulator in terms of predictive accuracy. MaxPro designs also regularly lead to the highest empirical coverage probabilities paired with the shortest median relative interval lengths in our simulations.

We reiterate that the set of design methods chosen are not exhaustive. There may currently or one day be a design method which out performs those included here over the set of test functions. This is where the novelty of the VLSE comes in; with the ease of access to the included test functions and the step by step process outlined, a researcher or practitioner could generate comparison data of their own and make direct comparison to what has been done here.

Chapter 7

Conclusion

In this thesis we outlined methods for evaluation and comparison of design and analysis methods for computer experiments. Examples for both contexts were presented utilizing test functions found in the VLSE. The focus here is not necessarily on the conclusions from our simulations, but rather the approach used. To make recommendations or come to conclusions as to which method should be used in practice, one should always draw from evidence based on many examples. The procedure outlined for creating data for comparison can be applied to any collection of test functions that best suits the problem at hand. By creating many sets of test and training data for each test function, we account for variability in the chosen performance metrics that is not due to the methods itself. And, with the resulting data one can make numerical and visual comparisons between their chosen emulators or designs and determine how to proceed.

This work introduced two new measures of comparative performance; the proportion of times a method does worse than the best median and the scaled average reverse rank. These were utilized in both our comparison of emulators and our comparison of design methods. For emulators, we found that GPs and TGPs performed best in terms of predictive accuracy, but for some test functions the BASS model was competitive. For designs we found that MaxPro designs most frequently lead to the best GP performance in terms of both predictive accuracy and prediction interval coverage and length. But, these comparative results must always be paired with an understanding of the methods on each individual test function. In many cases, for both design and analysis, the differences observed between methods can either be classified as choosing a best poor performer or more commonly, choosing from various methods which do a satisfactory job (i.e. no practical difference between methods was observed). This is particularly relevant to the design literature as many proposed methods are justified theoretically [36, 7]. But, for any of these methods to be adopted with any sort of regularity they should be shown make practical differences in emulator performance.

The novelty in this work is in both the existence of a library of test functions and the onset of establishing an agreed upon method in which to compare methods for the design and analysis of computer experiments. The use of these in tandem will allow for researchers to make quicker comparisons of new methods to well-established methods, whilst allowing practitioners to adapt the comparison to their situation by imparting knowledge on the choice of test functions from the library, or perhaps adding new ones. A future endeavor useful for practitioners would be research into a measure of response surface complexity that could be assigned to the test functions found in the VLSE. This way the complexity measure could be suggested for novel simulators and used to choose the suite of test functions appropriate for their comparison.

The prospect for this work is to serve as a guide for users of computer experiments to choose the right design and emulation methods for their problems. A fundamental component of this is to grow the VLSE so that it contains test functions which cover the whole of conceivable real-world problems. A full survey of design and emulation methods used in the computer experiments literature should be done; making the process of choosing which methods to use in a certain contexts quick and easy. Ultimately the greatest limitation to performing such a comparison is time and resources, thus having a standardized method of evaluation would allow researchers and practitioners to create and add to a body of comparisons. Hopefully this work is the beginning of that process.

Bibliography

[1] P Audze. New approach to planning out of experiments. *Problems of dynamics and strengths*, 35:104–107, 1977.

[2] Shan Ba and V Roshan Joseph. Maxpro: Maximum projection designs. r package version 3.1-2, 2015.

[3] Leonardo S Bastos and Anthony O'Hagan. Diagnostics for gaussian process emulators. *Technometrics*, 51(4):425–438, 2009.

[4] Stuart J Bates, Jonathan Sienz, and Dean S Langley. Formulation of the audze–eglais uniform latin hypercube design of experiments. *Advances in Engineering Software*, 34(8):493–506, 2003.

[5] Maria J Bayarri, James O Berger, Rui Paulo, Jerry Sacks, John A Cafeo, James Cavendish, Chin-Hsu Lin, and Jian Tu. A framework for validation of computer models. *Technometrics*, 49(2):138–154, 2007.

[6] Einat Neumann Ben-Ari and David M Steinberg. Modeling data from computer experiments: an empirical comparison of kriging with mars and projection pursuit regression. *Quality Engineering*, 19(4):327–338, 2007.

[7] Derek Bingham, Randy R Sitter, and Boxin Tang. Orthogonal and nearly orthogonal designs for computer experiments. *Biometrika*, 96(1):51–65, 2009.

[8] Paul Bratley, Bennett L Fox, and Harald Niederreiter. Implementation and tests of low-discrepancy sequences. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 2(3):195–213, 1992.

[9] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees.* CRC press, 1984.

[10] Rob Carnell and Maintainer Rob Carnell. Package 'lhs'. *URL http://cran. stat. auckland. ac. nz/web/packages/lhs/lhs. pdf*, 780, 2020.

[11] William L Chapman, William J Welch, Kenneth P Bowman, Jerome Sacks, and John E Walsh. Arctic sea ice variability: Model sensitivities and a multidecadal simulation. *Journal of Geophysical Research: Oceans*, 99(C1):919–935, 1994.

[12] Hao Chen, Jason L Loeppky, Jerome Sacks, and William J Welch. Analysis methods for computer experiments: How to assess and what counts? *Statistical science*, pages 40–60, 2016.

[13] Ray-Bing Chen, Dai-Ni Hsieh, Ying Hung, and Weichung Wang. Optimizing latin hypercube designs by particle swarm. *Statistics and computing*, 23(5):663–676, 2013.

[14] Haiyan Cheng and Adrian Sandu. Collocation least-squares polynomial chaos method. In *Proceedings of the 2010 Spring Simulation Multiconference*, pages 1–6, 2010.

[15] Hugh A Chipman, Edward I George, and Robert E McCulloch. Bayesian cart model search. *Journal of the American Statistical Association*, 93(443):935–948, 1998.

[16] Hugh A Chipman, Edward I George, and Robert E McCulloch. Bayesian treed models. *Machine Learning*, 48(1-3):299–320, 2002.

[17] Carla Currin, Toby Mitchell, Max Morris, and Don Ylvisaker. A bayesian approach to the design and analysis of computer experiments. Technical report, Oak Ridge National Lab., TN (USA), 1988.

[18] Carla Currin, Toby Mitchell, Max Morris, and Don Ylvisaker. Bayesian prediction of deterministic functions, with applications to the design and analysis of computer experiments. *Journal of the American Statistical Association*, 86(416): 953–963, 1991.

[19] David GT Denison, Bani K Mallick, and Adrian FM Smith. Bayesian mars. *Statistics and Computing*, 8(4):337–346, 1998.

[20] Laurence Charles Ward Dixon. The global optimization problem. an introduction. *Toward global optimization*, 2:1–15, 1978.

[21] Delphine Dupuy, Céline Helbert, Jessica Franco, et al. Dicedesign and diceeval: Two r packages for design and analysis of computer experiments. *Journal of Statistical Software*, 65(11):1–38, 2015.

[22] Alexander Forrester, Andras Sobester, and Andy Keane. *Engineering design via surrogate modelling: a practical guide*. John Wiley & Sons, 2008.

[23] Devin Francom and Bruno Sansó. Bass: An r package for fitting and performing sensitivity analysis of bayesian adaptive spline surfaces. *Journal of Statistical Software*, 2, 2019.

[24] Devin Francom, Bruno Sansó, Ana Kupresanin, and Gardar Johannesson. Sensitivity analysis and emulation for functional data using bayesian adaptive splines. *Statistica Sinica*, pages 791–816, 2018.

[25] Richard Franke. A critical comparison of some methods for interpolation of scattered data. Technical report, NAVAL POSTGRADUATE SCHOOL MONTEREY CA, 1979.

[26] Jerome H Friedman. Multivariate adaptive regression splines. *The annals of statistics*, pages 1–67, 1991.

[27] Sushant S Garud, Iftekhar A Karimi, and Markus Kraft. Design of computer experiments: A review. *Computers & Chemical Engineering*, 106:71–95, 2017.

[28] Alan Genz. Testing multidimensional integration routines. In *Proc. of international conference on Tools, methods and languages for scientific and engineering computation*, pages 81–94, 1984.

[29] Robert B Gramacy and Herbert K H Lee. Bayesian treed gaussian process models with an application to computer modeling. *Journal of the American Statistical Association*, 103(483):1119–1130, 2008.

[30] Robert B Gramacy et al. tgp: an r package for bayesian nonstationary, semiparametric nonlinear regression and design by treed gaussian process models. *Journal of Statistical Software*, 19(9):6, 2007.

[31] Ofir Harari, Derek Bingham, Angela Dean, and Dave Higdon. Computer experiments: Prediction accuracy, sample size and model complexity revisited. *Statistica Sinica*, pages 899–919, 2018.

[32] William V Harper and Sumant K Gupta. *Sensitivity/uncertainty analysis of a borehole scenario comparing Latin hypercube sampling and deterministic sensitivity approaches.* Office of Nuclear Waste Isolation, Battelle Memorial Institute, 1983.

[33] Dave Higdon. Space and space-time modeling using process convolutions. In *Quantitative methods for current environmental issues*, pages 37–56. Springer, 2002.

[34] Dave Higdon, Marc Kennedy, James C Cavendish, John A Cafeo, and Robert D Ryne. Combining field data and computer simulations for calibration and prediction. *SIAM Journal on Scientific Computing*, 26(2):448–466, 2004.

[35] James Paul Holloway, Derek Bingham, Chuan-Chih Chou, Forrest Doss, R Paul Drake, Bruce Fryxell, Michael Grosskopf, Bart Van der Holst, Bani K Mallick, Ryan McClarren, et al. Predictive modeling of a radiative shock system. *Reliability Engineering & System Safety*, 96(9):1184–1193, 2011.

[36] Mark E Johnson, Leslie M Moore, and Donald Ylvisaker. Minimax and maximin distance designs. *Journal of statistical planning and inference*, 26(2):131–148, 1990.

[37] Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

[38] V Roshan Joseph, Evren Gul, and Shan Ba. Maximum projection designs for computer experiments. *Biometrika*, 102(2):371–380, 2015.

[39] Cari G Kaufman, Derek Bingham, Salman Habib, Katrin Heitmann, Joshua A Frieman, et al. Efficient emulators of computer experiments using compactly supported correlation functions, with an application to cosmology. *The Annals of Applied Statistics*, 5(4):2470–2492, 2011.

[40] Marc C Kennedy and Anthony O'Hagan. Bayesian calibration of computer models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(3):425–464, 2001.

[41] Q Ye Kenny, William Li, and Agus Sudjianto. Algorithmic construction of optimal symmetric latin hypercube designs. *Journal of statistical planning and inference*, 90(1):145–159, 2000.

[42] Daniel G Krige. A statistical approach to some basic mine valuation problems on the witwatersrand. *Journal of the Southern African Institute of Mining and Metallurgy*, 52(6):119–139, 1951.

[43] Manuel Laguna and Rafael Martí. Experimental testing of advanced scatter search designs for global optimization of multimodal functions. *Journal of Global Optimization*, 33(2):235–255, 2005.

[44] Yong B Lim, Jerome Sacks, WJ Studden, and William J Welch. Design and analysis of computer experiments when the output is highly correlated over the input space. *Canadian Journal of Statistics*, 30(1):109–126, 2002.

[45] C Devon Lin and Boxin Tang. Latin hypercubes and space-filling designs. *Handbook of design and analysis of experiments*, pages 593–625, 2015.

[46] Daniel J Lizotte, Russell Greiner, and Dale Schuurmans. An experimental methodology for response surface optimization methods. *Journal of Global Optimization*, 53(4):699–736, 2012.

[47] Jason L Loeppky, Jerome Sacks, and William J Welch. Choosing the sample size of a computer experiment: A practical guide. *Technometrics*, 51(4):366–376, 2009.

[48] Blake MacDonald, Pritam Ranjan, Hugh Chipman, et al. Gpfit: An r package for fitting a gaussian process model to deterministic simulator outputs. *Journal of Statistical Software*, 64(i12), 2015.

[49] M. D. McKay, R. J. Beckman, and W. J. Conover. A comparison of three methods for selecting values of input variables in the analysis of output from

a computer code. *Technometrics*, 21(2):239–245, 1979. ISSN 00401706. URL `http://www.jstor.org/stable/1268522`.

[50] Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

[51] Jonas Mockus, Vytautas Tiesis, and Antanas Zilinskas. The application of bayesian methods for seeking the extremum. *Towards global optimization*, 2 (117-129):2, 1978.

[52] Hyejung Moon. *Design and analysis of computer experiments for screening input variables*. PhD thesis, The Ohio State University, 2010.

[53] William J Morokoff and Russel E Caflisch. Quasi-monte carlo integration. *Journal of computational physics*, 122(2):218–230, 1995.

[54] Max D Morris and Toby J Mitchell. Exploratory designs for computational experiments. *Journal of statistical planning and inference*, 43(3):381–402, 1995.

[55] J Eamonn Nash and Jonh V Sutcliffe. River flow forecasting through conceptual models part i—a discussion of principles. *Journal of hydrology*, 10(3):282–290, 1970.

[56] Jeremy E Oakley and Anthony O'Hagan. Probabilistic sensitivity analysis of complex models: a bayesian approach. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 66(3):751–769, 2004.

[57] Luc Pronzato and Werner G Müller. Design of computer experiments: space filling and beyond. *Statistics and Computing*, 22(3):681–701, 2012.

[58] Jerome Sacks, William J Welch, Toby J Mitchell, and Henry P Wynn. Design and analysis of computer experiments. *Statistical science*, pages 409–423, 1989.

[59] Andrea Saltelli, Karen Chan, M Scott, et al. Sensitivity analysis. probability and statistics series. *John and Wiley & Sons, New York*, 2000.

[60] Thomas J Santner, Brian J Williams, William I Notz, and Brain J Williams. *The design and analysis of computer experiments*, volume 1. Springer, 2003.

[61] Rafal Stocki. A method to improve design reliability using optimal latin hypercube sampling. *Computer Assisted Mechanics and Engineering Sciences*, 12(4): 393, 2005.

[62] HF Stripling, RG McClarren, CC Kuranz, MJ Grosskopf, E Rutter, and BR Torralva. A calibration and data assimilation method using the bayesian mars emulator. *Annals of Nuclear Energy*, 52:103–112, 2013.

[63] Sonja Surjanovic and Derek Bingham. Virtual library of simulation experiments: test functions and datasets. *Simon Fraser University, Burnaby, BC, Canada, accessed May*, 13:2015, 2013.

[64] Boxin Tang. Orthogonal array-based latin hypercubes. *Journal of the American statistical association*, 88(424):1392–1397, 1993.

[65] Edwin R Van Dam, Bart Husslage, Dick Den Hertog, and Hans Melissen. Maximin latin hypercube designs in two dimensions. *Operations Research*, 55(1): 158–169, 2007.

[66] William J Welch, Robert J Buck, Jerome Sacks, Henry P Wynn, Toby J Mitchell, and Max D Morris. Screening, predicting, and computer experiments. *Technometrics*, 34(1):15–25, 1992.

[67] Brian Williams, Dave Higdon, Jim Gattiker, Leslie Moore, Michael McKay, Sallie Keller-McNulty, et al. Combining experimental data and computer simulations, with an application to flyer plate experiments. *Bayesian Analysis*, 1(4):765–792, 2006.

[68] Christopher K. I Williams and Carl Edward Rasmussen. *Gaussian Processes for Machine Learning.* Adaptive Computation and Machine Learning series. The MIT Press, 2019. ISBN 9780262256834.

[69] CF Jeff Wu and Michael S Hamada. *Experiments: planning, analysis, and optimization*, volume 552. John Wiley & Sons, 2011.

[70] Shelemyahu Zacks. *Modern industrial statistics: design and control of quality and reliability.* Cengage Learning, 1998.

Appendix A

Performance Metric Tables For Emulator Comparison

Table A.1: Median (IQR) of performance metrics computed for emulator comparison at $n = 5d$.

Test Function	Measures of Predictive Performance					
	RMSPE			NSE		
	GP	TGP	BASS	GP	TGP	BASS
Bohachevsky 1	**0.134 (0.0624)**	0.146 (0.0608)	0.741 (0.153)	**0.971 (0.0319)**	0.967 (0.0335)	0.0707 (0.106)
Borehole	**0.0474 (0.00912)**	0.128 (0.0258)	0.0783 (0.0283)	**0.998 (0.000942)**	0.983 (0.00506)	0.994 (0.00478)
Branin	**0.452 (0.221)**	0.598 (0.351)	0.794 (0.275)	**0.693 (0.224)**	0.543 (0.316)	0.0714 (0.122)
Bratley	0.132 (0.0296)	0.133 (0.0638)	**0.0129 (0.0144)**	0.98 (0.0103)	0.98 (0.0181)	**1 (0.000438)**
Cheng & Sandu	**0.135 (0.0947)**	0.258 (0.164)	0.474 (0.606)	**0.979 (0.0209)**	0.913 (0.0378)	0.723 (0.373)
Corner-Peak	2.08 (3.75)	1.9 (3.81)	**1.55 (3.46)**	0.0586 (0.203)	0.136 (0.204)	**0.386 (0.446)**
Currin Exponetial	0.306 (0.147)	**0.258 (0.076)**	0.342 (0.0932)	0.895 (0.0982)	**0.928 (0.0766)**	0.856 (0.0698)
Currin Sinusoidal	**0.0514 (0.0407)**	0.469 (0.179)	0.812 (0.144)	**0.997 (0.00544)**	0.76 (0.108)	0.229 (0.0697)
Franke	**0.391 (0.171)**	0.405 (0.285)	0.758 (0.416)	**0.815 (0.104)**	0.808 (0.112)	0.343 (0.141)
Forrester	0.586 (0.465)	**0.54 (0.417)**	0.603 (0.317)	0.307 (0.78)	**0.351 (0.783)**	0.158 (0.475)
G-Function	**0.689 (0.274)**	0.744 (0.225)	0.831 (0.305)	0.283 (0.25)	**0.32 (0.231)**	0.0632 (0.0756)
GP	**0.297 (0.127)**	0.437 (0.168)	0.735 (0.299)	**0.91 (0.0578)**	0.8 (0.111)	0.38 (0.369)
Gramacy	0.599 (1.41)	**0.434 (1.2)**	0.779 (1.03)	0.506 (0.691)	**0.735 (0.639)**	0.105 (0.203)
Morokoff & Calflisch	**0.281 (0.105)**	0.3 (0.0908)	0.484 (0.484)	**0.91 (0.0607)**	0.884 (0.0588)	0.702 (0.508)
Oakley & O'Hagan	**0.00259 (0.00215)**	0.0129 (0.00548)	0.00564 (0.00589)	**1 (1.66e-05)**	1 (0.000206)	1 (9.36e-05)
OTL	**0.0339 (0.0148)**	0.0962 (0.032)	0.0562 (0.0262)	**0.999 (0.00109)**	0.99 (0.005)	0.997 (0.0027)
Piston	**0.124 (0.0298)**	0.281 (0.0612)	0.211 (0.0732)	**0.983 (0.00889)**	0.927 (0.0184)	0.952 (0.0272)
Product-Peak	1.17 (0.687)	**1.13 (0.603)**	1.35 (0.651)	0.199 (0.396)	**0.303 (0.191)**	-0.0147 (0.0489)
Santner	0.658 (0.277)	**0.503 (0.267)**	0.879 (0.252)	**0.346 (0.656)**	0.19 (0.27)	0.0104 (0.125)
Schwefel	1.06 (0.333)	**1.01 (0.243)**	1.01 (0.335)	-0.0943 (0.235)	**0.0143 (0.0759)**	0.0032 (0.0168)

Test Function	Measures of Prediction Variability					
	ECP			RIL		
	GP	TGP	BASS	GP	TGP	BASS
Bohachevsky 1	**0.998 (0.0222)**	0.899 (0.178)	0.436 (0.198)	**3.05 (1.64)**	3.28 (3.21)	58.2 (54.3)
Borehole	**0.949 (0.0294)**	0.899 (0.0742)	0.922 (0.148)	**0.853 (0.252)**	2.37 (1.05)	1.59 (0.636)
Branin	0.7 (0.247)	**0.892 (0.0865)**	0.476 (0.147)	0.836 (0.514)	11.7 (14.4)	**1.07 (0.184)**
Bratley	0.909 (0.082)	0.925 (0.074)	**1 (0)**	1.8 (0.281)	1.8 (1.4)	**0.287 (0.412)**
Cheng & Sandu	0.911 (0.0772)	0.953 (0.089)	**0.995 (0.106)**	**0.29 (0.0689)**	0.982 (0.194)	1.22 (0.275)
Corner-Peak	0.756 (0.137)	0.937 (0.127)	**0.964 (0.0868)**	6.86 (6.41)	13.6 (11.6)	**5.76 (5.62)**
Currin Exponential	0.798 (0.131)	**0.945 (0.0663)**	0.922 (0.106)	**3.21 (2.34)**	5.76 (4.06)	7.59 (7.68)
Currin Sinusoidal	0.998 (0.002)	**1 (0)**	0.323 (0.102)	**0.345 (0.12)**	2.31 (0.597)	1.1 (0.0413)
Franke	0.783 (0.193)	**0.888 (0.2)**	0.581 (0.185)	**5.32 (6.26)**	8.44 (5.18)	27.3 (15.1)
Forrester	0.767 (0.437)	**0.793 (0.258)**	0.774 (0.163)	23.1 (75.6)	**8.48 (23.7)**	10.9 (16.3)
G-Function	0.613 (0.15)	**0.9 (0.0642)**	0.376 (0.274)	**9.06 (5.83)**	16.1 (11.5)	47.9 (52.4)
GP	0.846 (0.313)	**0.921 (0.115)**	0.63 (0.214)	**6.73 (7.71)**	11.3 (9.24)	27.5 (31.4)
Gramacy	0.726 (0.422)	**0.837 (0.268)**	0.633 (0.262)	1.14 (1.67)	9.99 (8.62)	**1.09 (0.0452)**
Morokoff & Calflisch	0.822 (0.11)	**0.965 (0.0749)**	0.904 (0.243)	**4.51 (3.48)**	9.34 (7.22)	20.6 (19.2)
Oakley & O'Hagan	**1 (0)**	0.868 (0.26)	**1 (0)**	0.505 (0.405)	0.19 (0.112)	**0.0208 (0.053)**
OTL	0.976 (0.0505)	0.968 (0.0572)	**0.996 (0.0237)**	**0.938 (0.309)**	2.28 (0.208)	1.94 (1.06)
Piston	0.879 (0.0519)	**0.913 (0.0859)**	0.824 (0.111)	**2.36 (0.926)**	6.25 (2.99)	4.19 (3.7)
Product-Peak	0.678 (0.361)	**0.908 (0.116)**	0.152 (0.0678)	**14.7 (13.3)**	29.1 (29.9)	195 (156)
Santner	0.996 (0.135)	**1 (0.1)**	0.356 (0.1)	2.63 (1.43)	3.28 (0.365)	**1.01 (0.0376)**
Schwefel	0.658 (0.124)	**0.866 (0.097)**	0.177 (0.0563)	48.2 (69.3)	**40.2 (33.5)**	144 (189)

Table A.2: Median (IQR) of performance metrics computed for emulator comparison at $n = 10d$.

Test Function	Measures of Predictive Performance					
	RMSPE			NSE		
	GP	TGP	BASS	GP	TGP	BASS
Bohachevsky 1	0.0496 (0.0177)	**0.00618 (0.00247)**	0.0641 (0.0155)	0.997 (0.00159)	**1 (3.2e-05)**	0.995 (0.00152)
Borehole	**0.0203 (0.00514)**	0.0677 (0.0119)	0.0389 (0.00492)	**1 (0.000241)**	0.995 (0.00109)	0.998 (0.000526)
Branin	**0.118 (0.0435)**	0.131 (0.0659)	0.268 (0.183)	**0.984 (0.0117)**	0.982 (0.0181)	0.923 (0.132)
Bratley	0.0521 (0.0252)	0.0284 (0.013)	**0.00193 (0.0014)**	0.997 (0.00348)	0.999 (0.00113)	**1 (5.68e-06)**
Cheng & Sandu	0.0463 (0.0198)	**0.0173 (0.0159)**	0.0374 (0.0217)	0.997 (0.000988)	**1 (0.000602)**	0.998 (0.00149)
Corner-Peak	1.71 (2.77)	1.78 (2.72)	**1.45 (2.57)**	0.143 (0.223)	0.161 (0.167)	**0.414 (0.353)**
Currin Exponential	**0.138 (0.0456)**	0.174 (0.0657)	0.149 (0.0631)	**0.978 (0.0183)**	0.968 (0.0284)	0.97 (0.0236)
Currin Sinusoidal	0.00543 (0.00471)	**0.00442 (0.0053)**	0.504 (0.265)	**1 (5.02e-05)**	**1 (5.85e-05)**	0.711 (0.256)
Franke	**0.163 (0.113)**	0.23 (0.145)	0.374 (0.103)	**0.969 (0.0286)**	0.949 (0.0473)	0.869 (0.0487)
Forrester	0.0423 (0.0325)	**0.0243 (0.0136)**	0.268 (0.161)	0.997 (0.00444)	**0.999 (0.000991)**	0.862 (0.236)
G-Function	0.558 (0.211)	0.574 (0.196)	**0.535 (0.198)**	0.634 (0.127)	0.615 (0.121)	**0.668 (0.0806)**
GP	**0.151 (0.0597)**	0.188 (0.121)	0.382 (0.13)	**0.981 (0.0166)**	0.968 (0.0475)	0.876 (0.0883)
Gramacy	**0.293 (0.0415)**	0.296 (0.0718)	0.828 (0.173)	**0.906 (0.029)**	0.898 (0.048)	0.338 (0.266)
Morokoff & Calflisch	**0.182 (0.0292)**	0.22 (0.0381)	0.223 (0.072)	**0.965 (0.0141)**	0.951 (0.0129)	0.944 (0.0262)
Oakley & O'Hagan	0.000586 (0.000235)	0.000593 (0.000292)	**2e-05 (0.000159)**	**1 (3.69e-07)**	**1 (4.13e-07)**	**1 (3.61e-08)**
OTL	0.0177 (0.00365)	0.0504 (0.00961)	**0.0143 (0.00536)**	**1 (9.35e-05)**	0.997 (0.000879)	**1 (0.00017)**
Piston	**0.0773 (0.0123)**	0.196 (0.0465)	0.126 (0.0246)	**0.994 (0.00143)**	0.955 (0.0198)	0.983 (0.00557)
Product-Peak	**0.65 (0.537)**	0.845 (0.585)	1.23 (0.54)	**0.766 (0.175)**	0.432 (0.208)	0.0144 (0.0602)
Santner	0.00907 (0.0121)	**0.00514 (0.00485)**	0.801 (0.042)	1 (0.000349)	**0.957 (0.0672)**	0.151 (0.113)
Schwefel	0.998 (0.143)	0.993 (0.185)	**0.972 (0.104)**	-0.00804 (0.158)	**0.0603 (0.187)**	0.0137 (0.0231)

Test Function	Measures of Prediction Variability					
	ECP			RIL		
	GP	TGP	BASS	GP	TGP	BASS
Bohachevsky 1	**1 (0.011)**	0.97 (0.0582)	**1 (0)**	0.987 (0.625)	**0.0821 (0.157)**	2.79 (2.13)
Borehole	0.962 (0.0309)	**0.97 (0.0194)**	0.74 (0.197)	0.464 (0.17)	1.84 (0.666)	**0.377 (0.15)**
Branin	**0.97 (0.0353)**	0.947 (0.0803)	0.956 (0.088)	**0.363 (0.0617)**	1.14 (0.827)	0.992 (0.168)
Bratley	0.98 (0.0468)	0.919 (0.129)	**1 (0.002)**	0.819 (0.36)	0.282 (0.146)	**0.0216 (0.0401)**
Cheng & Sandu	0.962 (0.0275)	0.962 (0.0527)	**1 (0)**	0.145 (0.0193)	**0.0617 (0.0287)**	0.281 (0.0484)
Corner-Peak	0.828 (0.076)	**0.976 (0.0273)**	0.9 (0.119)	5.33 (8.34)	12.3 (8.43)	**1.48 (1.77)**
Currin Exponential	0.908 (0.0628)	**0.909 (0.11)**	0.884 (0.127)	**1.56 (1.52)**	2.25 (1.41)	1.94 (1.27)
Currin Sinusoidal	0.998 (5e-04)	**1 (0)**	0.845 (0.195)	0.0874 (0.0152)	**0.0633 (0.0321)**	1.16 (0.0881)
Franke	0.91 (0.133)	0.936 (0.108)	**0.86 (0.0853)**	**3.42 (1.84)**	6.5 (3.45)	6.93 (4.12)
Forrester	0.998 (0.001)	**1 (0)**	0.865 (0.14)	4.73 (3.07)	**1.61 (1.51)**	4.86 (11.6)
G-Function	**0.771 (0.123)**	0.888 (0.112)	0.817 (0.0867)	**7.67 (4.41)**	14 (7.74)	12.1 (6.17)
GP	**0.959 (0.0963)**	0.877 (0.112)	0.733 (0.137)	**2.87 (1.36)**	4.44 (5.77)	4.97 (2.34)
Gramacy	0.605 (0.068)	**0.705 (0.2)**	0.656 (0.077)	**0.337 (0.121)**	2.38 (5.4)	1.01 (0.086)
Morokoff & Calflisch	0.829 (0.0488)	0.896 (0.116)	**0.912 (0.0924)**	**3 (1.53)**	5 (3.07)	4.18 (2.35)
Oakley & O'Hagan	**1 (0)**	0.982 (0.0465)	**1 (0)**	0.191 (0.152)	0.00575 (0.00219)	**0.000299 (0.000138)**
OTL	**0.972 (0.0234)**	0.93 (0.0516)	0.857 (0.189)	0.52 (0.534)	1.3 (0.894)	**0.245 (0.244)**
Piston	0.909 (0.0276)	**0.96 (0.0344)**	0.744 (0.138)	**1.69 (0.607)**	4.16 (4.04)	1.87 (0.969)
Product-Peak	0.867 (0.114)	**0.891 (0.103)**	0.112 (0.0925)	**11.2 (10.3)**	19.6 (18)	90.5 (90.9)
Santner	0.998 (0.002)	**1 (0)**	0.259 (0.054)	**0.199 (0.0202)**	1.82 (0.72)	0.812 (0.102)
Schwefel	0.707 (0.152)	**0.876 (0.109)**	0.122 (0.0707)	**32.4 (40.7)**	35.1 (16.6)	98.6 (89.6)

Table A.3: Median (IQR) of performance metrics computed for emulator comparison at $n = 20d$.

Test Function	Measures of Predictive Performance					
	RMSPE			NSE		
	GP	TGP	BASS	GP	TGP	BASS
Bohachevsky 1	0.0284 (0.00601)	**0.000287 (0.000128)**	0.0188 (0.00507)	0.999 (0.000402)	**1 (8.43e-08)**	**1 (0.000268)**
Borehole	**0.0105 (0.00248)**	0.0384 (0.00583)	0.0129 (0.00495)	**1 (4.56e-05)**	0.999 (0.000374)	**1 (0.000124)**
Branin	0.044 (0.027)	**0.00592 (0.00464)**	0.0791 (0.0368)	0.998 (0.0018)	**1 (3.54e-05)**	0.994 (0.00425)
Bratley	0.0193 (0.00557)	0.00176 (0.000835)	**0.000784 (0.00113)**	**1 (0.000227)**	**1 (3.93e-06)**	**1 (1.92e-06)**
Cheng & Sandu	0.0198 (0.0125)	**0.000412 (0.000484)**	0.0205 (0.00913)	**1 (0.000496)**	**1 (5.24e-07)**	**1 (0.000364)**
Corner-Peak	2.42 (3)	2.51 (3.08)	**2.21 (2.76)**	0.234 (0.201)	0.208 (0.199)	**0.403 (0.308)**
Currin Exponential	**0.058 (0.031)**	0.0701 (0.0268)	0.0664 (0.0148)	**0.996 (0.00283)**	0.995 (0.00387)	0.995 (0.00279)
Currin Sinusoidal	0.000908 (0.000668)	**4.01e-06 (1.05e-06)**	0.0285 (0.00626)	**1 (1.37e-06)**	**1 (8.64e-12)**	0.999 (0.000413)
Franke	0.0803 (0.0252)	**0.0623 (0.0211)**	0.185 (0.0623)	0.993 (0.00255)	**0.996 (0.00197)**	0.962 (0.0262)
Forrester	0.00526 (0.00401)	**4.17e-05 (2.19e-05)**	0.0854 (0.022)	**1 (5.07e-05)**	**1 (2.85e-09)**	0.991 (0.00525)
G-Function	0.356 (0.109)	0.381 (0.104)	**0.34 (0.0818)**	0.869 (0.0474)	0.864 (0.0658)	**0.882 (0.0476)**
GP	0.0479 (0.0212)	**0.0295 (0.0103)**	0.219 (0.065)	0.998 (0.00168)	**0.999 (0.000563)**	0.951 (0.0339)
Gramacy	**0.0719 (0.024)**	0.196 (0.0207)	0.159 (0.0246)	**0.995 (0.0033)**	0.961 (0.0069)	0.974 (0.00717)
Morokoff & Calflisch	**0.00818 (0.00405)**	0.0178 (0.00501)	0.0122 (0.0066)	**1 (6.08e-05)**	**1 (0.000106)**	**1 (0.000137)**
Oakley & O'Hagan	0.000203 (7e-05)	1.95e-05 (1.72e-05)	**1.56e-06 (6.51e-07)**	**1 (3.63e-08)**	**1 (7.45e-10)**	**1 (2.32e-12)**
OTL	0.00857 (0.00161)	0.0225 (0.00458)	**0.00548 (0.00121)**	**1 (3e-05)**	0.999 (0.000164)	**1 (1.85e-05)**
Piston	**0.0509 (0.00878)**	0.111 (0.0235)	0.0829 (0.0126)	**0.997 (0.000536)**	0.987 (0.00648)	0.993 (0.00183)
Product-Peak	**0.256 (0.0776)**	0.398 (0.112)	0.642 (0.138)	**0.927 (0.0313)**	0.834 (0.066)	0.565 (0.19)
Santner	0.00206 (0.0024)	**1.46e-05 (6.65e-06)**	0.0827 (0.0267)	**1 (1.22e-05)**	**1 (2e-04)**	0.992 (0.0044)
Schwefel	0.99 (0.0615)	**0.892 (0.0799)**	1.01 (0.0648)	0.0299 (0.0868)	**0.262 (0.0856)**	0.014 (0.0125)

Test Function	Measures of Prediction Variability					
	ECP			RIL		
	GP	TGP	BASS	GP	TGP	BASS
Bohachevsky 1	0.986 (0.0162)	**1 (0)**	**1 (0.00125)**	0.547 (0.458)	**0.175 (0.0904)**	0.7 (0.175)
Borehole	**0.963 (0.0123)**	0.95 (0.0228)	0.718 (0.194)	0.199 (0.0396)	0.685 (0.313)	**0.119 (0.0859)**
Branin	**0.987 (0.0125)**	0.971 (0.0223)	0.972 (0.068)	0.189 (0.0169)	**0.0365 (0.0431)**	0.26 (0.0563)
Bratley	0.987 (0.0095)	0.959 (0.0585)	**0.992 (0.0233)**	0.514 (0.271)	0.0197 (0.0186)	**0.0017 (0.00276)**
Cheng & Sandu	0.974 (0.0355)	**1 (0.002)**	0.994 (0.0268)	0.0763 (0.00876)	**0.0148 (0.00244)**	0.074 (0.0128)
Corner-Peak	0.89 (0.0437)	**0.98 (0.0322)**	0.694 (0.162)	5.83 (3.45)	11.5 (13.1)	**0.632 (0.576)**
Currin Exponential	**0.946 (0.0415)**	0.94 (0.07)	0.863 (0.12)	0.858 (0.241)	0.723 (0.25)	**0.699 (0.394)**
Currin Sinusoidal	0.998 (5e-04)	**1 (0)**	**1 (0)**	0.0307 (0.00135)	**0.00351 (0.000264)**	0.18 (0.022)
Franke	**0.954 (0.0208)**	0.936 (0.07)	0.852 (0.0927)	1.47 (0.933)	**1.27 (1.35)**	2.46 (2.67)
Forrester	0.998 (0.002)	**1 (0)**	0.873 (0.125)	1.29 (0.459)	**0.202 (0.0658)**	3.44 (3.76)
G-Function	0.846 (0.0653)	**0.908 (0.0533)**	0.698 (0.0955)	5.02 (2.28)	7.23 (3.56)	**3.72 (3.96)**
GP	**0.995 (0.016)**	0.982 (0.0482)	0.745 (0.077)	1.36 (0.643)	**0.672 (0.265)**	2.93 (1.26)
Gramacy	**0.965 (0.0515)**	0.905 (0.092)	0.78 (0.071)	**0.276 (0.0377)**	5.71 (4.8)	0.323 (0.155)
Morokoff & Calflisch	**0.979 (0.0252)**	0.909 (0.0614)	0.752 (0.283)	0.302 (0.189)	0.309 (0.252)	**0.249 (0.23)**
Oakley & O'Hagan	**1 (0.001)**	**1 (0)**	**1 (0.001)**	0.0621 (0.0246)	0.0128 (0.00517)	**6.7e-05 (2.34e-05)**
OTL	**0.97 (0.0263)**	0.921 (0.0347)	0.774 (0.197)	0.246 (0.113)	0.57 (0.67)	**0.0761 (0.0241)**
Piston	0.906 (0.0274)	**0.977 (0.0199)**	0.712 (0.157)	**0.966 (0.422)**	2.2 (0.922)	1.03 (0.641)
Product-Peak	0.931 (0.042)	**0.979 (0.0242)**	0.77 (0.12)	8.15 (3.15)	17.2 (3.59)	**6.76 (5.44)**
Santner	0.998 (0.002)	**1 (0)**	0.98 (0.044)	0.067 (0.00691)	**0.00667 (0.00104)**	0.407 (0.0487)
Schwefel	0.757 (0.0927)	**0.891 (0.0392)**	0.085 (0.0342)	75.7 (170)	37.1 (10.6)	86 (95.2)

Table A.4: Median (IQR) of performance metrics computed for emulator comparison at $n = 50d$.

Test Function	Measures of Predictive Performance					
	RMSPE			NSE		
	GP	TGP	BASS	GP	TGP	BASS
Bohachevsky 1	0.00588 (0.00362)	**0.000107 (3.6e-05)**	0.00489 (0.000801)	**1 (3.98e-05)**	**1 (6.31e-09)**	**1 (7.8e-06)**
Borehole	**0.00347 (0.000689)**	0.00646 (0.00094)	0.00486 (0.000997)	**1 (4.38e-06)**	**1 (1.01e-05)**	**1 (9.79e-06)**
Branin	0.0116 (0.00735)	**0.000173 (0.000139)**	0.0194 (0.00693)	**1 (0.000166)**	**1 (4.89e-08)**	**1 (0.000201)**
Bratley	0.00573 (0.0022)	**0.000138 (4.43e-05)**	0.0015 (0.000838)	**1 (2.27e-05)**	**1 (1.18e-08)**	**1 (2.41e-06)**
Cheng & Sandu	0.00722 (0.00625)	**7.72e-05 (0.00015)**	0.00605 (0.00355)	**1 (8.66e-05)**	**1 (3.14e-08)**	**1 (3.63e-05)**
Corner-Peak	1.02 (1.54)	1.1 (1.75)	**0.965 (1.51)**	0.535 (0.344)	0.521 (0.361)	**0.633 (0.363)**
Currin Exponential	0.023 (0.00674)	**0.017 (0.0121)**	0.0315 (0.0134)	0.999 (0.000297)	**1 (0.000447)**	0.999 (0.000947)
Currin Sinusoidal	0.000102 (7.03e-05)	**1.86e-06 (1.05e-07)**	0.0045 (0.000652)	**1 (1.49e-08)**	**1 (4.21e-13)**	**1 (5.5e-06)**
Franke	0.0292 (0.0101)	**0.0148 (0.00523)**	0.14 (0.0309)	0.999 (0.00035)	**1 (0.000136)**	0.979 (0.00633)
Forrester	0.000441 (0.000558)	**7.56e-06 (7.75e-07)**	0.029 (0.0029)	**1 (7.54e-07)**	**1 (1.44e-11)**	0.999 (0.000233)
G-Function	0.356 (0.109)	0.381 (0.104)	**0.34 (0.0818)**	0.869 (0.0474)	0.864 (0.0658)	**0.882 (0.0476)**
GP	0.00952 (0.00288)	**0.00207 (0.000649)**	0.118 (0.0182)	**1 (6.38e-05)**	**1 (2.98e-06)**	0.987 (0.00435)
Gramacy	**0.0181 (0.00614)**	0.0325 (0.0159)	0.0343 (0.00992)	**1 (0.000242)**	0.999 (0.0011)	0.999 (0.000671)
Morokoff & Calflisch	**0.0752 (0.0187)**	0.0836 (0.0157)	0.0755 (0.0134)	**0.994 (0.00229)**	0.993 (0.00227)	**0.994 (0.00223)**
Oakley & O'Hagan	0.000208 (8.87e-05)	6.07e-06 (2.91e-06)	**3.67e-07 (1.02e-07)**	**1 (3.65e-08)**	**1 (4.01e-11)**	**1 (8.26e-14)**
OTL	0.00266 (0.000298)	0.00397 (0.000334)	**0.0016 (0.000352)**	**1 (1.63e-06)**	**1 (3.1e-06)**	**1 (1.1e-06)**
Piston	**0.0238 (0.00496)**	0.0473 (0.0205)	0.0498 (0.00528)	**0.999 (0.00015)**	0.998 (0.00196)	0.997 (0.000417)
Product-Peak	**0.106 (0.044)**	0.128 (0.0459)	0.346 (0.0665)	**0.99 (0.00501)**	0.985 (0.00689)	0.888 (0.0177)
Santner	0.000245 (0.000324)	**3.64e-06 (1.44e-07)**	0.012 (0.00174)	**1 (1.86e-07)**	**1 (1.86e-12)**	**1 (4.05e-05)**
Schwefel	**0.0621 (0.0115)**	0.723 (0.0804)	0.15 (0.0369)	**0.996 (0.00146)**	0.461 (0.0657)	0.975 (0.0116)

Test Function	Measures of Prediction Variability					
	ECP			RIL		
	GP	TGP	BASS	GP	TGP	BASS
Bohachevsky 1	0.995 (0.00675)	**1 (0)**	0.913 (0.172)	0.172 (0.0631)	0.253 (0.086)	**0.0907 (0.0433)**
Borehole	**0.976 (0.0127)**	0.95 (0.0171)	0.707 (0.124)	0.109 (0.168)	0.107 (0.03)	**0.0478 (0.0247)**
Branin	0.99 (0.009)	**1 (0)**	0.718 (0.194)	0.0604 (0.00459)	0.121 (0.0722)	**0.0336 (0.0131)**
Bratley	0.991 (0.0105)	**1 (0)**	0.939 (0.115)	0.125 (0.035)	0.129 (0.0434)	**0.00184 (0.00181)**
Cheng & Sandu	0.976 (0.0157)	**1 (0)**	0.702 (0.156)	0.0265 (0.00213)	0.0174 (0.00181)	**0.00856 (0.00293)**
Corner-Peak	0.937 (0.0245)	**0.986 (0.0227)**	0.788 (0.109)	6.45 (5.02)	6.5 (4.96)	**1.58 (1.76)**
Currin Exponential	0.964 (0.015)	**0.988 (0.014)**	0.7 (0.189)	0.352 (0.218)	**0.14 (0.0754)**	0.177 (0.0797)
Currin Sinusoidal	0.998 (0.002)	**1 (0)**	**1 (0)**	0.0081 (0.000235)	**0.00262 (0.000102)**	0.0299 (0.00301)
Franke	0.964 (0.0207)	**0.996 (0.017)**	0.694 (0.146)	0.458 (0.144)	**0.257 (0.152)**	1.11 (1.01)
Forrester	0.998 (0.0025)	**1 (0)**	0.602 (0.114)	0.386 (0.203)	**0.114 (0.0373)**	0.985 (0.628)
G-Function	0.846 (0.0653)	**0.908 (0.0533)**	0.698 (0.0955)	5.02 (2.28)	7.23 (3.56)	**3.72 (3.96)**
GP	**1 (0)**	0.999 (0.00817)	0.704 (0.142)	0.502 (0.227)	**0.0529 (0.0427)**	1.57 (1.31)
Gramacy	0.972 (0.011)	**0.999 (0.004)**	0.732 (0.104)	0.0703 (0.0128)	1.13 (0.859)	**0.035 (0.0243)**
Morokoff & Calflisch	0.894 (0.0203)	**0.983 (0.00875)**	0.634 (0.113)	1.2 (0.529)	1.81 (0.705)	0.402 (0.175)
Oakley & O'Hagan	**1 (0)**	**1 (0)**	0.982 (0.0762)	0.0163 (0.00976)	0.014 (0.0054)	**6.2e-06 (4.6e-06)**
OTL	0.985 (0.00733)	**0.9 (0.0157)**	0.71 (0.111)	0.099 (0.0433)	0.0703 (0.0394)	**0.0233 (0.0125)**
Piston	0.926 (0.0125)	**0.939 (0.0846)**	0.698 (0.145)	**0.438 (0.137)**	0.777 (0.906)	0.584 (0.396)
Product-Peak	**0.952 (0.0243)**	**0.952 (0.0595)**	0.729 (0.0752)	3.8 (1.39)	5.46 (4.4)	**3.56 (1.17)**
Santner	0.998 (0.002)	**1 (0)**	1 (0.0065)	0.0172 (0.000428)	**0.00523 (0.000175)**	0.0753 (0.00755)
Schwefel	**1 (0)**	0.94 (0.031)	0.771 (0.122)	3.91 (1.96)	44.5 (54.4)	**2.56 (1.37)**

Appendix B

Performance Metric Tables For Design Comparison

Table B.1: Median (IQR) of performance metrics computed for design comparison at $n = 5d$.

Test Function	Measures of Predictive Performance							
	RMSPE				NSE			
	LHD	MaxPro	S-optimal	Maximin	LHD	MaxPro	S-optimal	Maximin
Bohachevsky 1	**1210 (501)**	4310 (85.3)	1520 (804)	1250 (620)	**0.967 (0.0283)**	0.581 (0.0138)	0.947 (0.0621)	0.964 (0.0373)
Borehole	2.35 (0.436)	**2.09 (0.549)**	2.2 (0.477)	2.26 (0.561)	0.997 (0.00106)	**0.998 (0.00111)**	**0.998 (0.001)**	**0.998 (0.00119)**
Branin	31.5 (12.2)	**27.1 (3.96)**	28.7 (8)	30.4 (10.3)	**0.621 (0.307)**	0.718 (0.0662)	0.691 (0.175)	0.648 (0.232)
Bratley	0.0495 (0.011)	0.0429 (0.0124)	**0.0408 (0.0112)**	0.0444 (0.00747)	0.958 (0.0188)	0.967 (0.0189)	**0.971 (0.0164)**	0.966 (0.0115)
Cheng & Sandu	0.0605 (0.0263)	**0.04 (0.0384)**	0.0462 (0.0191)	0.0498 (0.0179)	0.974 (0.0222)	**0.988 (0.0283)**	0.985 (0.0127)	0.983 (0.0144)
Corner-Peak	0.00513 (0.00245)	**0.00469 (0.00227)**	0.00487 (0.00233)	0.00485 (0.00231)	0.0551 (0.0797)	**0.126 (0.114)**	0.0575 (0.0659)	0.0529 (0.0885)
Currin	1.43 (0.417)	1.48 (0.0535)	1.46 (0.375)	**1.32 (0.461)**	0.708 (0.169)	0.688 (0.0356)	0.697 (0.155)	**0.753 (0.168)**
Franke	0.0561 (0.03)	**0.041 (0.014)**	0.0511 (0.0171)	0.0477 (0.0154)	0.962 (0.042)	**0.98 (0.0137)**	0.968 (0.0213)	0.973 (0.0178)
Friedman	1.57 (0.624)	**1.5 (0.567)**	1.69 (0.992)	1.7 (0.513)	0.898 (0.0908)	0.905 (0.0782)	**0.879 (0.147)**	**0.879 (0.0806)**
G-function	0.732 (0.132)	0.746 (0.12)	**0.785 (0.103)**	0.734 (0.139)	0.282 (0.26)	0.253 (0.25)	**0.18 (0.175)**	0.288 (0.281)
GP	1 (0.618)	1.04 (0.955)	**0.865 (0.421)**	0.982 (0.535)	0.958 (0.0572)	0.954 (0.104)	**0.968 (0.0302)**	0.957 (0.0447)
Lim	0.567 (0.19)	**0.363 (0.225)**	0.441 (0.121)	0.556 (0.247)	0.917 (0.0557)	**0.965 (0.0558)**	0.948 (0.0317)	0.919 (0.0713)
Moon	0.0438 (0.0178)	0.0302 (0.00736)	**0.03 (0.0117)**	0.0324 (0.0155)	0.998 (0.00183)	**0.999 (0.000519)**	**0.999 (0.000832)**	**0.999 (0.00127)**
Morokoff & Caflisch	0.148 (0.0287)	0.142 (0.0185)	0.156 (0.0179)	**0.131 (0.0217)**	0.834 (0.0661)	0.847 (0.0403)	**0.818 (0.0409)**	0.868 (0.0429)
Oakley & O'Hagan	7.45e-05 (4.74e-05)	**4.62e-05 (3.98e-06)**	5.96e-05 (1.73e-05)	8.04e-05 (4.75e-05)	**1 (2.39e-05)**	**1 (1.14e-06)**	**1 (6.28e-06)**	**1 (2.18e-05)**
OTL	0.0453 (0.0148)	0.0391 (0.00863)	**0.0389 (0.00862)**	0.0421 (0.0136)	0.998 (0.000984)	**0.999 (0.000571)**	**0.999 (0.000509)**	**0.999 (0.000999)**
Piston	0.0196 (0.00435)	**0.0185 (0.00406)**	0.0192 (0.00428)	0.0203 (0.00547)	0.98 (0.00882)	**0.983 (0.00799)**	0.981 (0.0085)	0.979 (0.0116)
Product Peak	457000 (104000)	450000 (61400)	**447000 (26600)**	558000 (212000)	-0.0457 (0.442)	**0.0031 (0.203)**	-0.0161 (0.073)	-0.55 (1.23)
Schwefel	492 (50.5)	496 (31.1)	**483 (42.8)**	497 (72.5)	-0.293 (0.241)	-0.315 (0.176)	**-0.232 (0.225)**	-0.306 (0.377)
Trid	319 (262)	534 (103)	552 (90.7)	**298 (213)**	0.707 (0.562)	0.188 (0.297)	0.123 (0.293)	**0.751 (0.379)**

Test Function	Measures of Prediction Variability							
	ECP				RIL			
	LHD	MaxPro	S-optimal	Maximin	LHD	MaxPro	S-optimal	Maximin
Bohachevsky 1	**1 (0.0085)**	0.139 (0.0101)	0.947 (0.15)	0.995 (0.0229)	2.25 (1.94)	**0.212 (3.86e-05)**	**1.76 (1)**	2.02 (1.21)
Borehole	0.957 (0.038)	0.965 (0.0336)	**0.967 (0.032)**	0.96 (0.0328)	0.168 (0.0226)	**0.164 (0.0258)**	0.166 (0.0139)	0.168 (0.0237)
Branin	0.838 (0.303)	**0.878 (0.0591)**	0.859 (0.156)	0.865 (0.22)	3.23 (4.67)	**2.47 (0.303)**	3.64 (3.97)	4.75 (5.86)
Bratley	0.867 (0.0997)	0.83 (0.106)	**0.872 (0.109)**	**0.872 (0.091)**	2.14 (1.42)	1.91 (1.17)	**1.75 (1.35)**	2.09 (1.32)
Cheng & Sandu	0.916 (0.0914)	**0.965 (0.117)**	0.95 (0.0524)	0.935 (0.0705)	0.9 (0.387)	**0.571 (0.227)**	0.601 (0.231)	0.659 (0.579)
Corner Peak	0.764 (0.116)	**0.815 (0.0966)**	0.804 (0.112)	0.792 (0.104)	**15 (24.8)**	20.4 (18)	21 (23.6)	39.9 (65.6)
Currin	**0.746 (0.178)**	0.783 (0.0252)	0.748 (0.172)	0.787 (0.14)	0.26 (0.166)	0.346 (0.0314)	0.278 (0.217)	**0.257 (0.145)**
Franke	0.917 (0.118)	**0.954 (0.0924)**	0.922 (0.0954)	0.941 (0.0753)	0.874 (0.672)	**0.776 (0.228)**	0.806 (0.272)	0.846 (0.273)
Friedman	0.914 (0.125)	**0.923 (0.198)**	0.839 (0.264)	0.871 (0.167)	0.419 (0.0674)	**0.377 (0.0692)**	**0.377 (0.0679)**	0.407 (0.107)
G-function	0.667 (0.207)	0.59 (0.186)	0.572 (0.177)	**0.668 (0.139)**	4.62 (4.22)	3.18 (3.37)	**2.63 (2.73)**	3.54 (4.18)
GP	**1 (0.0723)**	**1 (0.00425)**	**1 (0)**	**1 (0)**	0.908 (0.555)	**0.665 (0.362)**	0.872 (0.653)	0.922 (1.01)
Lim	0.879 (0.115)	**0.943 (0.0469)**	0.925 (0.0899)	0.889 (0.127)	0.912 (1.14)	**0.673 (0.136)**	1 (1)	1.38 (1.41)
Moon	**1 (0.000667)**	**1 (0)**	**1 (0)**	**1 (0)**	0.231 (0.022)	**0.187 (0.0101)**	0.196 (0.0151)	0.217 (0.0171)
Morokoff & Caflisch	**0.842 (0.0823)**	0.777 (0.107)	0.745 (0.156)	0.836 (0.0787)	0.504 (0.205)	**0.379 (0.0991)**	0.443 (0.139)	0.397 (0.102)
Oakley & O'Hagan	**1 (0)**	**1 (0)**	**1 (0)**	**1 (0)**	0.000281 (3.39e-05)	**0.000232** (1.32e-06)	0.000239 (9.58e-06)	0.000268 (2.79e-05)
OTL	0.986 (0.0298)	0.988 (0.0153)	0.992 (0.021)	**0.993 (0.0298)**	0.0399 (0.00283)	**0.0367 (0.00223)**	0.0382 (0.00244)	0.0396 (0.0032)
Piston	0.914 (0.0608)	0.912 (0.0549)	0.904 (0.0654)	**0.903 (0.0662)**	0.132 (0.0325)	**0.122 (0.0221)**	0.124 (0.02)	0.125 (0.0192)
Product Peak	0.774 (0.0858)	0.729 (0.133)	**0.584 (0.132)**	0.827 (0.144)	20.7 (11.5)	11.4 (10)	**2.68 (2.61)**	18.8 (20.1)
Schwefel	**0.694 (0.175)**	0.558 (0.157)	0.675 (0.133)	0.66 (0.185)	0.53 (0.188)	**0.392 (0.128)**	0.504 (0.154)	0.5 (0.166)
Trid	0.912 (0.295)	0.668 (0.162)	0.636 (0.206)	**0.96 (0.282)**	1.27 (0.272)	1.22 (0.344)	**1.03 (0.281)**	1.05 (0.13)

Table B.2: Median (IQR) of performance metrics computed for design comparison at $n = 10d$.

Test Function	Measures of Predictive Performance							
	RMSPE				NSE			
	LHD	MaxPro	S-optimal	Maximin	LHD	MaxPro	S-optimal	Maximin
Bohachevsky 1	478 (155)	**324 (65.5)**	347 (106)	347 (87.4)	0.995 (0.00346)	**0.998 (0.000983)**	0.997 (0.00182)	0.997 (0.00135)
Borehole	1.1 (0.305)	**0.947 (0.143)**	1.02 (0.241)	0.961 (0.244)	0.999 (0.000309)	**1 (0.000131)**	**1 (0.00023)**	**1 (0.000211)**
Branin	8.32 (5.91)	**5 (1.21)**	6.18 (3.2)	6.33 (3.89)	0.973 (0.0394)	**0.991 (0.00488)**	0.985 (0.0152)	0.984 (0.0207)
Bratley	0.0256 (0.00517)	0.0235 (0.00415)	**0.0231 (0.00427)**	0.0241 (0.0052)	0.989 (0.00502)	0.99 (0.00339)	**0.991 (0.00351)**	0.99 (0.00428)
Cheng & Sandu	0.0237 (0.014)	**0.0164 (0.00633)**	0.0185 (0.00645)	0.0189 (0.0112)	0.996 (0.00465)	**0.998 (0.00156)**	**0.998 (0.00178)**	0.997 (0.00319)
Corner-Peak	0.00541 (0.0027)	**0.00516 (0.00232)**	0.00548 (0.00255)	0.00553 (0.00305)	0.0875 (0.122)	0.164 (0.139)	0.14 (0.139)	**0.165 (0.15)**
Currin	0.556 (0.364)	**0.37 (0.0759)**	0.455 (0.162)	0.442 (0.263)	0.956 (0.0614)	**0.98 (0.00765)**	0.97 (0.0211)	0.972 (0.038)
Franke	0.018 (0.00497)	**0.0148 (0.00432)**	0.0171 (0.00303)	0.0173 (0.00484)	0.996 (0.00219)	**0.997 (0.00156)**	0.996 (0.00125)	0.996 (0.00191)
Friedman	0.599 (0.156)	**0.482 (0.123)**	0.546 (0.118)	0.566 (0.145)	0.985 (0.00749)	**0.99 (0.00558)**	0.988 (0.0051)	0.986 (0.00752)
G-function	**0.609 (0.0806)**	0.613 (0.0761)	0.679 (0.0892)	**0.609 (0.0967)**	**0.503 (0.139)**	0.491 (0.118)	0.39 (0.157)	0.495 (0.179)
GP	0.269 (0.183)	**0.151 (0.0544)**	0.167 (0.0617)	0.183 (0.0773)	0.997 (0.0048)	**0.999 (0.000734)**	**0.999 (0.000931)**	**0.999 (0.00126)**
Lim	0.212 (0.104)	**0.121 (0.019)**	0.148 (0.0481)	0.154 (0.057)	0.988 (0.012)	**0.996 (0.00117)**	0.994 (0.00378)	0.994 (0.00472)
Moon	0.0149 (0.00553)	0.0116 (0.00292)	0.0126 (0.00262)	**0.0112 (0.00267)**	**1 (2e-04)**	**1 (7.44e-05)**	**1 (7.5e-05)**	**1 (8.13e-05)**
Morokoff & Caflisch	0.111 (0.0219)	0.104 (0.0199)	0.114 (0.0278)	**0.0986 (0.0124)**	0.905 (0.0378)	0.918 (0.0308)	0.902 (0.0469)	**0.927 (0.0183)**
Oakley & OH́agan	1.5e-05 (5.6e-06)	**1.24e-05 (2.2e-06)**	1.33e-05 (3.23e-06)	1.38e-05 (5.2e-06)	**1 (5.32e-07)**	**1 (1.67e-07)**	**1 (2.62e-07)**	**1 (4.49e-07)**
OTL	0.0222 (0.00344)	0.0207 (0.00231)	**0.0206 (0.00259)**	0.0212 (0.00391)	**1 (0.00012)**	**1 (6.98e-05)**	**1 (8.22e-05)**	**1 (0.000122)**
Piston	0.0116 (0.00203)	**0.0102 (0.00144)**	0.0107 (0.00162)	0.0116 (0.00191)	0.993 (0.00231)	**0.995 (0.00137)**	0.994 (0.00189)	0.993 (0.00213)
Product Peak	**421000 (132000)**	424000 (77500)	432000 (39400)	457000 (108000)	**0.145 (0.531)**	0.0734 (0.283)	0.0669 (0.142)	0.0233 (0.538)
Schwefel	**457 (27.4)**	468 (32.2)	465 (26.9)	464 (30.1)	**-0.117 (0.141)**	-0.164 (0.156)	-0.159 (0.14)	-0.151 (0.164)
Trid	62.3 (15.8)	51.4 (9.08)	55.8 (7.61)	**47.7 (9.69)**	0.989 (0.00526)	0.993 (0.0029)	0.991 (0.00212)	**0.994 (0.00278)**

Test Function	Measures of Prediction Variability							
	ECP				RIL			
	LHD	MaxPro	S-optimal	Maximin	LHD	MaxPro	S-optimal	Maximin
Bohachevsky 1	**1 (0.0025)**	0.996 (0.00525)	0.999 (0.00575)	**1 (0.001)**	1.06 (0.807)	**0.806 (0.501)**	1 (0.542)	1.02 (0.607)
Borehole	0.972 (0.0215)	**0.979 (0.0165)**	0.974 (0.0176)	0.978 (0.0189)	0.0868 (0.00969)	**0.0831 (0.00659)**	0.0841 (0.00566)	0.0878 (0.00814)
Branin	0.971 (0.0583)	**0.982 (0.0106)**	0.977 (0.0359)	0.972 (0.0399)	2.81 (3.34)	**2.29 (1.34)**	2.74 (1.7)	2.76 (1.94)
Bratley	0.941 (0.0408)	0.922 (0.0453)	0.943 (0.0457)	**0.944 (0.0478)**	1.66 (1.02)	**1.28 (0.962)**	1.29 (0.768)	1.4 (0.868)
Cheng & Sandu	0.963 (0.0525)	**0.974 (0.011)**	**0.974 (0.0176)**	**0.974 (0.0418)**	0.357 (0.261)	**0.28 (0.0953)**	0.338 (0.172)	0.364 (0.186)
Corner Peak	0.847 (0.07)	**0.875 (0.0453)**	0.862 (0.0422)	0.864 (0.0563)	**24.5 (32.8)**	37 (35.4)	28.4 (19.1)	39.5 (41.4)
Currin	0.902 (0.108)	**0.94 (0.0415)**	0.911 (0.063)	0.888 (0.0706)	0.194 (0.0643)	**0.154 (0.0266)**	0.161 (0.0498)	0.164 (0.0395)
Franke	**0.973 (0.0204)**	**0.973 (0.0209)**	0.971 (0.0181)	0.97 (0.0311)	0.394 (0.0915)	**0.344 (0.0727)**	0.363 (0.0648)	0.379 (0.0634)
Friedman	0.984 (0.0297)	**0.99 (0.0191)**	0.983 (0.0251)	0.973 (0.033)	0.199 (0.0138)	**0.18 (0.0104)**	0.183 (0.0101)	0.184 (0.0138)
G-function	**0.785 (0.0868)**	0.718 (0.0748)	0.719 (0.102)	0.742 (0.106)	5.87 (3.38)	4.87 (2.23)	**4.11 (2.75)**	4.38 (3.3)
GP	**1 (0)**	**1 (0)**	**1 (0)**	**1 (0)**	0.472 (0.375)	0.406 (0.396)	**0.402 (0.337)**	0.448 (0.384)
Lim	0.943 (0.0786)	**0.982 (0.0103)**	0.968 (0.0275)	0.966 (0.0486)	0.549 (0.522)	**0.385 (0.437)**	0.553 (0.464)	0.606 (0.486)
Moon	**1 (0.000583)**	**1 (0.000667)**	**1 (0.000333)**	**1 (0)**	0.0945 (0.00674)	**0.0787 (0.00388)**	0.0852 (0.00434)	0.0881 (0.00683)
Morokoff & Caflisch	**0.847 (0.0584)**	0.833 (0.0672)	0.82 (0.0869)	0.82 (0.0576)	0.403 (0.121)	0.33 (0.0741)	0.344 (0.0812)	**0.276 (0.0751)**
Oakley & OH́agan	**1 (0)**	**1 (0)**	**1 (0)**	**1 (0)**	9.69e-05 (6.21e-06)	**8.79e-05 (1.79e-06)**	**8.79e-05 (2.57e-06)**	9.04e-05 (3.39e-06)
OTL	0.981 (0.0205)	0.981 (0.0128)	**0.987 (0.0139)**	0.98 (0.0168)	0.0191 (0.00107)	**0.0176 (0.000762)**	0.0189 (0.00106)	0.0187 (0.000827)
Piston	0.924 (0.0373)	0.934 (0.0276)	**0.937 (0.024)**	0.926 (0.0356)	0.0781 (0.0106)	**0.0732 (0.00696)**	0.0776 (0.0114)	0.076 (0.0104)
Product Peak	0.847 (0.0764)	0.808 (0.0986)	**0.752 (0.0918)**	0.879 (0.0805)	30.9 (21.7)	19.7 (16.3)	**6.11 (6.42)**	21.8 (18.8)
Schwefel	0.794 (0.122)	**0.712 (0.114)**	0.779 (0.0869)	0.767 (0.112)	0.604 (0.117)	**0.511 (0.0996)**	0.578 (0.0877)	0.582 (0.13)
Trid	0.999 (0.00175)	0.999 (0.0019)	0.998 (0.00155)	**1 (8e-04)**	0.569 (0.102)	0.533 (0.109)	0.536 (0.0814)	**0.527 (0.0615)**

Table B.3: Median (IQR) of performance metrics computed for design comparison at $n = 15d$.

Test Function	Measures of Predictive Performance							
	RMSPE				NSE			
	LHD	MaxPro	S-optimal	Maximin	LHD	MaxPro	S-optimal	Maximin
Bohachevsky 1	272 (111)	**184 (28.3)**	210 (56)	186 (36.2)	0.998 (0.00141)	**0.999 (0.000231)**	**0.999 (0.000535)**	**0.999 (0.000313)**
Borehole	0.663 (0.151)	**0.58 (0.124)**	0.614 (0.144)	0.608 (0.113)	**1 (9.26e-05)**	**1 (7.03e-05)**	**1 (8.66e-05)**	**1 (6.6e-05)**
Branin	3.83 (2.3)	**2.54 (0.868)**	2.76 (1.61)	2.69 (1.24)	0.994 (0.00694)	**0.998 (0.0017)**	0.997 (0.0038)	0.997 (0.00262)
Bratley	0.0186 (0.00316)	0.0162 (0.00279)	0.0163 (0.00211)	**0.0159 (0.00276)**	0.994 (0.00208)	0.995 (0.00175)	0.995 (0.00111)	**0.996 (0.00157)**
Cheng & Sandu	0.0124 (0.00957)	0.00987 (0.00278)	0.0106 (0.00394)	**0.00878 (0.00479)**	**0.999 (0.00203)**	**0.999 (4e-04)**	**0.999 (0.000589)**	**0.999 (0.000709)**
Corner-Peak	0.00479 (0.00284)	0.00447 (0.00288)	**0.00452 (0.00267)**	0.00482 (0.00277)	0.165 (0.141)	**0.256 (0.206)**	0.224 (0.189)	0.193 (0.248)
Currin	0.276 (0.106)	**0.205 (0.0404)**	0.252 (0.115)	0.272 (0.138)	0.989 (0.00926)	**0.994 (0.0026)**	0.991 (0.00929)	0.989 (0.0107)
Franke	0.011 (0.00293)	**0.00917 (0.00215)**	0.011 (0.0022)	0.00999 (0.00288)	**0.999 (0.000792)**	**0.999 (0.000454)**	**0.999 (0.000587)**	**0.999 (0.000707)**
Friedman	0.347 (0.0801)	**0.28 (0.0435)**	0.316 (0.0836)	0.297 (0.054)	0.995 (0.00237)	**0.997 (0.0011)**	0.996 (0.00247)	0.996 (0.00138)
G-function	0.505 (0.07)	0.497 (0.0517)	0.514 (0.0837)	**0.469 (0.0576)**	0.653 (0.101)	0.664 (0.0628)	0.642 (0.128)	**0.701 (0.0795)**
GP	0.117 (0.0537)	**0.0705 (0.03)**	0.0863 (0.0258)	0.0739 (0.0269)	0.999 (0.000513)	**1 (0.000193)**	**1 (0.000202)**	**1 (0.000182)**
Lim	0.0917 (0.0399)	**0.0658 (0.0122)**	0.0839 (0.0186)	0.0817 (0.0262)	0.998 (0.00198)	**0.999 (0.000406)**	0.998 (0.000815)	0.998 (0.00111)
Moon	0.00834 (0.0027)	**0.00606 (0.00154)**	0.00706 (0.00184)	0.00639 (0.00141)	**1 (5.34e-05)**	**1 (2.23e-05)**	**1 (3.17e-05)**	**1 (2.11e-05)**
Morokoff & Caflisch	0.0877 (0.017)	**0.0735 (0.0069)**	0.0857 (0.0202)	0.0832 (0.00982)	0.941 (0.0206)	**0.959 (0.00793)**	0.944 (0.0272)	0.947 (0.0133)
Oakley & OH́agan	6.45e-06 (2.06e-06)	**5.47e-06 (1.17e-06)**	5.92e-06 (1.44e-06)	5.91e-06 (1.81e-06)	**1 (8.62e-08)**	**1 (3.93e-08)**	**1 (5.08e-08)**	**1 (6.75e-08)**
OTL	0.0148 (0.0024)	**0.0125 (0.00165)**	0.0139 (0.00194)	0.0143 (0.00148)	**1 (5.69e-05)**	**1 (2.98e-05)**	**1 (4.25e-05)**	**1 (3.47e-05)**
Piston	0.00886 (0.00157)	**0.00791 (0.00115)**	0.00823 (0.00125)	0.00827 (0.00136)	0.996 (0.00136)	**0.997 (0.000966)**	0.996 (0.00101)	0.996 (0.00121)
Product Peak	323000 (118000)	304000 (144000)	384000 (70600)	**302000** (198000)	0.513 (0.409)	0.53 (0.481)	**0.204 (0.315)**	0.544 (0.816)
Schwefel	450 (16.8)	450 (22)	**447 (22.3)**	455 (25.4)	**-0.0696 (0.0788)**	-0.08 (0.113)	-0.0708 (0.119)	-0.108 (0.115)
Trid	36.5 (4.47)	32.3 (4.3)	32.9 (4.12)	**28.3 (3.08)**	0.996 (0.000932)	0.997 (0.000764)	0.997 (0.000662)	**0.998 (0.000531)**

Test Function	Measures of Prediction Variability							
	ECP				RIL			
	LHD	MaxPro	S-optimal	Maximin	LHD	MaxPro	S-optimal	Maximin
Bohachevsky 1	0.998 (0.009)	**1 (0.00238)**	**1 (0.0035)**	**1 (0)**	0.663 (0.387)	**0.523 (0.247)**	0.551 (0.285)	0.58 (0.272)
Borehole	0.977 (0.0162)	**0.982 (0.0143)**	0.98 (0.0158)	0.981 (0.0132)	0.0584 (0.00475)	**0.0564 (0.00457)**	0.0577 (0.00394)	0.0585 (0.00337)
Branin	0.986 (0.02)	0.99 (0.00738)	0.99 (0.0159)	**0.994 (0.00937)**	2.29 (1.82)	2.03 (1.33)	**1.98 (1.35)**	2.12 (1.17)
Bratley	0.948 (0.0446)	0.956 (0.0319)	0.96 (0.0187)	**0.97 (0.0277)**	1.2 (0.968)	1.07 (0.649)	**0.983 (0.321)**	1.06 (0.528)
Cheng & Sandu	0.974 (0.0379)	0.981 (0.0119)	0.981 (0.0129)	**0.987 (0.0169)**	0.225 (0.0942)	**0.187 (0.0859)**	0.19 (0.106)	0.208 (0.135)
Corner Peak	0.885 (0.0719)	0.894 (0.0417)	0.893 (0.0443)	**0.9 (0.0376)**	**22.1 (28.2)**	24.4 (23.2)	25.5 (34.6)	57.2 (72.2)
Currin	0.942 (0.0541)	**0.963 (0.0224)**	0.95 (0.0397)	0.942 (0.0415)	0.134 (0.0305)	**0.106 (0.011)**	0.11 (0.0201)	0.117 (0.0242)
Franke	**0.977 (0.0209)**	0.969 (0.0178)	0.969 (0.024)	0.973 (0.0116)	0.246 (0.0389)	**0.215 (0.0239)**	0.236 (0.0442)	0.234 (0.0436)
Friedman	0.992 (0.014)	0.996 (0.00675)	**0.991 (0.0157)**	0.995 (0.0112)	0.13 (0.00594)	**0.119 (0.0038)**	0.123 (0.00579)	0.124 (0.00369)
G-function	**0.84 (0.0902)**	0.81 (0.0842)	0.772 (0.0902)	0.838 (0.0725)	4.96 (3.29)	4.55 (3.36)	**4.52 (2.47)**	4.78 (2.41)
GP	**1 (0)**	**1 (0)**	**1 (0)**	**1 (0)**	0.322 (0.327)	**0.219 (0.154)**	0.253 (0.159)	0.261 (0.149)
Lim	0.98 (0.0284)	**0.99 (0.00975)**	0.98 (0.0171)	0.978 (0.0274)	0.283 (0.299)	**0.254 (0.18)**	0.31 (0.309)	0.305 (0.276)
Moon	**1 (0.001)**	**1 (0.000333)**	**1 (0.000333)**	**1 (0)**	0.0553 (0.00319)	**0.0479 (0.00182)**	0.052 (0.00194)	0.0524 (0.0022)
Morokoff & Caflisch	0.866 (0.042)	**0.894 (0.0372)**	0.864 (0.0456)	0.839 (0.0406)	0.31 (0.0924)	0.282 (0.0513)	**0.32 (0.0666)**	0.276 (0.0974)
Oakley & OH́agan	**1 (5e-04)**	**1 (5e-04)**	**1 (5e-04)**	**1 (0.001)**	5.21e-05 (2.3e-06)	4.94e-05 (9.87e-07)	**4.88e-05 (1.55e-06)**	4.93e-05 (1.65e-06)
OTL	0.981 (0.0174)	**0.989 (0.00796)**	0.986 (0.0165)	0.981 (0.0123)	0.0126 (0.000765)	0.0121 (0.000415)	**0.0124 (0.000499)**	0.0125 (0.000481)
Piston	0.926 (0.0308)	**0.937 (0.0199)**	0.932 (0.021)	0.933 (0.0277)	0.0592 (0.0097)	**0.0564 (0.00561)**	0.0584 (0.00638)	0.0583 (0.00678)
Product Peak	0.904 (0.0754)	0.91 (0.0494)	0.816 (0.0561)	**0.94 (0.0523)**	34.5 (22.3)	37 (22.2)	**12.6 (14)**	34.8 (38.3)
Schwefel	**0.834 (0.0686)**	0.799 (0.0905)	0.826 (0.101)	0.791 (0.0891)	0.646 (0.0972)	**0.586 (0.0934)**	0.625 (0.102)	0.598 (0.106)
Trid	0.998 (0.00155)	0.998 (0.0022)	0.998 (0.0014)	**1 (0.00095)**	0.348 (0.0378)	**0.328 (0.0589)**	0.349 (0.0795)	0.347 (0.0723)

Table B.4: Median (IQR) of performance metrics computed for design comparison at $n = 20d$.

Test Function	Measures of Predictive Performance							
	RMSPE				NSE			
	LHD	MaxPro	S-optimal	Maximin	LHD	MaxPro	S-optimal	Maximin
Bohachevsky 1	174 (62.4)	**109 (20)**	146 (36.6)	113 (37.1)	0.999 (0.000515)	**1 (9.56e-05)**	**1 (0.000237)**	**1 (0.00021)**
Borehole	0.453 (0.0634)	0.451 (0.108)	0.464 (0.0929)	**0.427 (0.0865)**	**1 (2.68e-05)**	**1 (4.71e-05)**	**1 (4.4e-05)**	**1 (3.74e-05)**
Branin	2.33 (1.53)	**1.69 (0.515)**	2.16 (0.972)	1.73 (1.06)	0.998 (0.00294)	**0.999 (0.000694)**	0.998 (0.00151)	**0.999 (0.00138)**
Bratley	0.0141 (0.00281)	0.0123 (0.00148)	0.0126 (0.00201)	**0.0119 (0.00179)**	0.997 (0.00136)	0.997 (0.00068)	0.997 (0.000871)	**0.998 (0.000759)**
Cheng & Sandu	0.00907 (0.00529)	**0.00681 (0.00266)**	0.00763 (0.00452)	0.00698 (0.00212)	0.999 (0.000703)	**1 (0.000278)**	**1 (0.000536)**	**1 (0.000211)**
Corner-Peak	0.00481 (0.00315)	**0.00476 (0.00251)**	0.00478 (0.00248)	0.00523 (0.00373)	0.259 (0.23)	**0.299 (0.222)**	0.228 (0.175)	0.241 (0.262)
Currin	0.221 (0.065)	**0.157 (0.0421)**	0.169 (0.0445)	0.174 (0.0697)	0.993 (0.00393)	**0.996 (0.00178)**	**0.996 (0.00209)**	**0.996 (0.00376)**
Franke	0.0234 (0.00758)	**0.0163 (0.00202)**	0.018 (0.0038)	0.0175 (0.00324)	0.993 (0.00433)	**0.997 (0.000758)**	0.996 (0.00176)	0.996 (0.00153)
Friedman	0.215 (0.0548)	**0.178 (0.0309)**	0.202 (0.0372)	0.194 (0.0284)	0.998 (0.00103)	**0.999 (0.000451)**	0.998 (0.000633)	0.998 (0.000476)
G-function	0.437 (0.052)	0.42 (0.0731)	0.448 (0.0513)	**0.388 (0.0473)**	0.747 (0.0724)	0.766 (0.0709)	0.732 (0.0635)	**0.8 (0.0442)**
GP	0.067 (0.0353)	**0.0393 (0.0137)**	0.052 (0.0198)	0.044 (0.0121)	**1 (0.000184)**	**1 (4.38e-05)**	**1 (8.74e-05)**	**1 (5.16e-05)**
Lim	0.0621 (0.0118)	**0.0439 (0.00681)**	0.0537 (0.011)	0.0525 (0.014)	**0.999 (0.000455)**	**0.999 (0.000131)**	**0.999 (0.000319)**	**0.999 (0.000388)**
Moon	0.00553 (0.00162)	**0.00396 (0.000729)**	0.00458 (0.00102)	0.00407 (0.000926)	**1 (2.11e-05)**	**1 (6.39e-06)**	**1 (1.1e-05)**	**1 (8.79e-06)**
Morokoff & Caflisch	0.0732 (0.0112)	**0.0664 (0.00451)**	0.0706 (0.00828)	0.0733 (0.00979)	0.959 (0.0133)	**0.967 (0.00461)**	0.962 (0.009)	0.959 (0.0115)
Oakley & OH́agan	4.84e-06 (1.34e-06)	**3.78e-06 (4.9e-07)**	4.37e-06 (9.3e-07)	4.56e-06 (7.47e-07)	**1 (3.7e-08)**	**1 (1.07e-08)**	**1 (2.45e-08)**	**1 (2e-08)**
OTL	0.00996 (0.0019)	**0.00929 (0.00105)**	0.00947 (0.0015)	0.00936 (0.00137)	**1 (2.85e-05)**	**1 (1.5e-05)**	**1 (2.24e-05)**	**1 (2.01e-05)**
Piston	0.00687 (0.00103)	**0.00657 (0.000781)**	0.00682 (0.00122)	0.00666 (0.00122)	**0.998 (0.000715)**	**0.998 (0.00051)**	**0.998 (0.000875)**	**0.998 (0.000891)**
Product Peak	3e+05 (87000)	281000 (105000)	341000 (65300)	**257000 (95500)**	0.588 (0.244)	0.643 (0.301)	0.458 (0.173)	**0.691 (0.202)**
Schwefel	**440 (20.6)**	447 (20.5)	443 (15.8)	445 (19.3)	**-0.0296 (0.0958)**	-0.0663 (0.118)	-0.0449 (0.0859)	-0.0576 (0.0926)
Trid	25.4 (3.71)	23.9 (3.18)	25.3 (4.44)	**20.6 (2.26)**	0.998 (0.000547)	0.998 (0.000423)	0.998 (0.00064)	**0.999 (0.000282)**

Test Function	Measures of Prediction Variability							
	ECP				RIL			
	LHD	MaxPro	S-optimal	Maximin	LHD	MaxPro	S-optimal	Maximin
Bohachevsky 1	0.998 (0.0065)	**1 (0)**	**1 (0.0025)**	**1 (5e-04)**	0.433 (0.375)	**0.295 (0.145)**	0.418 (0.234)	0.401 (0.213)
Borehole	0.982 (0.00969)	0.98 (0.00934)	0.979 (0.0141)	**0.984 (0.00925)**	0.0431 (0.00273)	**0.0413 (0.00258)**	0.0426 (0.00343)	0.0435 (0.00302)
Branin	**0.992 (0.0181)**	0.99 (0.00487)	0.989 (0.0111)	0.994 (0.0124)	1.71 (0.984)	**1.53 (0.765)**	1.73 (0.788)	1.62 (0.891)
Bratley	0.957 (0.0293)	0.96 (0.0165)	0.961 (0.0199)	**0.975 (0.0197)**	1.04 (0.395)	0.823 (0.416)	**0.782 (0.395)**	0.843 (0.446)
Cheng & Sandu	0.978 (0.0308)	**0.985 (0.0112)**	0.982 (0.0235)	0.984 (0.0119)	0.17 (0.166)	**0.145 (0.0812)**	0.148 (0.0704)	0.152 (0.0646)
Corner Peak	0.905 (0.0371)	**0.921 (0.0401)**	0.919 (0.0279)	0.913 (0.0264)	39.4 (56.6)	32.1 (48)	**27.9 (36.8)**	47.3 (75)
Currin	0.948 (0.0375)	**0.972 (0.0159)**	0.964 (0.0319)	0.953 (0.0324)	0.0989 (0.0207)	**0.0808 (0.00523)**	0.0926 (0.0107)	0.0866 (0.0167)
Franke	0.969 (0.0245)	0.97 (0.0175)	0.971 (0.0218)	**0.974 (0.0231)**	0.501 (0.133)	**0.355 (0.0453)**	0.407 (0.0764)	0.419 (0.0715)
Friedman	0.998 (0.00735)	**0.999 (0.00215)**	0.998 (0.00485)	0.998 (0.0036)	0.0958 (0.00402)	**0.0876 (0.00298)**	0.0918 (0.00229)	0.0905 (0.00228)
G-function	0.865 (0.0493)	0.841 (0.0811)	0.804 (0.0884)	**0.871 (0.0576)**	4.57 (1.98)	4.95 (2.27)	**4.33 (1.85)**	4.52 (2.06)
GP	**1 (0)**	**1 (0)**	**1 (0)**	**1 (0)**	0.19 (0.163)	0.163 (0.124)	0.162 (0.123)	**0.15 (0.0881)**
Lim	0.985 (0.0214)	**0.992 (0.00775)**	0.99 (0.0161)	0.987 (0.0168)	0.169 (0.121)	0.165 (0.115)	0.166 (0.0945)	**0.153 (0.0702)**
Moon	**1 (0.000667)**	**1 (0)**	**1 (0.000333)**	**1 (0)**	0.0388 (0.00238)	**0.0335 (0.00136)**	0.0367 (0.00216)	0.0361 (0.00196)
Morokoff & Caflisch	0.89 (0.0376)	**0.894 (0.0333)**	0.888 (0.0428)	0.859 (0.0378)	0.264 (0.0762)	**0.239 (0.0463)**	0.28 (0.048)	0.24 (0.043)
Oakley & OH́agan	**1 (0.000875)**	**1 (0.001)**	**1 (0.0015)**	**1 (0.001)**	3.43e-05 (1.43e-06)	3.32e-05 (7.58e-07)	**3.25e-05 (8.16e-07)**	3.27e-05 (1.18e-06)
OTL	0.987 (0.0118)	0.988 (0.00763)	0.989 (0.011)	**0.99 (0.00908)**	0.00936 (0.000341)	**0.0088 (0.00018)**	0.00929 (0.000295)	0.00923 (0.000238)
Piston	0.936 (0.0229)	**0.94 (0.0218)**	0.935 (0.0216)	0.935 (0.0176)	0.0476 (0.00511)	**0.0453 (0.00377)**	0.0465 (0.00416)	0.0474 (0.00352)
Product Peak	0.922 (0.0384)	0.933 (0.0325)	0.87 (0.0587)	**0.955 (0.0364)**	42.4 (27.5)	47.8 (37.8)	**22 (13.8)**	49.3 (37.9)
Schwefel	**0.874 (0.0489)**	0.806 (0.0665)	0.846 (0.0622)	0.843 (0.0741)	0.677 (0.0945)	**0.602 (0.0812)**	0.641 (0.0554)	0.631 (0.078)
Trid	0.998 (0.00155)	0.997 (0.0018)	0.997 (0.00255)	**0.999 (8e-04)**	0.249 (0.0348)	**0.234 (0.021)**	0.246 (0.0326)	0.248 (0.0304)

Appendix C

Test Functions

Table C.1: Test functions used for emulation and design comparison in this work. Implementations of all functions in R were retrieved from the Virtual Library of Simulation Experiments [63].

Test Function Name	**D**	$\boldsymbol{f(x)}$
Currin et al. [17] Sinusoidal Function	1	$f(x) = \sin(2\pi(x - 0.1))$
Forrester et al. [22] Function	1	$f(x) = (6x - 2)^2 \sin(12x - 4)$
Higdon [33] and Gramacy and Lee [29] Function	1	$f(x) = \begin{cases} \sin\left(\frac{\pi x}{5}\right) + 0.2\cos\left(\frac{4\pi x}{5}\right), & \text{if } x < 10 \\ \frac{x}{10} - 1, & \text{otherwise} \end{cases}$
Bohachevsky Function 1	2	$f(x) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1) - 0.4\cos(4\pi x_2) + 0.7$
Branin Function	2	$f(x) = a(x_2 - bx_1^2 + cx_1 - r^2) + s(1 - t)\cos(x_1) + s$
Cheng & Sandu [14] Function	2	$f(x) = \cos(x_1 + x_2)\exp(x_1 x_2)$
Currin et al. [17] Exponential Function	2	$f(x) = \left[1 - \exp - \frac{1}{2x_2}\right] \frac{2300x_1^3 + 1900x_1^2 + 2092x_1 + 60}{100x_1^3 + 500x_1^2 + 4x_1 + 20}$

Continued on next page

Table C.1 – continued from previous page

Test Function Name	D	$f(x)$
Franke's Function [25]	2	$f(x) = 0.75\exp\left(-\frac{(9x_1-2)^2}{4} - \frac{(9x_2-2)^2}{4}\right) + 0.75\exp\left(-\frac{(9x_1+1)^2}{49} - \frac{(9x_2+12}{10}\right) + 0.5\exp\left(-\frac{(9x_1-7)^2}{4} - \frac{(9x_2-3)^2}{4}\right) - 0.2\exp -(9x_1-4)^2 - (9x_2-7)^2$
Lim et al. [44] Function	2	$f(x) = 9 + \frac{5}{2}x_1 - \frac{35}{2}x_2 + \frac{5}{2}x_1x_2 + 19x_2^2 - \frac{15}{2}x_1^3 - \frac{5}{2}x_1x_2^2 - \frac{11}{2}x_2^2 + x_1^3x_2^2$
Oakley and O'Hagan [56] 2-Dimensional Function	2	$f(x) = 5 + x_1 + x_2 + 2\cos(x_1) + 2\sin(x_2)$
Williams et al. [67] Function	2	$f(x) = (x_1+1)\cos(\pi x_2) + 0x_3$
Bratley et al. [8] Function	3	$f(x) = \sum_{i=1}^{d}(-1)^i \prod_{j=1}^{i} x_j$
Corner Peak Integrand Family [28]	3	$f(x) = \left(1 + \sum_{i=1}^{d} a_i x_i\right)^{-(d+1)}$
Gaussian Process Realization	3	
G-Function [59]	4	$f(x) = \prod_{i=1}^{d} \frac{\|4x_i-2\|+(i-2)/2}{1+(i-1)/2}$
Moon [52] Low-Dimensionality Function	3	$f(x) = x_1 + x_2 + 3x_1x_3$
Friedman [26] Function	5	$f(x) = 10\sin(\pi x_1 x_2) + 20(x_3-0.5)^2 + 10x_4 + 5x_5$
Product Peak Integrand Family [28]	5	$f(x) = \prod_{i=1}^{d} \frac{1}{(a_i^{-2} + (x_i-u_i)^2)}$
Schwefel [43] Function	5	$f(x) = 418.9828d - \sum_{i=1}^{d} x_i \sin(\sqrt{\|x_i\|})$
Morokoff and Caflisch [53] Function 1	6	$f(x) = (1+1/d)^d \prod_{i=1}^{d} x_i^{1/d}$

Continued on next page

Table C.1 – continued from previous page

Test Function Name	**D**	$\boldsymbol{f(x)}$
OTL Circuit [6] Function	6	$V_m(x) = \frac{(12R_{b2}/(R_{b1}R_{b2}))\beta(R_{c2}+9)}{\beta(R_{c2+9)+R_f}} + \frac{11.35R_f}{\beta(R_{c2}+9)+R_f} + \frac{0.74R_f\beta(R_{c2}+9)}{(\beta(R_{c2}+9)+R_f)R_{c1}}$
Piston Simulation [70]	7	$C(x) = 2\pi\sqrt{\frac{M}{k+S^2\frac{P_0V_0}{T_0}\frac{T_a}{V^2}}},$ where $V = \frac{S}{2k}\left(\sqrt{A^2+4k\frac{P_0V_0}{T_0}T_a} - A\right)$ $A = P_0S + 19.62M - kV_0/S$
Borehole Function [32]	8	$f(x) = \frac{2\pi T_u(H_u-H_l)}{\ln(r/r_w)\left(1+\frac{2LT_u}{\ln(r/r_w)r_w^2K_w}+\frac{T_u}{T_l}\right)}$