University of Southampton

Faculty of Social Sciences
School of Mathematical Sciences

Topics in Design of Computer Experiments

by

Hendriico Merila
ORCID ID 0009-0006-4693-1267

A thesis for the degree of
Doctor of Philosophy

August 2025

University of Southampton

Abstract

Faculty of Social Sciences
School of Mathematical Sciences

Doctor of Philosophy

Topics in Design of Computer Experiments

by Hendriico Merila

Computer models are used in many fields to simulate real-world processes. One of the goals is to optimise the value of the computer model. Due to the fact that the computer model is usually expensive to evaluate, one can only make a limited number of evaluations of this computer model. Using Gaussian processes in sequential design through the use of acquisition functions is a common approach for sample-efficient optimisation in such cases. Despite many recent successes, there are still a number of outstanding problems in that field. In this work, we address some of these problems.

We first give a brief overview of Bayesian optimisation and the common techniques used. After that, we focus on scenarios where one is interested in finding both the minimum and the maximum of the computer model simultaneously. We use the entropy of the location of the optima to define a sequential design algorithm. The design is then created in a way that would minimise the entropy. Monte Carlo methods are used to approximate a number of probability distributions. The resulting algorithm is then compared against a baseline algorithm to demonstrate its superior performance.

In the second part of this paper, we are interested in optimising high-dimensional computer models. This is a complicated task and comes with a number of additional challenges compared to the standard problems. Our focus is on computer models whose accuracy we can control by changing the amount of computational resources allocated to them. This is also often referred to as multi-fidelity optimisation. We then use a number of techniques from mathematical optimisation to define a multi-fidelity optimisation algorithm that can be used in high-dimensional settings and scaled to large number of evaluations. Its performance is then compared to that of another state-of-the-art optimisation algorithm.

In the final part of this paper, we explore computer models that are non-stationary. These are computer models whose properties change depending on which part of the design space it is evaluated. For example, a computer model that changes rapidly in one area and is completely flat in a different area is non-stationary. Such computer models can accurately be modelled with a non-stationary Gaussian process. However, fitting non-stationary Gaussian processes can be computationally very expensive and a large number of computer model evaluations is often needed to model the process accurately. This makes sequential design difficult. We create a novel acquisition function that allows us to create accurate sequential designs by using regular stationary Gaussian processes that are far easier to fit. Our acquisition function puts more emphasis on more interesting regions and less emphasis on less interesting regions. We then create sequential designs created by our novel acquisition function combined with stationary Gaussian processes and compare it with designs found by non-stationary Gaussian processes.

Contents

List of Figures

List of Tables

Definitions and Abbreviations

ALC	Active Learning Cohn
ALM	Active Learning MacKay
BO	Bayesian Optimisation
CDF	Cumulative Distribution Function
DoE	Design of Experiments
EI	Expected Improvement
ES	Entropy Search
GP	Gaussian Process
MC	Monte Carlo
MCMC	Markov Chain Monte Carlo
MSE	Mean Squared Error
NSALC	Non-Stationary Active Learning MacKay
OCBA	Optimal Computing Budget Allocation
PDF	Probability Density Function
PLBO	Partial Local Bayesian Optimisation
TR	Trust Region
TS	Thompson Sampling
UCB	Upper Confidence Bound

Chapter 1

Introduction to Design of Computer Experiments

Experiments underpin the scientific method. Physical experiments are used to understand the underlying processes behind many phenomena, often via the fitting of models to the observed data. However, running a physical experiment is often infeasible. It can be unethical (e.g. investigating the spread of disease through a population as in Herrmann et al. (2024)), too expensive (e.g. exploring the dynamics of a rocket booster as in Pamadi et al. (2004a)) or simply impossible (e.g. galaxy formation simulations as in Vogelsberger et al. (2019)).

In such cases, a computer model can be built. A computer model is a mathematical model that simulates real-world phenomena using the latest scientific knowledge. In this paper, we treat the computer model as a black-box taking inputs as arguments and returning an output. Computer models can be used to understand the underlying phenomena, for example, by performing a sensitivity analysis to identify the most important inputs. However, computer models are often computationally expensive to evaluate (see, for example Santner et al., 2003), complicating their usage for the aforementioned task.

Usually, one would like to learn as much information about this function as possible. Since this computer model is expensive to evaluate, we can only make a limited number of evaluations of it. To learn about this computer model, we perform a computer experiment. Similarly to a physical experiment, we create an experimental design where we collect data. Instead of collecting data through the physical process directly, we collect it by evaluating the computer model at each of the design points. The evaluated points can be used to fit a surrogate model, which aims to approximate the computer model for any set of inputs. Despite the popularity and success of this approach in many applications, several key challenges still remain to make Bayesian optimisation even more broadly applicable and robust in more complex and realistic settings. This thesis will focus on a number these challenges.

We begin by giving an overview of Bayesian optimisation. Bayesian optimisation has become a widely adopted approach for sample-efficient optimisation in expensive evaluation scenarios. This framework typically uses Gaussian Processes as a surrogate to estimate the underlying computer model and quantify uncertainty. Sequential design combined with various acquisition functions is used to intelligently balance exploration and exploitation to find the global optimum. Several different algorithms and acquisition functions are introduced and compared.

In the first part of this work we consider the case where both the minimum and the maximum of the computer model are of interest. These situations occur frequently in design and risk assessment tasks, where extreme values can inform worst-case and best-case analyses. This is an area that has received little to no attention in the past compared to single-objective optimisation. To address this problem, we draw inspiration from information theory and propose a new algorithm based on entropy. More specifically, we use the joint entropy of both optima locations to define a new acquisition function for multi-task Bayesian Optimisation. This acquisition function chooses a new point in a way that minimises the uncertainty about both locations. Since the exact calculation of this acquisition function is generally intractable, we employ Monte Carlo techniques to approximate the necessary probability distributions. Our algorithm is then evaluated against a baseline algorithm to demonstrate its superior ability to efficiently identify both optima more accurately and with fewer computer model evaluations.

The second part focuses on optimising high-dimensional computer models. Traditional Bayesian Optimisation algorithms struggle in higher dimensions due to curse of dimensionality. Curse of dimensionality makes it very difficult to fit Gaussian Processes accurately, which means standard algorithms that rely purely on the fit of the Gaussian Process are unable to select informative sample points. We specifically focus on multi-fidelity optimisation, where the accuracy of model evaluations can be controlled by varying the amount of computational resources. A low-fidelity evaluation will be much cheaper and quicker but also less accurate, while high-fidelity evaluation can be very accurate but also very expensive to run. We develop a novel algorithm that both chooses the location of the points to evaluate, as well as the fidelities at which to evaluate them. Our novel algorithm is run against an existing state-of-the-art algorithm to demonstrate that our algorithm finds better results using a significantly lower number of computer model evaluations.

Finally, we consider sequential design for non-stationary computer models whose behaviour varies significantly across different regions of the input space. For example, a model might exhibit sharp changes in one region while remaining flat in another. Standard Gaussian process models assume stationarity, which can lead to overexploring more boring regions and underexploring the more interesting regions, leading to poor performance when the stationarity assumption is violated. While non-stationary Gaussian processes can offer more flexibility and greater accuracy, they are computationally very

expensive to train. They typically also require many model evaluations to accurately capture the underlying process and its non-stationarity, undermining the benefits of sequential design. We therefore propose a novel acquisition function that can be used with standard stationary GPs, while mimicing the behaviour of non-stationary models by adapting the sampling strategy to focus more on complex, informative regions of the input space. This is achieved by essentially reweighing the design space to focus on the more interesting areas. This approach allows for efficient sequential design without the overhead of fitting a full non-stationary GP. We validate our approach by comparing sequential designs generated from our method to those obtained via fully non-stationary GP models. Our experiments demonstrate competitive or superior performance with only a fraction of the computational cost.

Overall, our work contributes several new algorithms and ideas to the field of design of computer experiments. Each proposed method is motivated by real-world challenges and is thoroughly evaluated by comparing against existing techniques. Each part addresses a limitation of current capabilities, broadening the applicability of sequential design methods.

1.1 Gaussian process

As stated previously, it is assumed that the computer model is computationally expensive to evaluate. To understand the behaviour of the computer model it is common to perform a computer experiment. This is where the computer model is evaluated at a carefully chosen, *small* number, n, of inputs. Then an emulator is fitted to the resulting outputs. The emulator, a form of statistical model, can then predict the value of the output for any inputs, including a measure of uncertainty. Consequently, the emulator can be used as an alternative to the computer model for tasks such as sensitivity analyses, and, of interest in this paper, optimisation.

Let the computer model be denoted by $f : \mathbb{X} \to \mathbb{R}$. It is a function of a $d \times 1$ vector of inputs $\mathbf{x} = (x_1, \ldots, x_d) \in \mathbb{X}$, where $\mathbb{X} \subset \mathbb{R}^d$ is the input space. Let $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ denote the set of n inputs chosen for the computer experiment. Let $\mathbf{y} = (y_1, \ldots, y_n)$ denote the $n \times 1$ vector of corresponding computer model outputs, i.e. $y_i = f(\mathbf{x}_i)$, for $i = 1, \ldots, n$. We allow the computer model to be stochastic, meaning, in general, $y_i \neq y_j$ for $\mathbf{x}_i = \mathbf{x}_j$, for $i, j = 1, \ldots, n$. In that case, we can also write our computer model as $f(\mathbf{x}) = g(\mathbf{x}) + \epsilon(\mathbf{x})$, where $g(\mathbf{x})$ is a deterministic function and $\epsilon(\mathbf{x})$ is some unknown stochastic function. We would then like to emulate the true deterministic function $g(\mathbf{x})$.

The most common type of emulator is the Gaussian process emulator (Fang et al., 2005; Santner et al., 2003). A Gaussian process is a stochastic process whereby any finite collection of variables has a multivariate normal distribution. We use a constant-mean

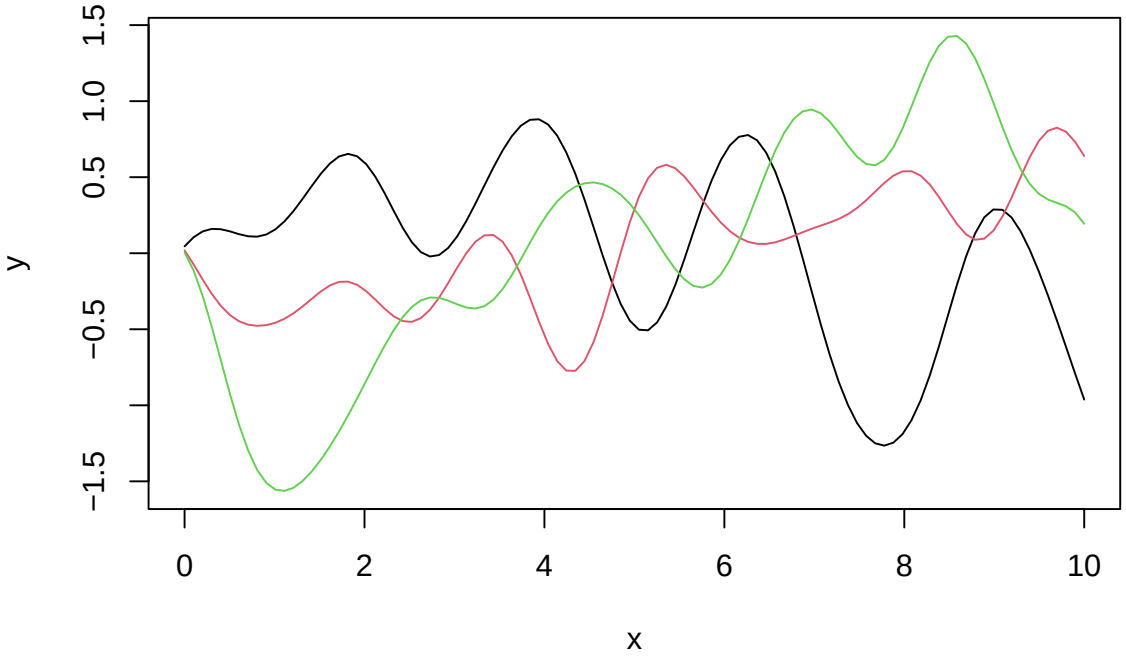

FIGURE 1.1: Examples of a Gaussian Process prior

Gaussian process where

$$\mathbf{y} \sim \mathrm{N}\left[\mu \mathbf{1}_n, \sigma^2 K(X, X)\right]. \tag{1.1}$$

An example of a Gaussian Process prior can be seen in Figure 1.1

In Equation (1.1), $\mathbf{1}_n$ denotes the $n \times 1$ vector of ones and $K(X, X)$ is an $n \times n$ matrix with ijth element $K(X, X)_{ij} = \rho I(i = j) + r(\mathbf{x}_i, \mathbf{x}_j; \boldsymbol{\theta})$, for $i, j = 1, \ldots, n$, where $I(i = j)$ is an indicator function meaning $I(i = j) = \begin{cases} 1 \text{ if } i = j \\ 0 \text{ if } i \neq j \end{cases}$. The term $r(\cdot, \cdot; \boldsymbol{\theta})$ is a correlation function depending on a vector of parameters $\boldsymbol{\theta}$ with the properties that $r(\mathbf{x}, \mathbf{x}'; \boldsymbol{\theta}) \in (0, 1)$, for $\mathbf{x} \neq \mathbf{x}'$, and $r(\mathbf{x}, \mathbf{x}; \boldsymbol{\theta}) = 1$. A common choice for $r(\cdot, \cdot; \boldsymbol{\theta})$ is a Gaussian correlation function

$$r(\mathbf{x}, \mathbf{x}'; \boldsymbol{\theta}) = \exp\left[-\sum_{k=1}^{d} \frac{(x_k - x'_k)^2}{\theta_k}\right], \tag{1.2}$$

where $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_d)$. The Gaussian correlation function is used throughout this paper. However, the methodologies proposed in the rest of this thesis can be combined with any choice of correlation function. The parameter $\boldsymbol{\theta}$ is called the lengthscale parameter and it controls how strong the correlation between two points $\mathbf{x}$ and $\mathbf{x}'$ is in $r(\mathbf{x}, \mathbf{x}'; \boldsymbol{\theta})$. This in turn controls the overall smoothness of the Gaussian Process. Examples on how the

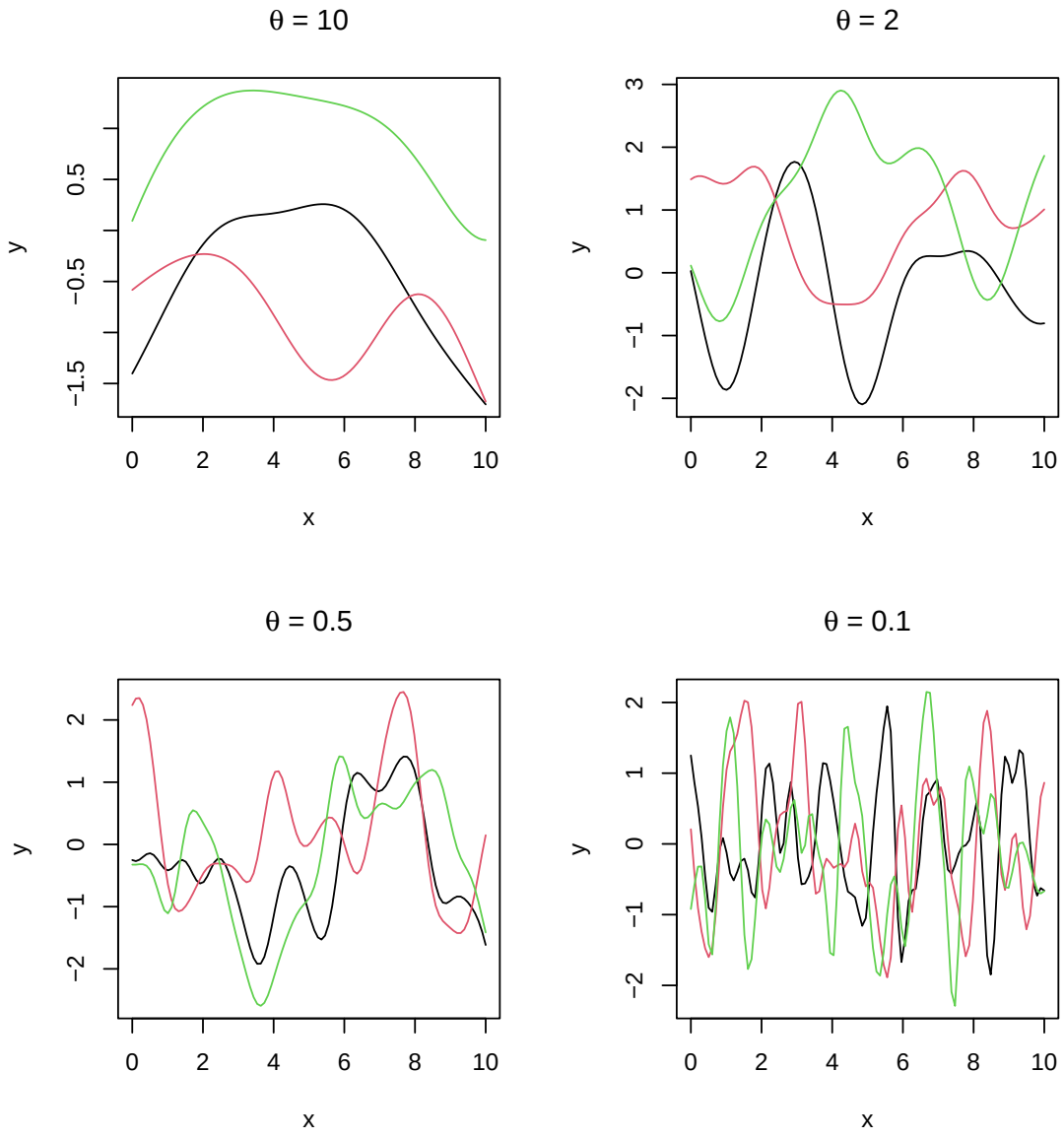

FIGURE 1.2: Gaussian Process priors under different lengthscales θ

choice of $\boldsymbol{\theta}$ affects the GP are seen in Figure 1.2. The larger $\boldsymbol{\theta}$ means that the correlation between different points is smaller and the GP is smoother and less wiggly. Smaller $\boldsymbol{\theta}$ means stronger correlation between points that are close, which makes the GP more wiggly with rapid changes.

The quantity $\rho \geq 0$ is sometimes called the nugget and represents the underlying output noise of a stochastic computer model. The nugget can be set to zero for deterministic computer models. However, Gramacy and Lee (2010) suggest retaining a non-zero nugget, even for deterministic computer models, as it can improve the predictive performance of the emulator, for example, with respect to the coverage of prediction intervals.

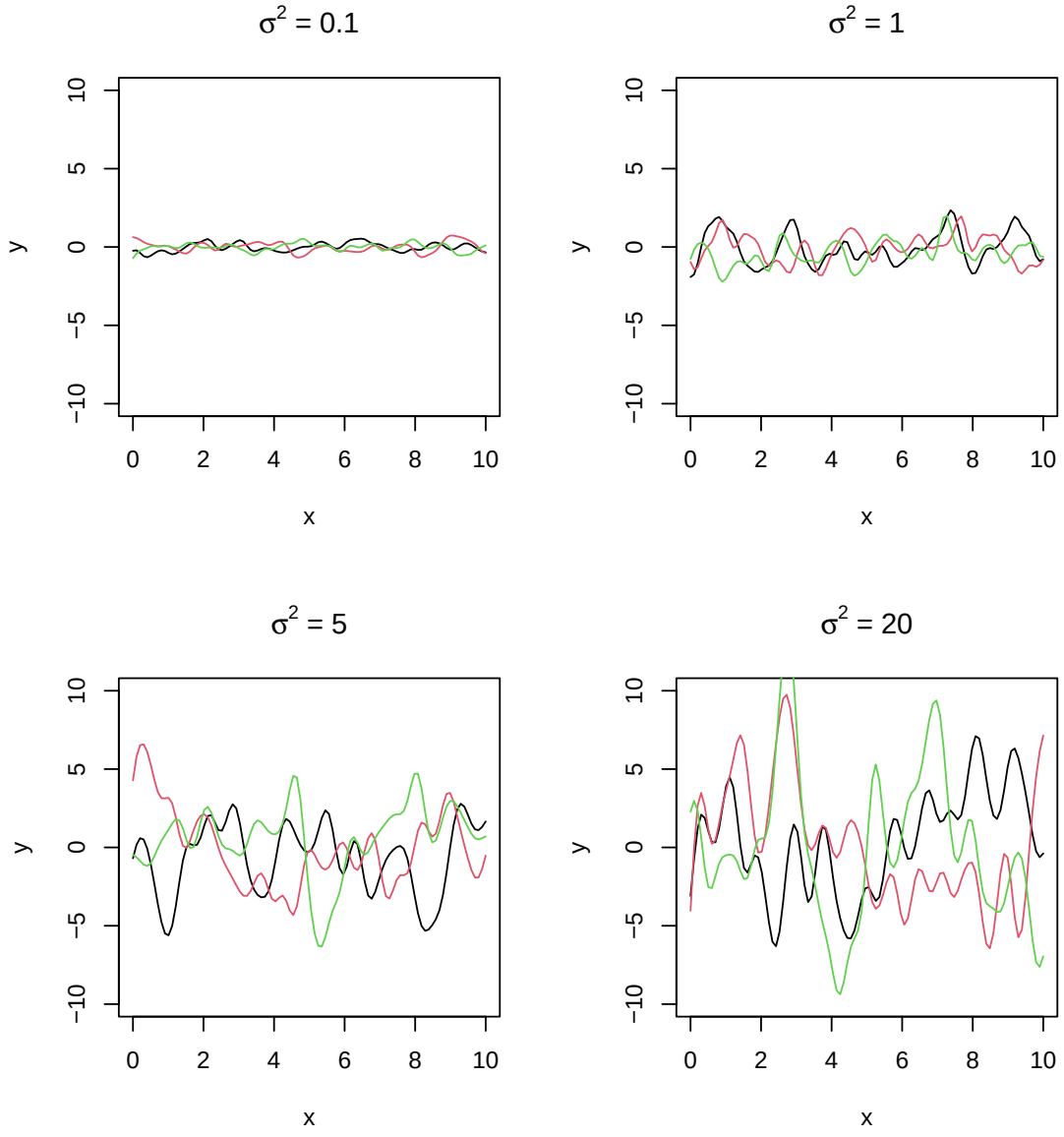

FIGURE 1.3: Gaussian Process priors under different scales σ^2

The parameter σ^2 is the scale parameter, which controls the magnitude of the GP. Examples of different GP priors with different σ^2 are shown in Figure 1.3.

Suppose we wish to predict the values of the computer model for m inputs in the set $\bar{X} = \{\bar{\mathbf{x}}_1, \ldots, \bar{\mathbf{x}}_m\}$. Let $\bar{\mathbf{y}} = (\bar{y}_1, \ldots, \bar{y}_m)$ be the $m \times 1$ vector with $\bar{y}_j = f(\bar{\mathbf{x}}_j)$, for $j = 1, \ldots, m$. It follows from the above specification of the Gaussian process that the joint distribution of $\mathbf{y}$ and $\bar{\mathbf{y}}$ is

$$\begin{pmatrix} \mathbf{y} \\ \bar{\mathbf{y}} \end{pmatrix} \sim \mathrm{N}\left[\mu \mathbf{1}_{n+m}, \sigma^2 \begin{pmatrix} K(X,X) & K(X,\bar{X}) \\ K(X,\bar{X})^T & K(\bar{X},\bar{X}) \end{pmatrix}\right],$$

where $K(X, \bar{X})$ is an $n \times m$ matrix with ijth element $K(X, \bar{X}) = r(\mathbf{x}_i, \bar{\mathbf{x}}_j; \boldsymbol{\theta})$, for

$i = 1, \ldots, n$ and $j = 1, \ldots, m$, and $K(\bar{X}, \bar{X})$ is an $m \times m$ matrix with ijth element $K(\bar{X}, \bar{X})_{ij} = \rho I(i = j) + r(\bar{\mathbf{x}}_i, \bar{\mathbf{x}}_j; \boldsymbol{\theta})$, for $i, j = 1, \ldots, m$.

Using properties of the multivariate normal distribution the conditional distribution of $\bar{\mathbf{y}}$, given $\mathbf{y}$, is

$$\bar{\mathbf{y}}|\mathbf{y} \sim \mathrm{N}\left[\mu(\bar{X}), \Sigma(\bar{X})\right], \tag{1.3}$$

where

$$\begin{aligned} \mu(\bar{X}; X, \mathbf{y}) &= \mu \mathbf{1}_m + K(X, \bar{X})^T K(X, X)^{-1} (\mathbf{y} - \mu \mathbf{1}_n) && (1.4) \\ \Sigma(\bar{X}; X, \mathbf{y}) &= \sigma^2 \left[K(\bar{X}, \bar{X}) - K(X, \bar{X})^T K(X, X)^{-1} K(X, \bar{X})\right]. && (1.5) \end{aligned}$$

The distribution in (1.3) can be used to predict the elements of $\bar{\mathbf{y}} = (\bar{y}_1, \ldots, \bar{y}_m)$. For example, the expectation can be used as a point prediction and the variance a measure of uncertainty. The distribution in (1.3) depends on the quantities μ, ρ, σ^2 and $\boldsymbol{\theta}$. We estimate these parameters using maximum likelihood, where the likelihood function follows from the model specification given by (1.1).

A special case of (1.3) arises when $m = 1$, i.e. we are predicting the value of the computer model, $y = f(\mathbf{x})$, at a single input $\mathbf{x}$, in which case $y|\mathbf{y} \sim \mathrm{N}\left[\mu(\mathbf{x}), \Sigma(\mathbf{x})\right]$ where

$$\begin{aligned} \mu(\mathbf{x}; X, \mathbf{y}) &= \mu + \mathbf{k}(X, \mathbf{x})^T K(X, X)^{-1} (\mathbf{y} - \mu \mathbf{1}_n) \\ \Sigma(\mathbf{x}; X, \mathbf{y}) &= \sigma^2 \left[1 + \rho - \mathbf{k}(X, \mathbf{x})^T K(X, X)^{-1} \mathbf{k}(X, \mathbf{x})\right], \end{aligned}$$

where $\mathbf{k}(X, \mathbf{x})$ is an $n \times 1$ vector with ith element $r(\mathbf{x}_i, \mathbf{x}; \boldsymbol{\theta})$ for $i = 1, \ldots, n$.

Imagine we have a computer model of the form

$$f_A(x) = 0.4 \sin\left(20(x + 0.5)(x - 0.5)\right) + 5(x - 0.5)^2 - 0.5; \text{ for } x \in (0, 1) \tag{1.6}$$

We can then choose a random design X of size $n = 10$ and evaluate $\mathbf{y} = f(X)$. Based off of X and $\mathbf{y}$, we estimate our parameters μ, ρ, σ^2 and $\boldsymbol{\theta}$. This will allow us to make predictions and calculate $\mu(\bar{X}; X, \mathbf{y})$ and $\Sigma(\bar{X}; X, \mathbf{y})$. These are shown in Figure 1.4. Note that in this example the maximum likelihood estimates are $\theta \approx 0.066$, $\rho \approx 0.093$, $\mu \approx 0.047$ and $\sigma^2 \approx 0.68$.

The fit of the GP depends on all the hyperparameters, especially $\boldsymbol{\theta}$ and ρ as they change both the mean and variance. As mentioned before, $\boldsymbol{\theta}$ changes the wiggliness of the Gaussian Process. This is illustrated in Figure 1.5.

In Figure 1.5, we fit 4 GPs to the same design, all with different values of θ. In all 4 scenarios, we have fixed $\rho = 10^{-4}$. Under this value of ρ, the maximum likelihood estimate of θ is $\theta \approx 0.01$. As expected, the GPs with a higher value of θ are more smooth

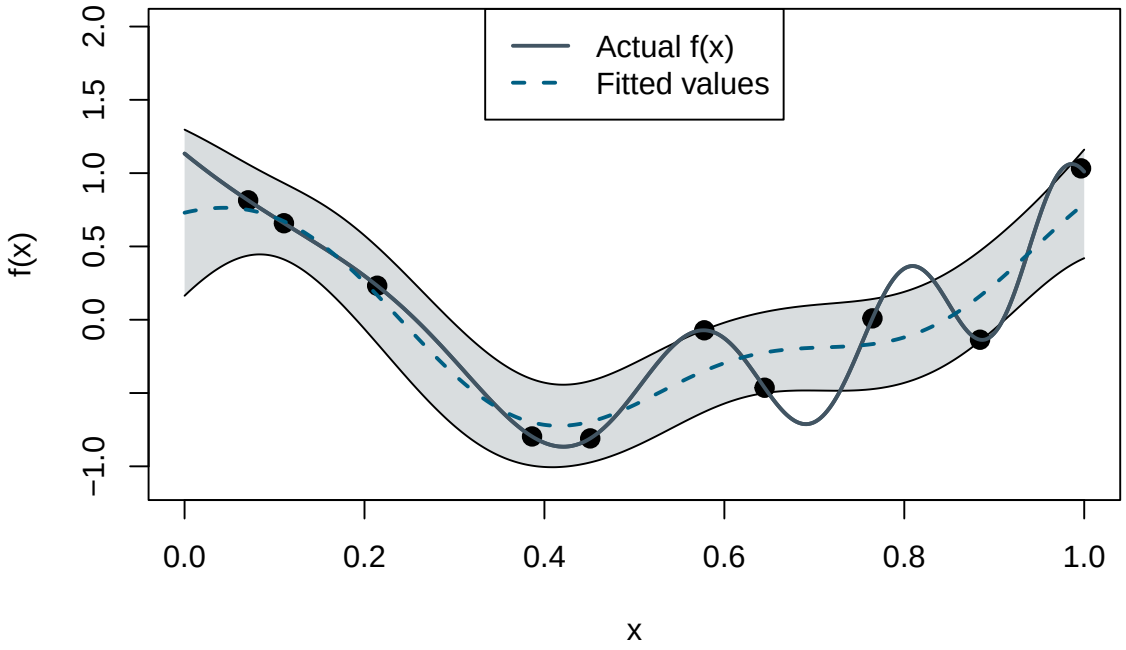

FIGURE 1.4: GP fit to $f_A(x)$

and do not interpolate the evaluated points, whereas the GPs with a lower value of θ are more wiggly and interpolate the evaluated points.

The parameter ρ is also found in both the mean as well as the variance of the GP. It models the level of noise in the computer model. Therefore low levels of ρ means that the GP mean interpolates the evaluated points, while high levels of ρ means that the GP mean does not interpolate the evaluated points and rather smooths over them. In Figure 1.6, four different GP fits are shown. Each of them has a fixed $\theta = 0.01$. As we can see, the GPs with a low ρ interpolate our design and high ρ smooths.

1.2 Bayesian Optimisation

Very often our goal is to minimise (or maximise) the computer model and find

$$\mathbf{x}_* \in \arg\min_{\mathbf{x}\in\mathbb{X}} f(\mathbf{x})$$

Since $f(\mathbf{x})$ is expensive to evaluate, we can only make a limited number of evaluations of it. What is more, the computer model can also be noisy, which makes it impossible for us to numerically approximate its gradient. Because of those reasons, using

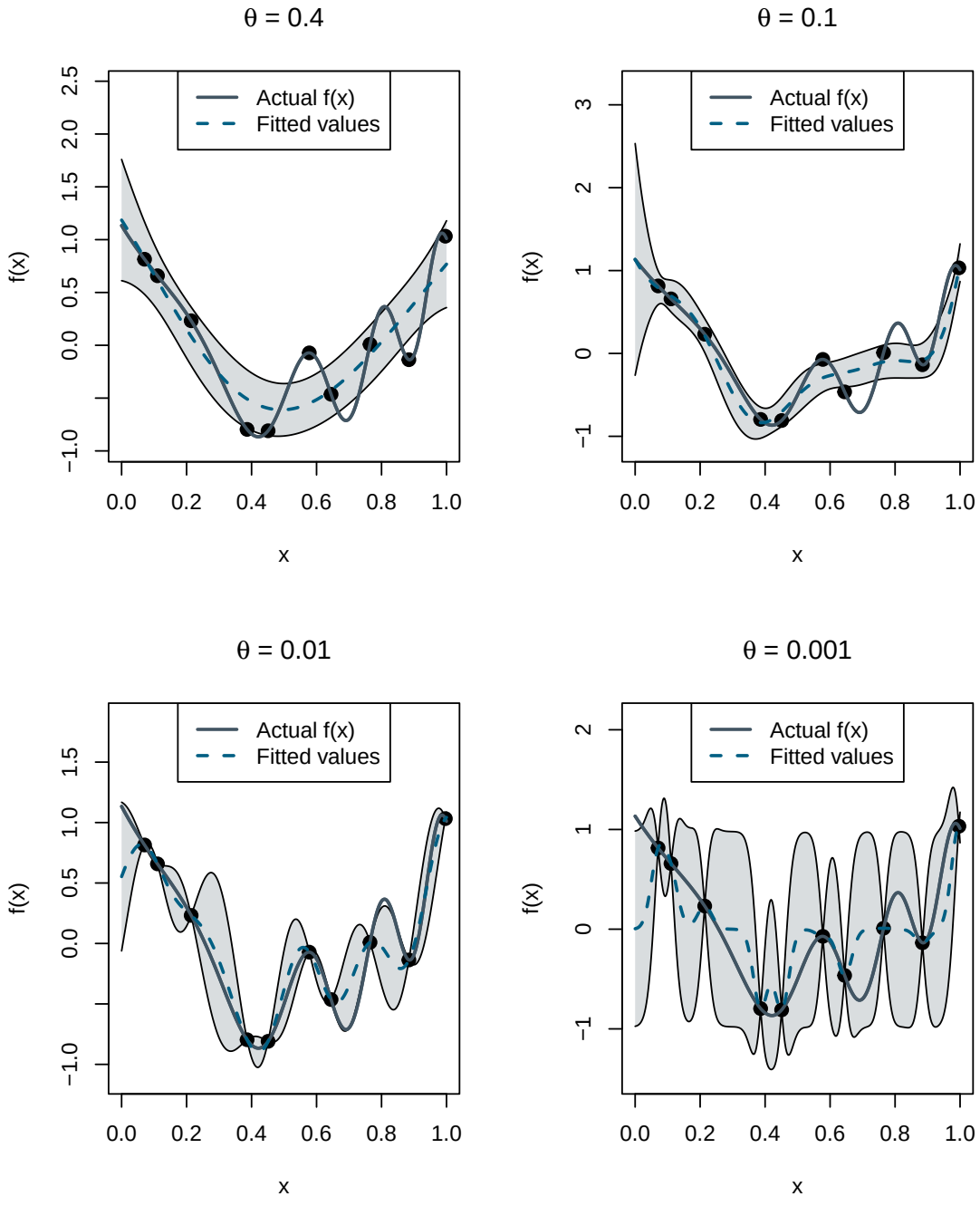

FIGURE 1.5: Gaussian Process posteriors under different lengthscales θ

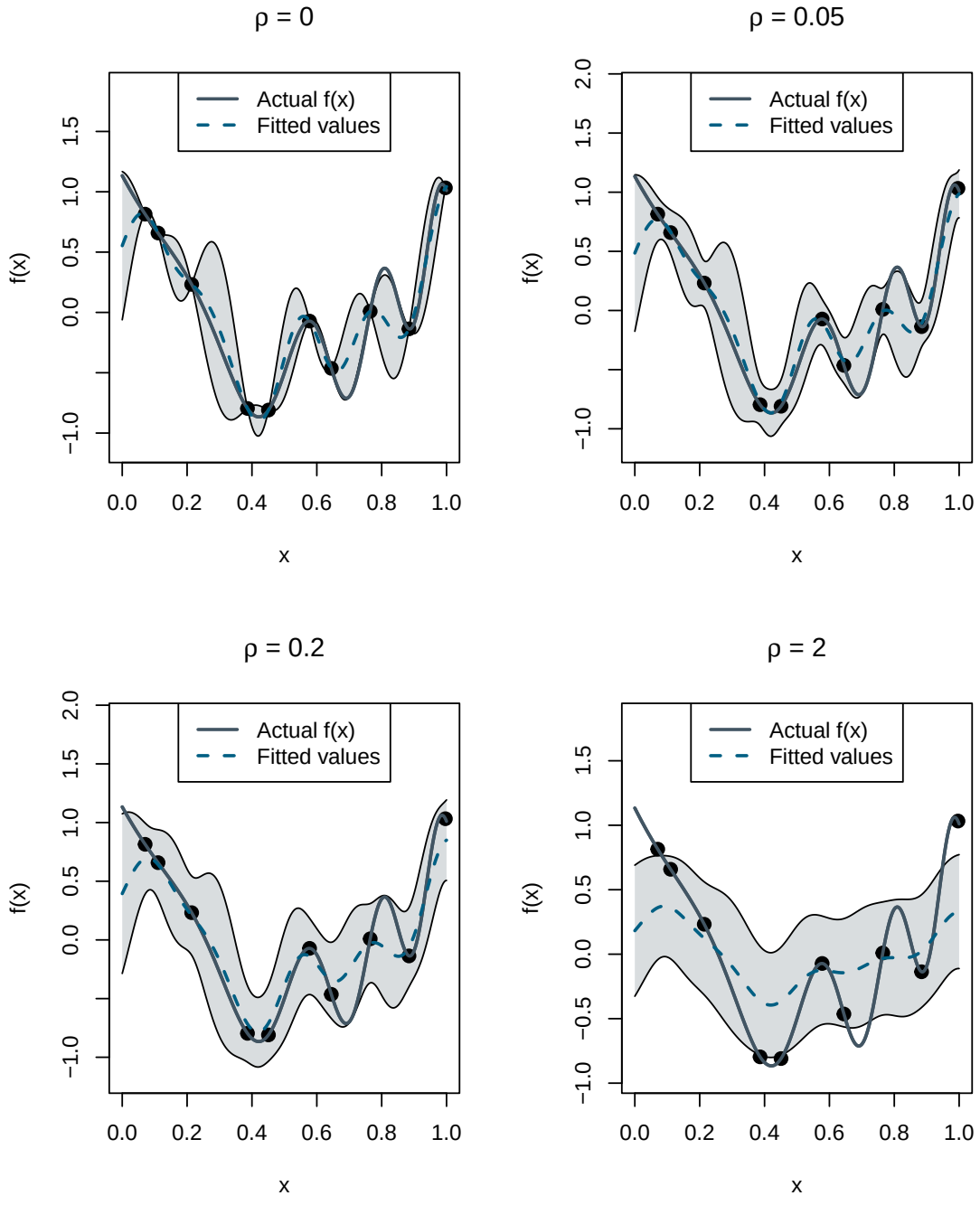

FIGURE 1.6: Gaussian Process posteriors under different scales ρ

traditional numerical optimisation algorithms (such as Newton-Raphson, Nelder-Mead, BFGS, etc.) are not an option. Instead, various Bayesian optimisation algorithms have been proposed to handle this problem (Jones et al. (1998); Picheny et al. (2013); Huang et al. (2006b); Frazier et al. (2009); Forrester (2013); Quan et al. (2013)). Each of those algorithms perform a computer experiment. Similarly to a physical experiment, we create an experimental design with N values and collect data for that design, by evaluating $f(\mathbf{x})$ at each of these N values. We then use that data to build a surrogate model that approximates the value of $f(\mathbf{x})$ for any other input value. In our case, we approximate $f(\mathbf{x})$ by $Y(\mathbf{x})$.

The choice of the inputs in the design $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ is crucial. In the early days of analysing computer experiments, the goal of running the computer experiment was to provide an accurate and precise prediction of the computer model for any $\mathbf{x} \in \mathbb{X}$. Designs such as space-filling designs were commonly-used (McKay et al., 1979; Garud et al., 2017a). However, recently model based sequential designs have been recommended as they provide superior performance compared to the static designs (Garud et al., 2017b; Crombecq et al., 2011; Provost et al., 1999). Initially, we consider optimisation, which is the problem of finding

$$\mathbf{x}_{min} \in \arg\min_{\mathbf{x}\in\mathbb{X}} f(\mathbf{x}).$$

Note that it is straightforward to extend this approach to maximisation tasks by minimising the negative of the computer model. Such a goal means we favour designs that place inputs "close" to $\mathbf{x}_{min}$. However, *a-priori*, we do not know $\mathbf{x}_{min}$. This motivates a sequential design approach where we sequentially update our knowledge about $\mathbf{x}_{min}$. A generic sequential design algorithm is shown in Algorithm 1. We begin with an initial, e.g. space-filling, design. After fitting a Gaussian process emulator to design and outputs, we sequentially augment the design with new inputs. These inputs are chosen to maximise an acquisition function, which is based on the current Gaussian process fit. The key to the algorithm is the choice of acquisition function. We now discuss a series of algorithms given by different choices of acquisition function.

Algorithm 1: A generic sequential optimisation algorithm

Require: An acquisition function $A(\cdot)$, a computer model $f(\cdot)$, maximum number of computer model evaluations available N and an initial design $X = (\mathbf{x}_1, ..., \mathbf{x}_n)$ of size n, where $n < N$

Let $\boldsymbol{y} = (y_1, ..., y_n)$, where $y_i = f(\mathbf{x}_i)$ for $i = 1, ..., n$;
Fit an initial Gaussian process model with X and $\boldsymbol{y}$;
while $n \leq N$ **do**
    Let $\mathbf{x}_{n+1} = \arg\max_{\mathbf{x}\in\mathbb{X}} A(\mathbf{x})$;
    Evaluate $y_{n+1} = f(\mathbf{x}_{n+1})$;
    Let $X := (X, \mathbf{x}_{n+1})$ and $\boldsymbol{y} = (\boldsymbol{y}, y_{n+1})$;
    Let $n := n + 1$;
    Update the Gaussian process model with X and $\boldsymbol{y}$
end
Return $\hat{\mathbf{x}}_{min} = \arg\min_{\mathbf{x}\in\mathbb{X}} \mu(\mathbf{x}; X, \mathbf{y})$ where $\mu(\cdot; X, \mathbf{y})$ is given in (1.6).

1.2.1 Expected improvement

One of the most well-known algorithms for sequential optimisation is *efficient global optimisation* (EGO; Jones et al. 1998). The acquisition function A is *expected improvement* (Schonlau, 1997); a measure of how much one can expect to improve the current minimum, by evaluating the computer model at a new point. It is defined as

$$A_{EI}(\mathbf{x}) = \mathrm{E}\left[\max(y_{\min} - f(\mathbf{x}), 0)\right],$$

where $y_{\min} = \min_{i\in\{1,..,n\}} y_i$ is the currently observed minimum.

If we model the computer model by a Gaussian process emulator, then

$$A_{EI}(\mathbf{x}) = (y_{\min} - \mu(\mathbf{x}; X, \mathbf{y}))\Phi\left(\frac{y_{\min} - \mu(\mathbf{x}; X, \mathbf{y})}{\sqrt{\Sigma(\mathbf{x}; X, \mathbf{y})}}\right) + \sqrt{\Sigma(\mathbf{x}; X, \mathbf{y})}\phi\left(\frac{y_{\min} - \mu(\mathbf{x}; X, \mathbf{y})}{\sqrt{\Sigma(\mathbf{x}; X, \mathbf{y})}}\right), \tag{1.7}$$

where Φ and ϕ are the cumulative distribution function (cdf) and probability density function (pdf) of the standard normal distribution, respectively.

Expected improvement balances exploitation and exploration by its two terms. The first term, given by $(y_{\min} - \mu(\mathbf{x}; X, \mathbf{y}))\Phi\left[(y_{\min} - \mu(\mathbf{x}; X, \mathbf{y}))/\sqrt{\Sigma(\mathbf{x}; X, \mathbf{y})}\right]$, increases as the predictive mean decreases, causing the algorithm to exploit those areas with low predictive mean. The second term, given by $\sqrt{\Sigma(\mathbf{x}; X, \mathbf{y})}\phi\left[(y_{\min} - \mu(\mathbf{x}; X, \mathbf{y}))/\sqrt{\Sigma(\mathbf{x}; X, \mathbf{y})}\right]$, increases with the predictive variance, causing the algorithm to explore those areas with high predictive variance, i.e. where there is large uncertainty about the value of the computer model. Indeed, some authors, for example, Sóbester et al. (2005), have weighted the two terms in (1.7) to prioritise exploitation or exploration.

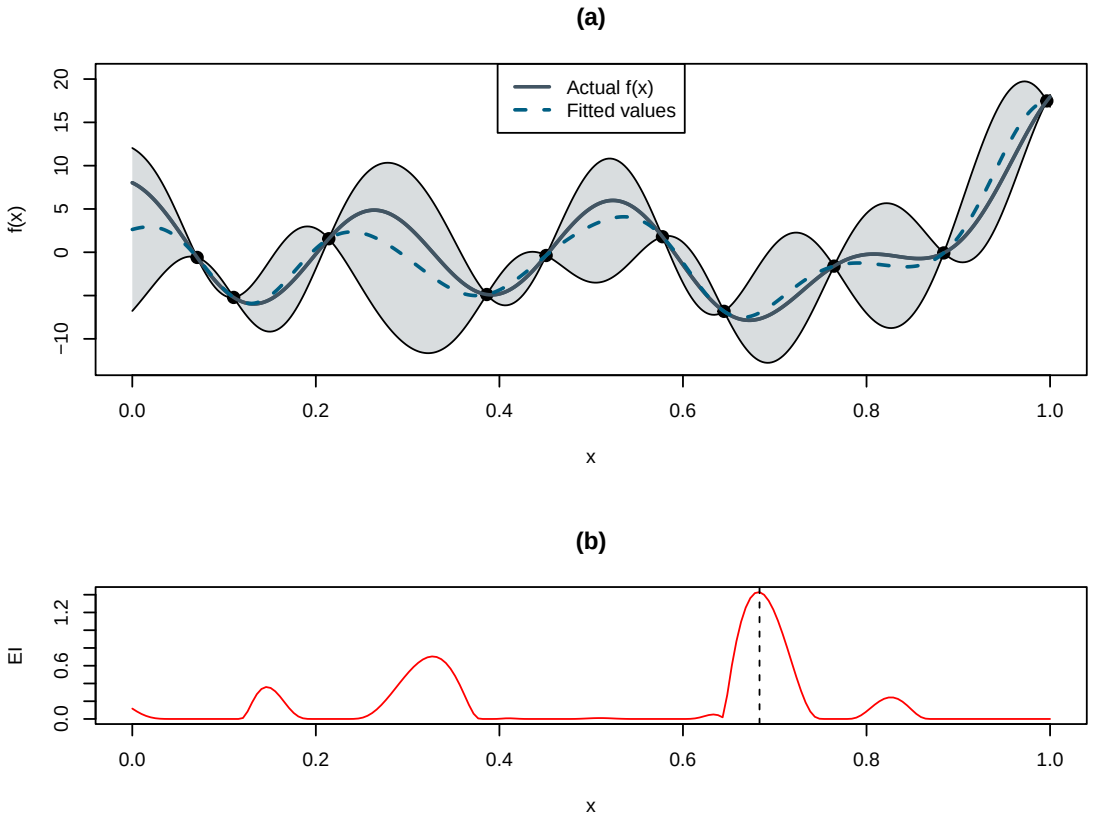

FIGURE 1.7: (a) Example of a GP fitted to $f_B(x)$ (b) Expected Improvement function evaluated

We can similarly define EI for maximisation problems as follows:

$$EI_{\max}(\mathbf{x}) = E\left[\max(f(\mathbf{x}) - y_{\max}, 0)\right], \tag{1.8}$$

where $y_{\max} = \max_{i\in(1,..,n)} y_i$ is the currently observed maximum.

Imagine we have a computer model $f_B(x)$ defined in $x \in (0, 1)$, which is given as

$$f_B(x) = (6x - 2)^2 \cdot \sin(12x - 4) + 5\cos(24x)$$

and that we would like to minimise. We then choose an initial design of size $n = 10$ and fit a GP, this can be seen in Figure 1.7 (a). We then use Equation 1.7 to calculate expected improvement for all $x \in (0, 1)$. This can be seen in Figure 1.7 (b).

In the case of a stochastic computer model, every element of $(y_1, ..., y_n)$ is a random variable, meaning $y_{\min}$ (or $y_{\max}$) is also a random variable. To fully take into account all uncertainty about $y_{\min}/y_{\max}$ is a difficult task. For this reason, expected improvement is not well defined for stochastic computer models.

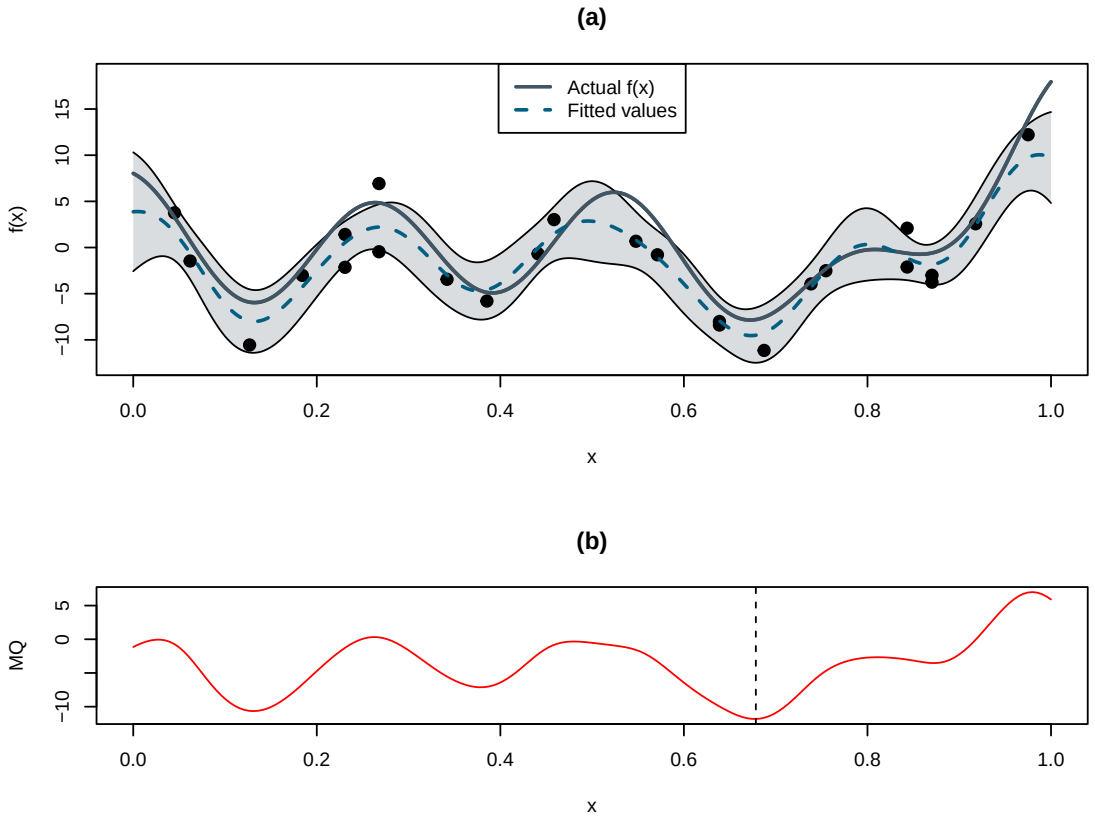

FIGURE 1.8: (a) Example of a noisy GP fitted to $f_C(x)$ (b) Minimum Quantile function evaluated

1.2.1.1 Minimum quantile

The minimum quantile (MQ; Picheny et al. 2013) algorithm has acquisition function $A_{MQ}(\mathbf{x}) = q(\mathbf{x}; X, \mathbf{y})$, where $q(\mathbf{x}; X, \mathbf{y}) = \mu(\mathbf{x}; X, \mathbf{y}) + \Phi^{-1}(\beta)\sqrt{\Sigma(\mathbf{x}; X, \mathbf{y})}$, for tuning parameter $\beta \in (0, 0.5]$ and where $\Phi^{-1}(\cdot)$ is the quantile function of a standard normal distribution. Typically $\beta = 0.1$, but it can also be varied throughout the iterations of the sequential design algorithm. In Srinivas et al. (2010), β is an increasing function of the number of iterations, meaning that in the early steps of the algorithm, the impact of $\sqrt{\Sigma(\mathbf{x}; X, \mathbf{y})}$ is large and the algorithm focuses on exploration. However, as β approaches $1/2$, $q(\mathbf{x})$ approaches $\mu(\mathbf{x}; X, \mathbf{y})$ and the algorithm focuses on exploitation.

Imagine we now have a stochastic computer model, which is given as

$$\begin{aligned} f_C(x) &= f_B(x) + \epsilon \\ &= (6x-2)^2 \cdot \sin(12x-4) + 5\cos(24x) + \epsilon, \end{aligned} \tag{1.9}$$

where $\epsilon \sim N(0, 2.5^2)$. We then choose a random design of size $n = 25$ and fit a GP. This can be seen in Figure 1.8 (a). We can then evaluate A_{MQ} for all $x \in (0, 1)$, which can be seen in Figure 1.8.

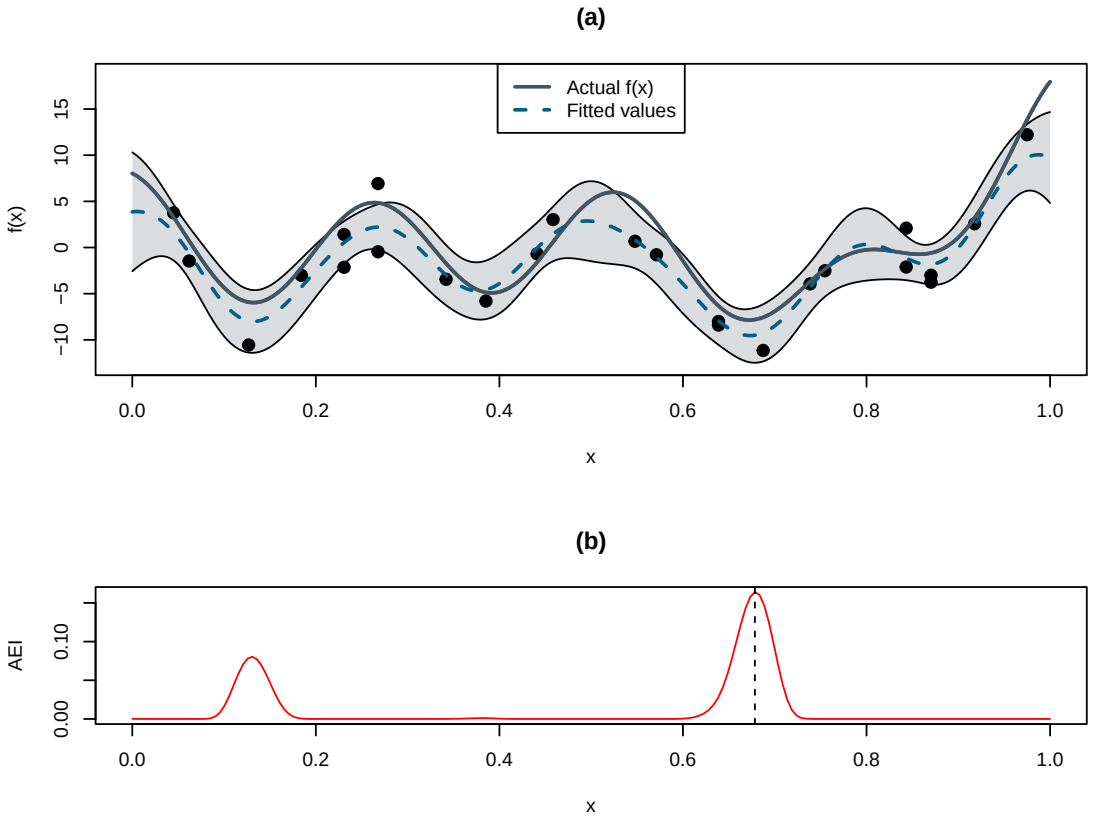

FIGURE 1.9: (a) Example of a noisy GP fitted to $f_C(x)$ (b) Augmented Expected Improvement function evaluated

1.2.1.2 Augmented expected improvement

Sequential kriging optimisation (SKO; Huang et al. 2006b) uses *augmented expected improvement* (AEI) as the acquisition function where

$$A_{AEI}(\mathbf{x}; X, \mathbf{y}) = \mathrm{E}\left[\max(\mu(\mathbf{x}^*_{min}; X, \mathbf{y}) - f(\mathbf{x}), 0)\right]\left(1 - \frac{\rho}{\sqrt{\Sigma(\mathbf{x}; X, \mathbf{y}) + \rho}}\right),$$

where $\mathbf{x}^*_{min} = \arg\min_{\mathbf{x}\in X} q(\mathbf{x}; X, \mathbf{y})$, with $\beta \in [0.5, 1)$. The first term is very similar to expected improvement apart from how $y_{\min}$ is defined. The second term $1 - \rho/\sqrt{\Sigma(\mathbf{x}; X, \mathbf{y}) + \rho}$ can be interpreted as a correction term encouraging exploration. If the computer model is deterministic and $\rho = 0$, then $A_{EI}(\mathbf{x}) = A_{AEI}(\mathbf{x})$, meaning efficient global optimisation and sequential kriging optimisation coincide.

Going back to our computer model in Equation 1.9, we can evaluate A_{AEI} for all $x \in (0, 1)$, with $\beta = 0.9$, which can be seen in Figure 1.9.

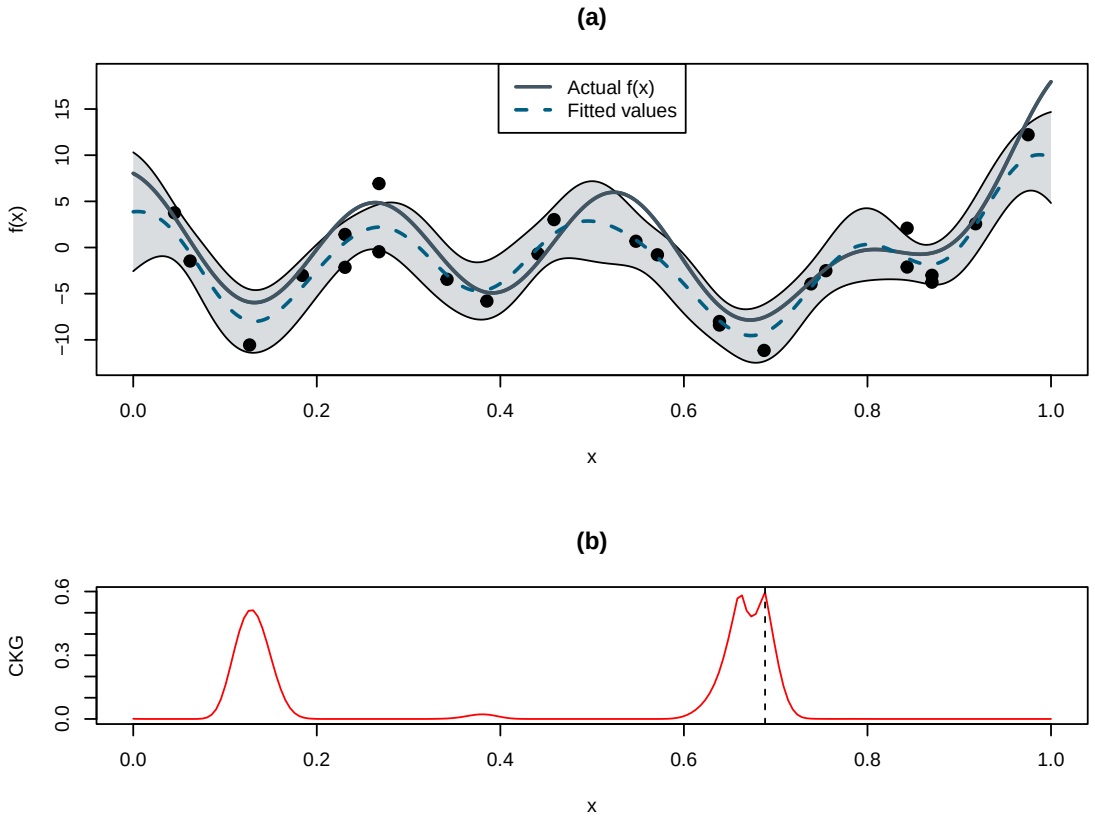

FIGURE 1.10: (a) Example of a noisy GP fitted to $f_C(x)$ (b) Correlated Knowledge Gradient function evaluated

1.2.1.3 Correlated knowledge gradient

The acquisition function for *correlated knowledge gradient* (CKG; Frazier et al. 2009) is the expected improvement to the lowest observed Gaussian process predictive mean at the next step, after adding a point $\mathbf{x}$ to the design. CKG takes into account the change to the Gaussian process predictive mean for all observed points in the design X, rather than for just $\mathbf{x}$. Specifically, the CKG acquisition function is

$$A_{CKG}(\mathbf{x}) = \mathrm{E}\left[\min_{\bar{\mathbf{x}} \in X \cup \mathbf{x}} \mu(\mathbf{x}; \bar{X}, \mathbf{y}) - \min_{\bar{\mathbf{x}} \in X \cup \mathbf{x}} \mu(\bar{\mathbf{x}}; \{X, \mathbf{x}\}, \{\mathbf{y}, y\})\right],$$

where expectation is with respect to $y = f(\mathbf{x})$.

In Figure 1.10, we have evaluated A_{CKG} for all $x \in (0, 1)$ for the computer model in Equation 1.9.

1.2.1.4 Expected quantile improvement (EQI):

Expected quantile improvement (EQI, Forrester 2013) defines improvement in terms of $q(\cdot; X, \mathbf{y})$ rather than the predictive mean $\mu(\cdot; X, \mathbf{y})$. To be more precise, it uses the

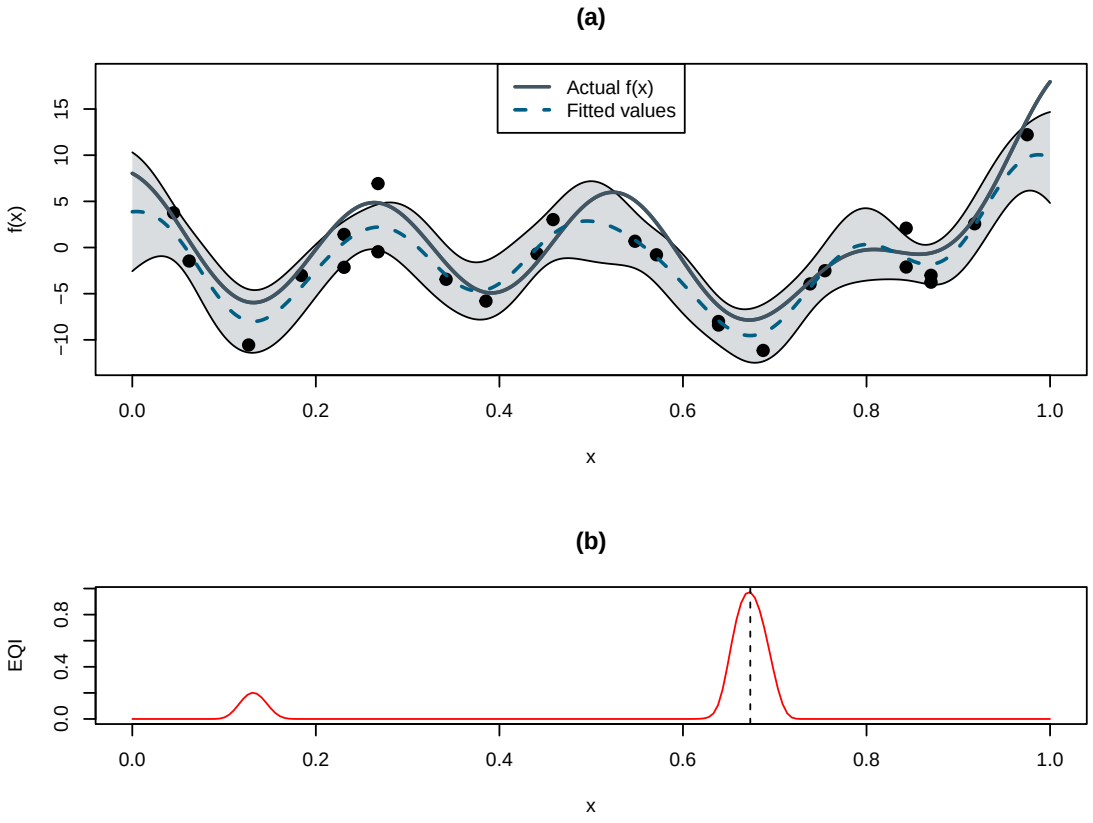

FIGURE 1.11: (a) Example of a noisy GP fitted to $f_C(x)$ (b) Expected Quantile Improvement function evaluated

expected value of $\hat{q}_n(\mathbf{x})$ at the next step to estimate the improvement to the current best solution. The acquisition function is

$$A_{EQI}(\mathbf{x}) = \mathrm{E}\left[\max(q_{min} - q(\mathbf{x}; \{X, \mathbf{x}\}, \{\mathbf{y}, y\}), 0)\right],$$

where $q_{min} = \min_{\mathbf{x} \in X} q(\mathbf{x}; X, \mathbf{y})$ and $\hat{q}(\mathbf{x}; X, \mathbf{y}) = \mu(\mathbf{x}; X, \mathbf{y}) + \Phi^{-1}(\beta)\sqrt{\Sigma(\mathbf{x}; X, \mathbf{y})}$, with $\beta \in [0.5, 1)$, so that expectation is with respect to $y = f(\mathbf{x})$.

By applying the computer model from Equation 1.9, A_{EQI} is evaluated over the interval $x \in (0, 1)$ for $\beta = 0.9$, with results presented in Figure 1.11.

1.2.1.5 Two stage sequential optimisation

Two stage sequential optimisation (TSSO; Quan et al. 2013) explicitly balances exploration and exploitation by addressing them individually with two stages per iteration. In the first stage, a new input is chosen to maximise the modified expected improvement (MEI) acquisition function

$$A_{\mathrm{MEI}}(\mathbf{x}) = \mathrm{E}\left[\max(\mu(\mathbf{x}^{**}_{min}; X, \mathbf{y}) - y_D(\mathbf{x}), 0)\right],$$

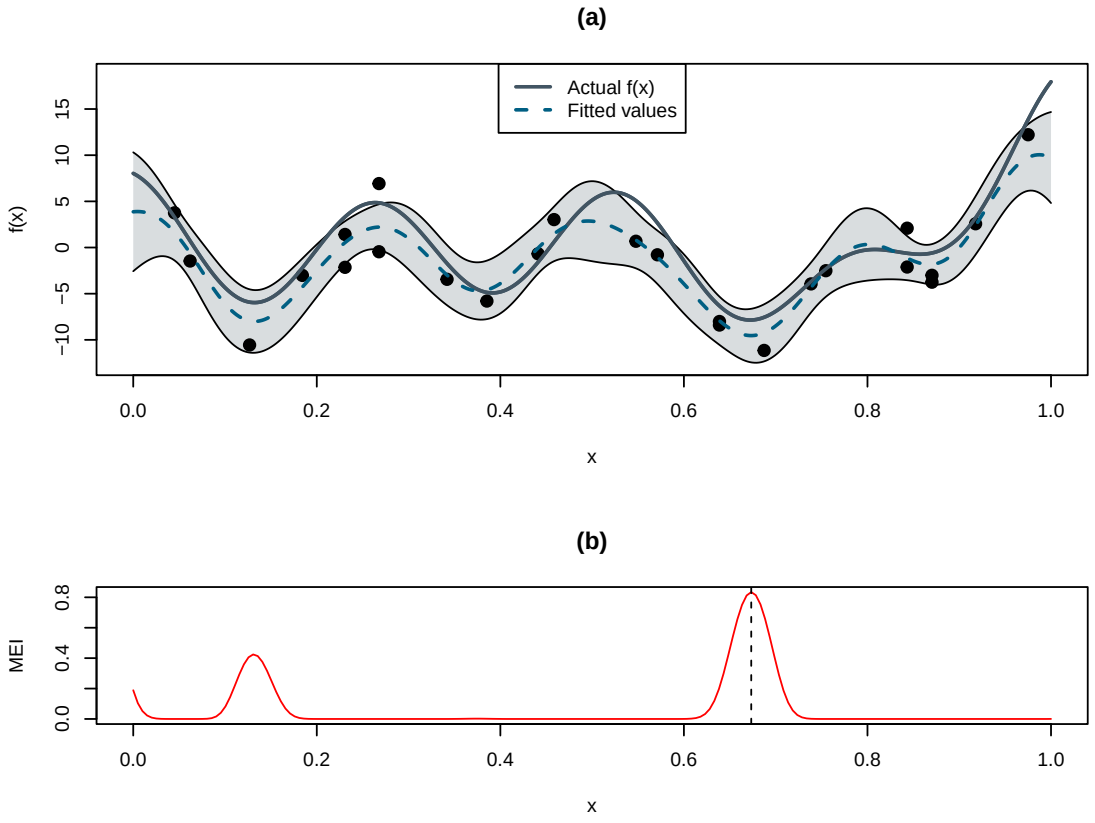

FIGURE 1.12: (a) Example of a noisy GP fitted to $f_C(x)$ (b) Modified Expected Improvement function evaluated

where $\mathbf{x}^{**}_{min} = \arg\min_{\mathbf{x} \in X} f(\mathbf{x})$ and $y_D(\mathbf{x})$ is a random variable with a normal distribution with mean and variance given by (1.4) and (1.5), respectively, but with $\rho = 0$. At the second stage, a predefined amount of computational budget is allocated to the existing inputs in the design. To determine which inputs get the budget, an optimal computing budget allocation (OCBA; Chen et al. 2000) algorithm is used.

Returning to the computer model in Equation 1.9, we evaluate A_{MEI} across the range of $x \in (0, 1)$, as presented in Figure 1.12.

Chapter 2

Entropy Search for Multi-Task Bayesian Optimisation

2.1 Introduction

Bayesian Optimisation (BO) is a powerful framework for the global optimisation of expensive computer models. Traditionally, it is used to locate a single global optimum, either a maximum or a minimum, by building a surrogate model, typically a Gaussian Process (GP), and strategically selecting points to evaluate, using an acquisition function. However, in many scientific and engineering applications, we are interested in both the minimum and the maximum at the same time. These involve cases where we are interested in a system's worst case performance as well as the best case performance. This can be useful in scenario analysis, where we want to understand the full range of possible outcomes, or in stress testing and robustness analysis, where it is important to evaluate how the system behaves under extreme conditions. For example, imagine finding optimal conditions for a chemical reaction model. In that case, we might be interested in both maximising the reaction rate to increase production speed as well as minimising it to prevent overheating and runaway reactions. Current approaches require us to treat those two objectives completely separately, potentially ignoring a lot of information, making the optimisation more inefficient. We address this by introducing a novel extension of Bayesian Optimisation tailored to this dual-objective setting.

The primary contribution of this chapter is a new algorithm that enables us to simultaneously and efficiently find both the minimum and maximum of a given computer model. Our acquisition function uses entropy to minimise uncertainty about the location of both these optima.

We first start this in section 2.3 by defining entropy, which will depend on a joint probability distribution of the minimum and maximum. Since this is not available in closed

form, we demonstrate how we can approximate it by using Monte Carlo integration. This requires us to discretise the sample space. In section 2.3.1 we show how we can achieve this by sampling points from an Expected Improvement function. Points are sampled by using a parallel tempering algorithm as well as particle filtering algorithm. In section 2.3.2 we extend these calculations and show how they can be used to find a point that minimises expected entropy at the next step. This is done by using properties of Gaussian Processes as well as properties of multivariate normal distributions. Finally in section 2.3.3, we compare our algorithm's performance against a baseline algorithm, which is based on Expected Improvement. Our algorithm is shown to achieve better results both in terms of sample efficiency as well as convergence reliability.

2.2 Entropy search

In this chapter, we will be focusing on a subcategory of Bayesian optimisation called entropy-search algorithms (see, for example Hennig and Schuler, 2012; Hernández-Lobato et al., 2014; Wang and Jegelka, 2017). In entropy-search, the acquisition function is the entropy of the posterior probability distribution for the location of the optimum inputs. Thus the idea is to sequentially minimise uncertainty about this distribution.

We propose new entropy-search methodology for multitask Bayesian optimisation, which expands on the existing entropy-search literature. The new methodology allows for multiple objectives to be considered simultaneously. This is achieved by forming a joint posterior probability distribution for minimising and maximising inputs and then minimising the entropy of that distribution.

The principle challenge of implementing this approach is that the joint posterior probability distribution is not available in closed form. That means the acquisition function, used to choose the next inputs, is not available in closed form. Novel methodology is developed to approximate the joint posterior probability distribution, the acquisition function and the inputs that optimise the acquisition function.

2.3 Entropy search for multi-task optimisation

In section 1.2, we introduced some Bayesian optimisation algorithms that can be used for either minimising or maximising a computer model. However, in some situations, we would like to learn about both extremum points (maximum and minimum) simultaneously. For example, when modelling stock prices in financial markets, we would like to know when the minimum and maximum price occurs, so we know when to buy and when to sell. So in other words, we would like to both minimise and maximise the computer model at the same time and find both

$$\mathbf{x}^*_{\min} = \arg \min_{\mathbf{x}\in\mathbb{X}} f(\mathbf{x})$$

and

$$\mathbf{x}^*_{\max} = \arg \max_{\mathbf{x}\in\mathbb{X}} f(\mathbf{x})$$

as accurately as possible.

In our approach, we use a Gaussian process $Y(\mathbf{x})$ as a surrogate for $f(\mathbf{x})$. Since $Y(\mathbf{x})$ is a random variable, $\mathbf{x}^*_{\min}$ and $\mathbf{x}^*_{\max}$ are also random variables. This means that we can rewrite our objective as follows: we want to design our experiment so that the uncertainty about $\mathbf{x}^*_{\min}$ and $\mathbf{x}^*_{\max}$ is minimised.

We can define the probability density function $p_*(\boldsymbol{\chi}_{\min}, \boldsymbol{\chi}_{\max})$ for $\mathbf{x}^*_{\min}$ and $\mathbf{x}^*_{\max}$. The probability that $\mathbf{x}^*_{\min}$ is in a subspace $\Xi_{\min}$ and $\mathbf{x}^*_{\max}$ is in a subspace $\Xi_{\max}$ is defined as follows:

$$P[\mathbf{x}^*_{\min} \in \Xi_{\min}, \mathbf{x}^*_{\max} \in \Xi_{\max}] = \int_{\Xi_{\max}} \int_{\Xi_{\min}} p_*(\boldsymbol{\chi}_{\min}, \boldsymbol{\chi}_{\max}) d\boldsymbol{\chi}_{\min} d\boldsymbol{\chi}_{\max},$$

where $\Xi_{\min}, \Xi_{\max} \subset \mathbb{X}$.

We propose using differential entropy of the probability distribution of $\mathbf{x}^*_{\min}$ and $\mathbf{x}^*_{\max}$ as a measure of uncertainty about $\mathbf{x}^*_{\min}$ and $\mathbf{x}^*_{\max}$. For a continuous distribution $p_*(\boldsymbol{\chi}_{\min}, \boldsymbol{\chi}_{\max})$, its entropy is

$$\begin{aligned} H(p_*(\boldsymbol{\chi}_{\min}, \boldsymbol{\chi}_{\max})) &= E[-\log(p_*(\boldsymbol{\chi}_{\min}, \boldsymbol{\chi}_{\max}))] \\ &= -\int_{\mathbb{X}} \int_{\mathbb{X}} p_*(\boldsymbol{\chi}_{\min}, \boldsymbol{\chi}_{\max}) \log(p_*(\boldsymbol{\chi}_{\min}, \boldsymbol{\chi}_{\max})) d\boldsymbol{\chi}_{\min} d\boldsymbol{\chi}_{\max} \end{aligned}$$

(Cover and Thomas (1991)).

Unfortunately, $p_*(\boldsymbol{\chi}_{\min}, \boldsymbol{\chi}_{\max})$ is not available in closed form and has to be approximated using numerical methods. For simplicity, we first note that it is possible to represent a continuous distribution by using a discrete set of points $\zeta^F = (\boldsymbol{\chi}_1, ..., \boldsymbol{\chi}_F)$ (Kohavi and Sahami (1996)). This means that $p_*(\boldsymbol{\chi}_{\min}, \boldsymbol{\chi}_{\max})$ becomes a discrete probability mass function and we can redefine it as

$$p_*(\boldsymbol{\chi}_{\min}, \boldsymbol{\chi}_{\max}) = P\left[\mathbf{x}^*_{\min} = \boldsymbol{\chi}_{\min}, \mathbf{x}^*_{\max} = \boldsymbol{\chi}_{\max}\right] \tag{2.1}$$

and entropy is now

$$H(p_*(\boldsymbol{\chi}_{\min}, \boldsymbol{\chi}_{\max})) = -\sum_{i=1}^{F} \sum_{j=1}^{F} p_*(\boldsymbol{\chi}_i, \boldsymbol{\chi}_j) \log(p_*(\boldsymbol{\chi}_i, \boldsymbol{\chi}_j)). \tag{2.2}$$

We propose using Monte Carlo (MC) integration to approximate the distribution in Equation 2.1. Monte Carlo integration is robust, flexible, unbiased and is also asymptotically exact. Unfortunately, it can also be computationally expensive. We perform MC integration by sampling G functions from the Gaussian process and choosing the point with the lowest and the highest function value at each step (see Algorithm 2).

Algorithm 2: Estimating $p_*(\boldsymbol{\chi}_{\min}, \boldsymbol{\chi}_{\max})$ by using GP mean and variance

Require: A set of F points $\zeta^F = (\boldsymbol{\chi}_1, ..., \boldsymbol{\chi}_F)$, a GP mean vector $\hat{\boldsymbol{m}}_n(\zeta^F)$ and a variance-covariance matrix $\hat{\Sigma}_n(\zeta^F)$ as defined in Equations 1.4 and 1.5, G as the number of iterations in the MC integration and a zero-matrix L with size $F \times F$.

1 **for** $g = 1, ..., G$ **do**
2 Sample $\mathbf{Y} = (\mathrm{Y}_1, ..., \mathrm{Y}_F) \sim N(\hat{\boldsymbol{m}}_n(\zeta^F), \hat{\Sigma}_n(\zeta^F))$;
3 Let $a = \arg\min_{1 \le i \le F}(\mathrm{Y}_i)$ be the index of the minimum sampled value in $\mathbf{Y}$;
4 Let $b = \arg\max_{1 \le i \le F}(\mathrm{Y}_i)$ be the index of the maximum sampled value in $\mathbf{Y}$;
5 $L_{a,b} = L_{a,b} + 1$;
6 **end**

Output : A discrete probability mass function with $p_*(\boldsymbol{\chi}_i, \boldsymbol{\chi}_j) = L_{i,j}/G$ for $i, j = 1, ..., F$

We can also calculate the marginal distribution of $p_*(\boldsymbol{\chi}_{\min})$ from the output of Algorithm 2 as follows:

$$p_*(\boldsymbol{\chi}_{\min}) = \sum_{j=1}^{F} p_*(\boldsymbol{\chi}_{\min}, \boldsymbol{\chi}_j) \tag{2.3}$$

and similarly for $p_*(\boldsymbol{\chi}_{\max})$ we get

$$p_*(\boldsymbol{\chi}_{\max}) = \sum_{i=1}^{F} p_*(\boldsymbol{\chi}_i, \boldsymbol{\chi}_{\max}). \tag{2.4}$$

As a simple example, assume that we have a deterministic computer model $f_D(x) = (3.6x - 1.4)\sin(22x)$. We can create an initial design of size $n = 10$ and fit a GP to this data, which can be seen in Figure 2.1 (a).

We now choose a number of discrete points ζ^F from the design space. For this example, we simply used $F = 200$ equally spaced points between 0 and 1. We calculate the mean and variance-covariance matrices $\boldsymbol{m}_n(\zeta^F)$ and $\Sigma_n(\zeta^F)$ for these points. We then use these to draw a number of random samples from the fitted Gaussian process, which can be seen in Figure 2.1 (b).

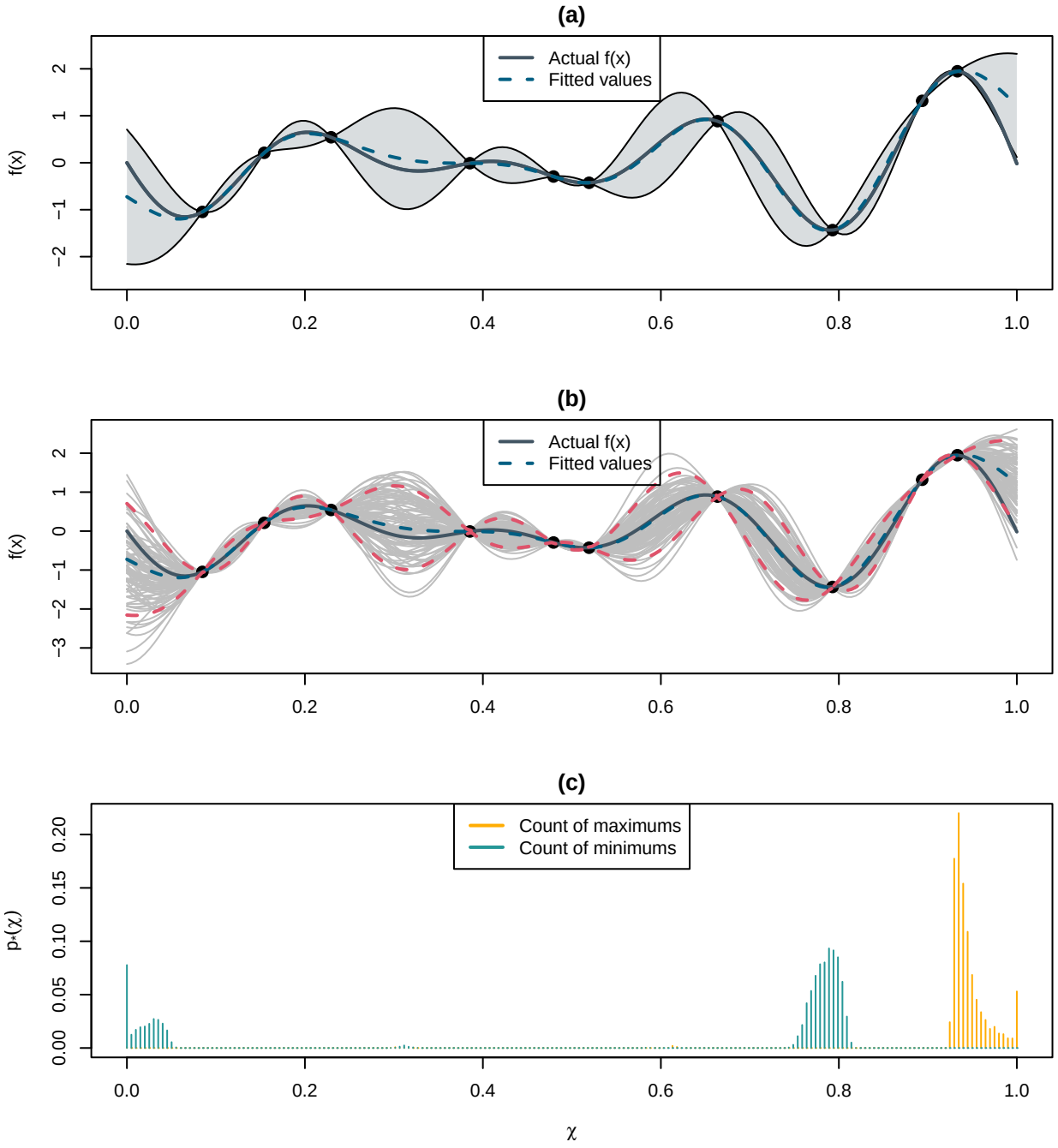

FIGURE 2.1: (a) Example of a Gaussian process. (b) Random samples drawn from a Gaussian process. (c) Counting how many times a point has either the maximum or the minimum sampled value

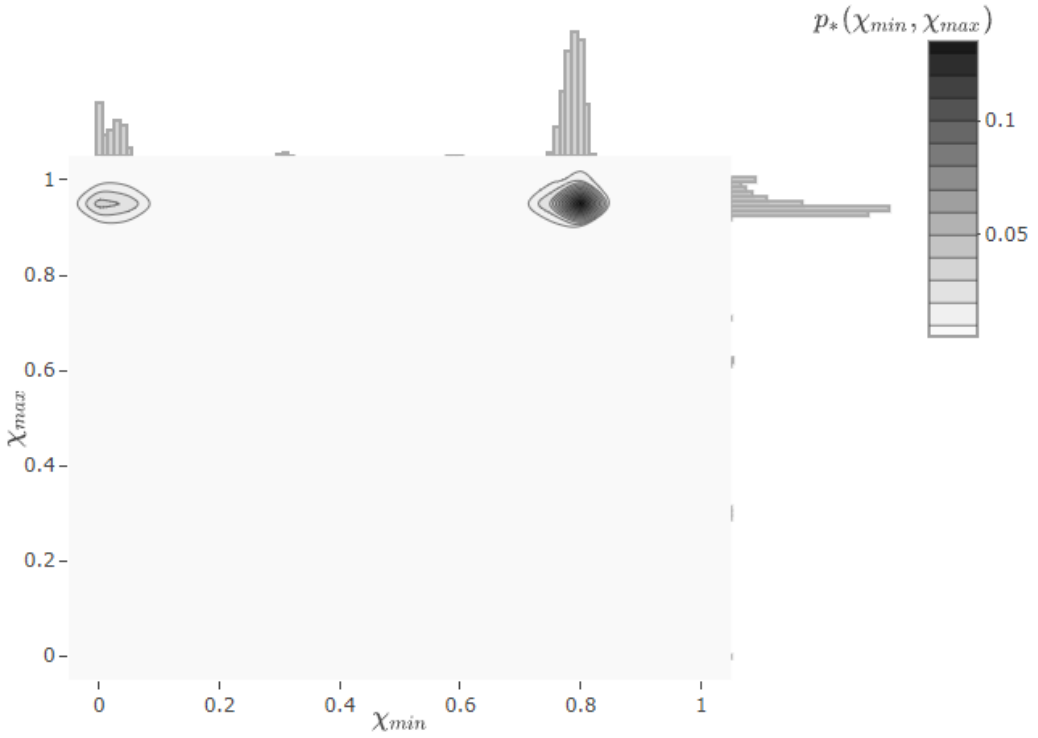

FIGURE 2.2: An approximation of $p_*(\chi_{\min}, \chi_{\max})$

Recording how many times a certain point from $\boldsymbol{\chi}_1, ..., \boldsymbol{\chi}_F$ has the minimum and the maximum sampled value (seen in Figure 2.1 (c)), we can estimate $p_*(\boldsymbol{\chi}_{\min}, \boldsymbol{\chi}_{\max})$. This can be seen in Figure 2.2. Note that in Figure Figure 2.1 (c), we see large weights on both boundaries, potentially biasing the approximation. However, in real world scenarios, boundaries could be extended or the computer model could be rescaled to avoid scenarios where interesting regions are located at the boundaries.

In Figure 2.2, darker colour means higher values of $p_*(\chi_{\min}, \chi_{\max})$. On the x-axis, we see $p_*(\chi_{\min})$ and on the y-axis we see $p_*(\chi_{\max})$. Most of the weight in $p_*(\chi_{\max})$ is around $\chi \approx 0.9$, while the weight in $p_*(\chi_{\min})$ is divided between $\chi \approx 0.1$, and $\chi \approx 0.8$.

2.3.1 Discretising the sample space

From Figure 2.2, we notice that $p_*(\chi_{\min}, \chi_{\max})$ is a distribution with clearly defined modes and $p_*(\chi_{\min}, \chi_{\max}) = 0$ for most of $\boldsymbol{\chi}_{\min}, \boldsymbol{\chi}_{\max} \in \zeta^F$. In the case of $p_*(\chi_{\min}, \chi_{\max}) = 0$, it is often a convention to define $p_*(\chi_{\min}, \chi_{\max}) \log(p_*(\chi_{\min}, \chi_{\max})) = 0$ due to the fact that $\lim_{p \to 0^+} p \log p = 0$ (Cover and Thomas (1991)). This means that ideally, we would like to approximate $p_*()$ only for $\boldsymbol{\chi}_{\min}$ and $\boldsymbol{\chi}_{\max}$, where $p_*(\chi_{\min}, \chi_{\max}) > 0$, since $p_*(\chi_{\min}, \chi_{\max}) = 0$ does not have any effect on entropy.

Therefore, we would like our discrete set of points ζ^F to be in areas with high values of p_*. Based on this requirement and looking at Equations 2.3 and 2.4, we can also say that

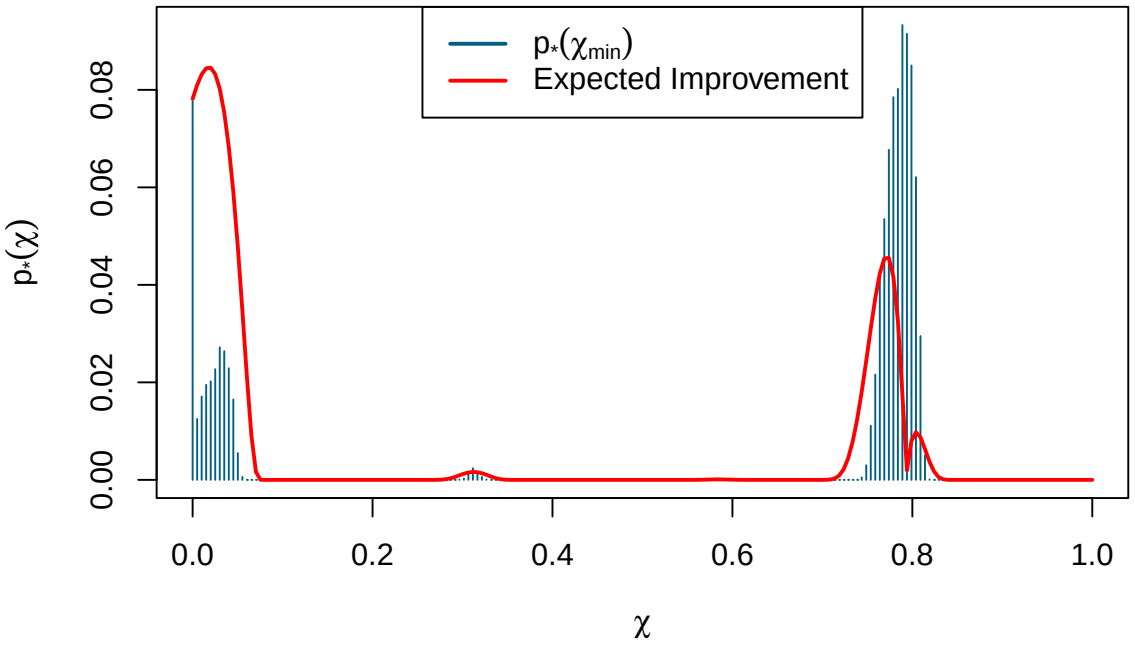

FIGURE 2.3: $p_*(\chi_{\min})$ plotted against expected improvement

we would like our discrete set of points ζ^F to be in areas where either $p_*(\boldsymbol{\chi}_{\min}) > 0$ or $p_*(\boldsymbol{\chi}_{\max}) > 0$. In single-task optimisation, researchers have used a proposal distribution π, from which they sample F points. This distribution π should have high values in areas where $p_*(\boldsymbol{\chi}_{\min})$ (or $p_*(\boldsymbol{\chi}_{\max})$, depending on the problem) is also high. It has been shown that choosing π to be a distribution that is induced by Expected Improvement provides good results (Hennig and Schuler (2012)). That is due to the fact that Expected Improvement often has a comparable shape to $p_*(\boldsymbol{\chi}_{\min}) > 0$ and $p_*(\boldsymbol{\chi}_{\max}) > 0$. In Figures 2.3 and 2.4 we have compared $EI_{\min}$ and $EI_{\max}$ to the distributions of $p_*(\boldsymbol{\chi}_{\min}) > 0$ and $p_*(\boldsymbol{\chi}_{\max}) > 0$ from the example in Figure 2.1 (c). We can see that they both have comparable shapes in terms of both having values greater than zero in similar regions of the input space. Therefore, we propose to sample $\frac{F}{2}$ points from $\pi_{\min}$ and $\frac{F}{2}$ points from $\pi_{\max}$, where $\pi_{\min}$ and $\pi_{\max}$ are induced by $EI_{\min}$ and $EI_{\max}$ respectively.

We propose to use the following distributions:

$$\pi_{\min}(\boldsymbol{\chi}) \propto EI_{\min}(\boldsymbol{\chi}) p_{\min}(\boldsymbol{\chi}) \tag{2.5}$$

$$\pi_{\max}(\boldsymbol{\chi}) \propto EI_{\max}(\boldsymbol{\chi}) p_{\max}(\boldsymbol{\chi}), \tag{2.6}$$

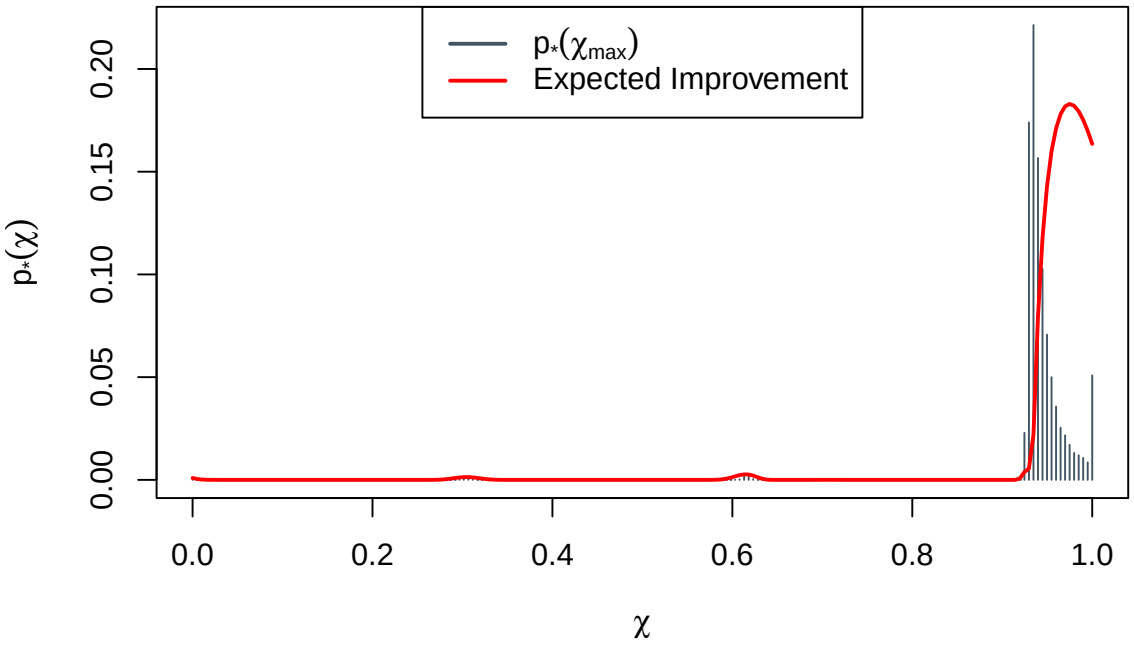

FIGURE 2.4: $p_*(\chi_{\max})$ plotted against expected improvement

where $EI_{\min}(\boldsymbol{\chi})$ and $EI_{\max}(\boldsymbol{\chi})$ are the Expected Improvement functions (Equations 1.7 and 1.8) and $p_{\min}(\boldsymbol{\chi})$ and $p_{\max}(\boldsymbol{\chi})$ are prior distributions. The prior distributions represent our prior knowledge about the location of the minimum or the maximum respectively. Often we have no prior knowledge, so we can choose the prior distributions to be uniform over the entire design region. This is what we will be doing throughout the rest of this chapter.

In Equation 1.7, $y_{\min}$ is defined as $y_{\min} = \min_{i\in(1,\dots,n)} y_i$. However, this is only appropriate in the noiseless setting, where $f(x_i) = y_i$. Using this approach in the stochastic modelling setting can lead to a scenario where we observe just a single very noisy observation that is unusually low and leads to severely underestimating $y_{\min}$, which also skews the shape of the EI function. A logical choice would be to define $y_{\min} = \min_{x\in X_n} \hat{m}_n(\mathbf{x})$. This is also known as expected improvement with "plug-in" (Picheny et al. (2013)). When using EI as a decision criteria for the next evaluation in the computer model, this approximation might not always be the best choice as it does not take into account the noise from observing any new points. However, our goal is to use EI to find areas with high values of p_* and for that purpose, we do not care about the noise of future observations, which means that this approximation works well. Similarly we define $y_{\max} = \max_{\mathbf{x}\in X_n} \hat{m}_n(\mathbf{x})$.

Our aim is to therefore generate a sample from $\pi_{\min}(\boldsymbol{\chi})$ and $\pi_{\max}(\boldsymbol{\chi})$. We note that EI function (and therefore also $\pi_{\min}(\boldsymbol{\chi})$ and $\pi_{\max}(\boldsymbol{\chi})$) is often multi-modal, with different modes being far from each other. This means that it might be difficult for traditional MCMC algorithms to fully explore the entirety of the sample space, as it's difficult to "jump" from one mode to another, especially in higher dimensions (Neal (1993)).

To demonstrate this, we have implemented a simple Metropolis-Hastings algorithm (Metropolis et al. (1953); Hastings (1970)), which we use to draw random samples from a target distribution. This algorithm is defined in Algorithm 3.

Algorithm 3: Simple random walk Metropolis-Hastings algorithm

Require: Function $f(x)$, which is proportional to the target distribution, starting points x_0, the variance of the random walk σ^2, number of samples drawn F

Output : A vector $\mathbf{x} = (x_1, ..., x_F)$ of length F, which is a sample from $f(x)$

1 **for** $i = 1, ..., F$ **do**
2 $\quad x_i \sim N(x_{i-1}, \sigma^2)$;
3 $\quad$ With probability $\min\left\{\frac{f(x_i)}{f(x_{i-1})}, 1\right\}$ accept proposal, otherwise $x_i = x_{i-1}$
4 **end**

As a simple example, consider the function

$$f_E(x) = 0.4\exp(-(0.6x)^2)(1 + \sin(2(x-1))).$$

This function is shown in Figure 2.5 as the red line. This distribution is bimodal and direct sampling from it is impossible. We therefore run Algorithm 3 to draw 2000 random samples from it. We choose $\sigma^2 = 4$. The sample is shown in Figure 2.5. This sample exhibits all the characteristics of a "good" sample from a Metropolis-Hastings algorithm. The MCMC chain manages to explore both modes and the acceptance ratio is 25.9%, which is very close to the ideal acceptance ratio of 23.4% demonstrated by Gelman et al. (1997).

Unfortunately, sampling from EI induced distributions can be much more difficult as we will now demonstrate. Let us assume our target distribution is proportional to the expected improvement function shown in Figure 2.3. This also has multiple modes, just like $f_E(x)$. However, the modes are much further apart and more narrow, making it difficult to sample from it. We have demonstrated this in Figure 2.6.

In Figure 2.6, we have run Algorithm 3 for four different values of σ^2. In the first instance, we have $\sigma^2 = 0.5^2$. For this value, we do manage to jump between the two modes, but the vast majority of proposed values are rejected. The acceptance ratio in that example is only 3.8%, which is far lower than we would like to see. In the second example, we have $\sigma^2 = 0.3^2$. The acceptance ratio is now slightly better at 8.7%, but the jumps between the two modes are now infrequent. In the third example with

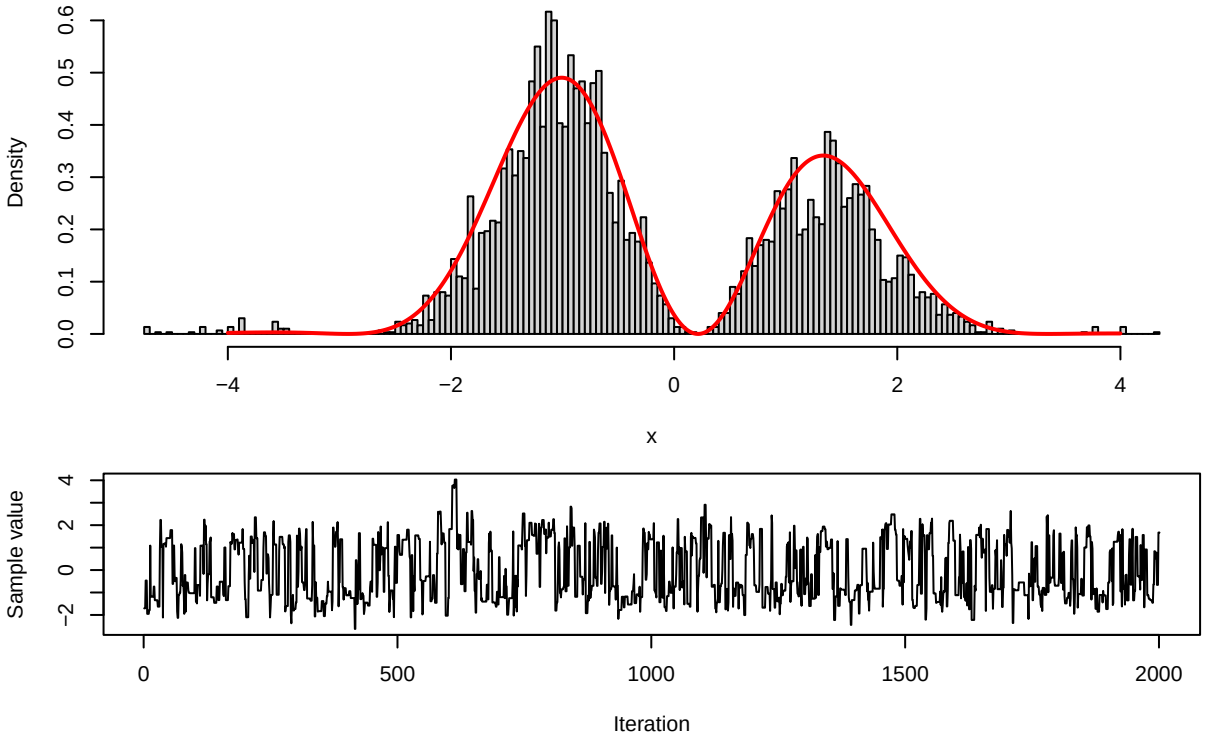

FIGURE 2.5: Ideal MCMC sample

$\sigma^2 = 0.1^2$, our acceptance ratio is 22.2%, which is very close to the ideal value of 23.4%. Unfortunately, the chain gets stuck in the left mode and is unable to reach the other one due to the fact that our σ^2 is now too low. In the final chain, we choose $\sigma^2 = 0.05^2$. Similarly to the previous chain, it starts off at around $x \approx 0.3$. While the previous chain managed to get out of that point fairly quickly, this final chain does not reach one of the other two modes at all.

This demonstrates how Metropolis Hastings does not work well for our problem. To address this, we use parallel tempering (Swendsen and Wang (1986); Geyer et al. (1991)). In a parallel tempering algorithm, we simultaneously sample from multiple versions of the distribution.

$$\pi_{T_j}(\boldsymbol{\chi}) \propto EI(\boldsymbol{\chi})^{\frac{1}{T_j}} p(\boldsymbol{\chi}) \text{ for } 1 = T_1 < T_2 < .... < T_D,$$

where T_j is a temperature associated with the distribution. The higher the temperature, the flatter the distribution and the easier it is to jump from one mode to another. In Figure 2.7, we have plotted the same function for 4 different temperatures. The function is the $EI_{\min}$ function from Figure 2.3 and the four temperatures are $T_1 = 1$, $T_2 = 2$, $T_3 = 10$ and $T_4 = 50$. As we see, with the higher temperatures, the function becomes more flat and it becomes much easier to sample from.

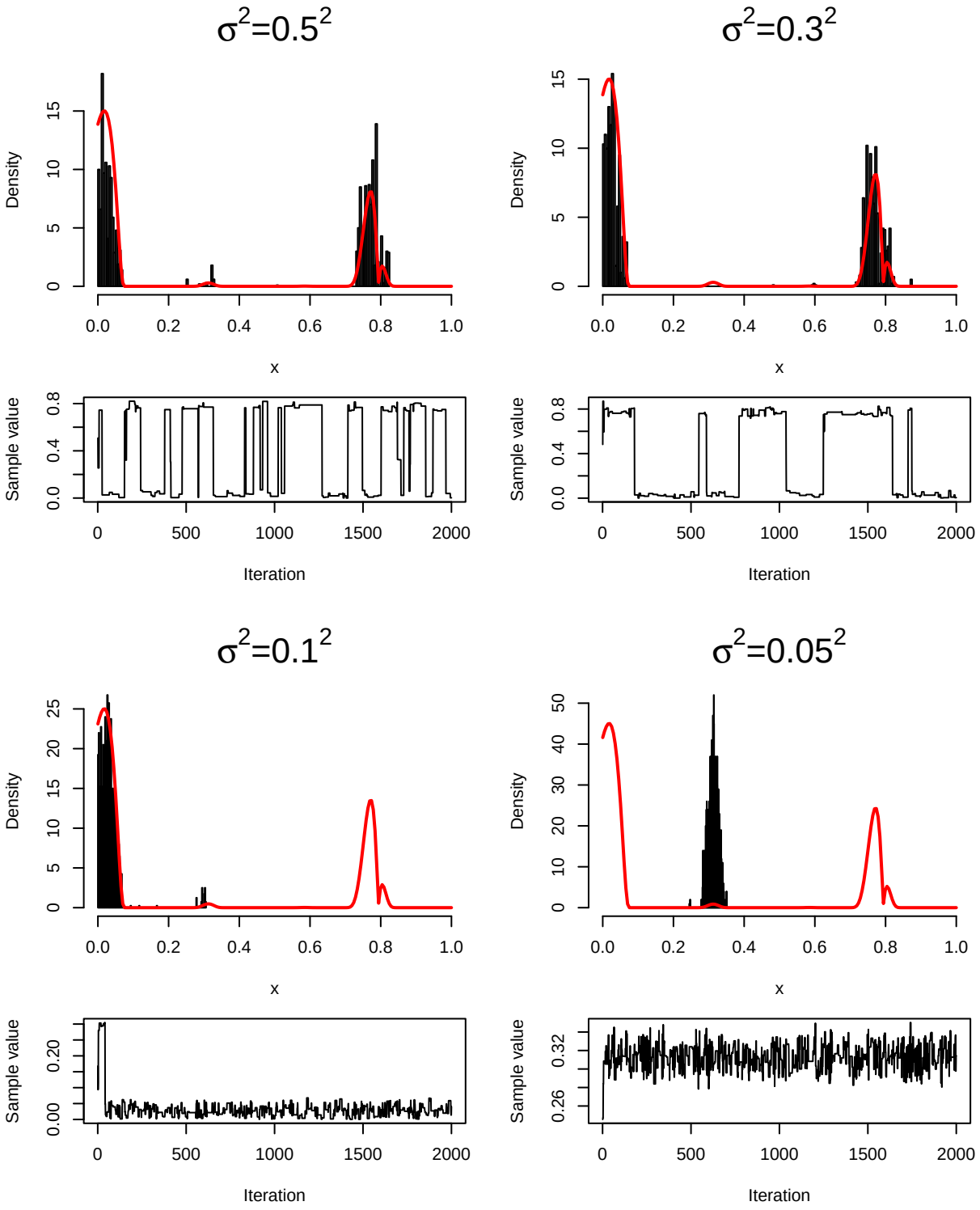

FIGURE 2.6: Algorithm 3 applied to Expected Improvement in Figure 2.3

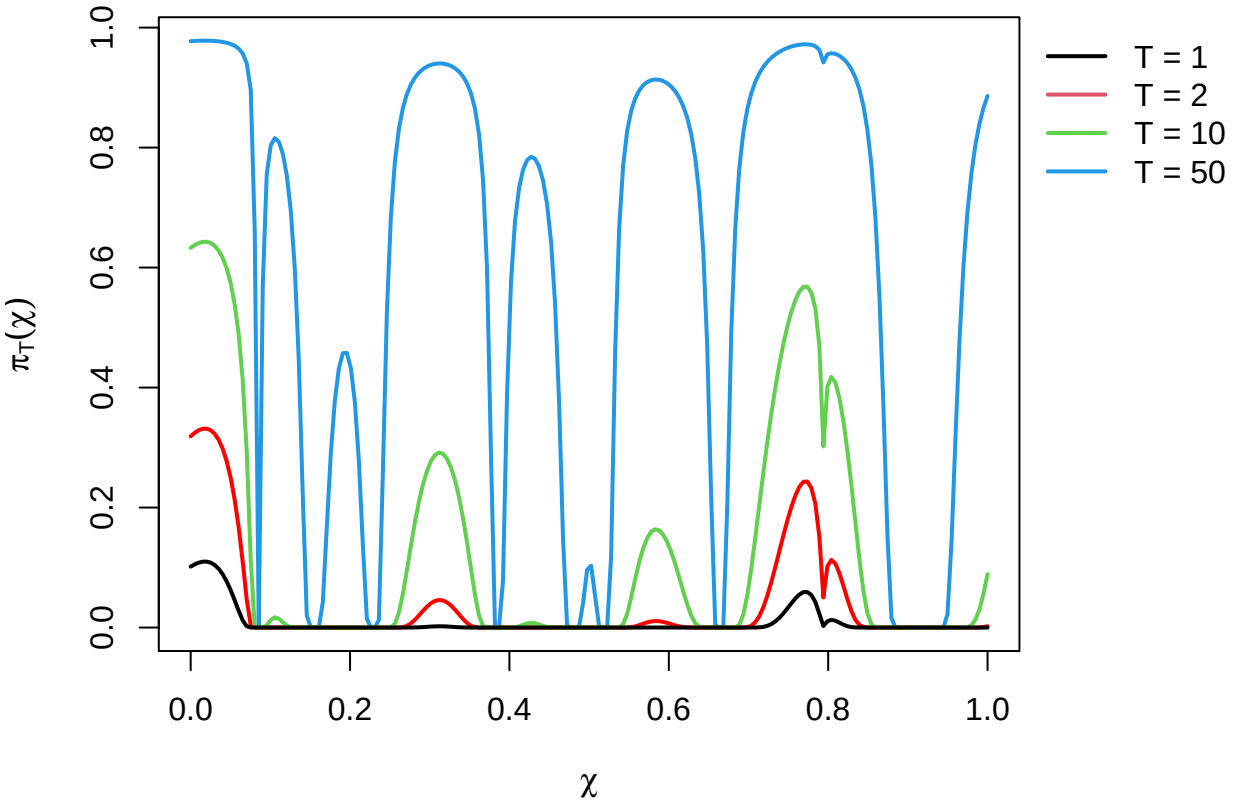

FIGURE 2.7: $EI_{\min}$ function from Figure 2.3 plotted at different temperatures. For $T = 1$, the function shows $(EI_{\min})^{\frac{1}{1}}$, for $T = 2$, the function shows $(EI_{\min})^{\frac{1}{2}}$, etc.

For the parallel tempering algorithm to work, temperatures must be chosen carefully. What a good set of temperatures looks like, depends on the shape of the Expected Improvement function. This shape, however, is unknown before running the algorithm, which makes choosing a correct set of temperatures a very difficult task. We can make this task simpler, by allowing the algorithm to dynamically alter the temperatures at the start of the MCMC chain. To do this, we're using an algorithm introduced by Vousden et al. (2016).

It has been suggested (Sugita and Okamoto (1999); Kofke (2002); Earl and Deem (2005)), that temperatures should be chosen in such a manner that acceptance ratios $A_{i,j}$ (i.e. the proportion of swaps accepted between chains i and j) should be uniform for all pairs of adjacent chains.

This can be achieved by first noting that as the difference between two temperatures T_i and T_j decreases, the expected value of the acceptance ratio $E[A_{i,j}]$ increases and as the difference between two temperatures T_i and T_j increases, the expected value of the acceptance ratio $E[A_{i,j}]$ decreases. This suggests that when swaps between two chains are not accepted often enough, one should decrease the difference between the

two temperatures and vice versa. This leads us to the parallel tempering algorithm seen in Algorithm 4.

Algorithm 4: Parallel tempering algorithm with dynamic temperature selection

Require: D temperatures $1 = T_1 < T_2 < ... < T_D$ and distributions $\pi_{T_1}(\boldsymbol{\chi}), ..., \pi_{T_D}(\boldsymbol{\chi})$, starting points $\chi_{EI} = (\boldsymbol{\chi}_{1_{EI}}, ..., \boldsymbol{\chi}_{D_{EI}})$, the variance of the random walk for each of the chains $\boldsymbol{\sigma}^2 = (\sigma_1^2, ..., \sigma_D^2)$, tuning parameters for dynamic temperature selection v and t_0

Define : $A_j(i)$ is the proportion of swaps accepted between chain j and $j+1$ in the first i iterations

Output : An $F' \times d$ matrix $\boldsymbol{\Psi}$, which is a sample from $\pi_{T_1}(\boldsymbol{\chi}) = \pi(\boldsymbol{\chi})$

Let $\chi_{EI}^{old} = \chi_{EI}$;
Let $\boldsymbol{y}_{EI}^{old} = (y_{1_{EI}}^{old}, ..., y_{1_{EI}}^{old})$ where $y_{i_{EI}}^{old} = \pi_{T_i}(\boldsymbol{\chi}_i)$ for $i = 1, ..., D$;
for $i = 1, ..., F'$ **do**
  for $j = 1, ..., D$ **do**
    $\boldsymbol{\chi}_{j_{EI}} \sim N(\boldsymbol{\chi}_{j_{EI}}^{old}, \sigma_j^2)$ and $y_{j_{EI}} = \pi_{T_j}(\boldsymbol{\chi}_{j_{EI}})$;
    With probability $min\left\{\frac{y_{j_{EI}}}{y_{j_{EI}}^{old}}, 1\right\}$ accept proposal, otherwise $\boldsymbol{\chi}_{j_{EI}} = \boldsymbol{\chi}_{j_{EI}}^{old}$
  end
  Sample r uniformly from $1, 2, ..., D-1$;
  With probability

$$\min\left\{\frac{\pi_{T_r}(\boldsymbol{\chi}_{r+1_{EI}})\pi_{T_{r+1}}(\boldsymbol{\chi}_{r_{EI}})}{\pi_{T_r}(\boldsymbol{\chi}_{r_{EI}})\pi_{T_{r+1}}(\boldsymbol{\chi}_{r+1_{EI}})}, 1\right\},$$

  swap $\boldsymbol{\chi}_{r_{EI}}$ and $\boldsymbol{\chi}_{r+1_{EI}}$;
  Let $\boldsymbol{\Psi}_i = \boldsymbol{\chi}_{1_{EI}}$, $\boldsymbol{\chi}_{EI}^{old} = \boldsymbol{\chi}_{EI}$ and $\boldsymbol{y}_{EI}^{old} = (\pi_{T_1}(\boldsymbol{\chi}_{1_{EI}}), , ..., \pi_{T_D}(\boldsymbol{\chi}_{D_{EI}}))$;
  Let $t = \max(2, r)$;
  Calculate $k(i) = \frac{1}{v}\frac{t_0}{i+t_0}$;
  Let $T_t = \exp\left(k(i)(A_{t-1}(i) - A_t(i)) + \log(T_t - T_{t-1})\right) + T_{t-1}$;
end

For optimal performance of the parallel tempering algorithm, one must specify the number of chains running in parallel. It has been argued (Falcioni and Deem (1999)), that for efficient sampling performance, the number of chains D, should scale with $\sqrt{d}$, which is the dimensionality of the computer model. In practice, we observed good performance by letting $D = \lfloor 4\sqrt{d} \rfloor$, where $\lfloor . \rfloor$ is the floor function.

In parallel tempering, we also need to specify the initial temperatures $\{T_1, ..., T_D\}$ and the variance of the random walk for each chain $\boldsymbol{\sigma}^2$. To fully explore the entire sample space, we wish for the largest temperature T_D to be large enough, so that the acceptance ratio for $\pi_{T_D}(\boldsymbol{\chi})$ is approximately 100% (Vousden et al. (2016)). This means that $\pi_{T_D}(\boldsymbol{\chi})$ should resemble the prior distribution $p(\boldsymbol{\chi})$. It is difficult to find that value a priori, but we have found in our experiments that setting $T_D = 100$ yields good results in most cases. After that, we simply set the temperatures in a geometric sequence between $T_1 = 1$ and $T_D = 100$, so that $\frac{T_j}{T_{j-1}}$ is constant for all $j \in \{2, ..., D\}$.

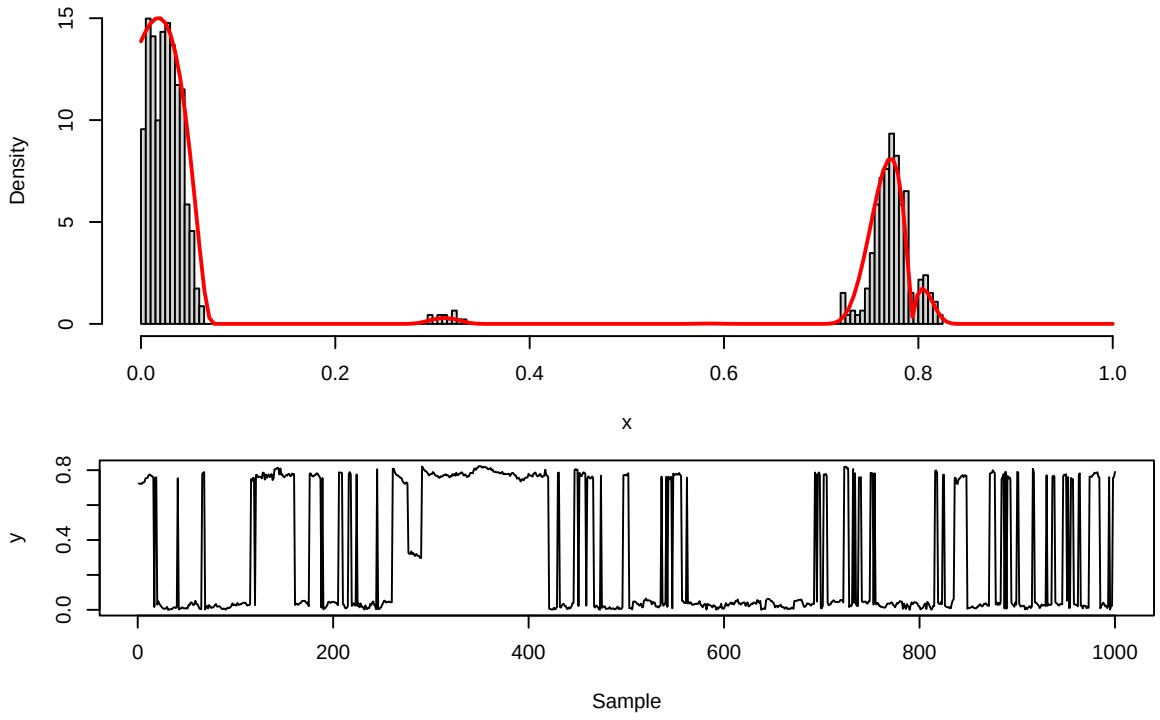

FIGURE 2.8: Algorithm 4 applied to Expected Improvement in Figure 2.3

To choose the values for $\boldsymbol{\sigma}$, we note that for any $j \in \{2, ..., D\}$, $\pi_{T_j}(\boldsymbol{\chi})$ is easier to sample from than $\pi_{T_{j-1}}(\boldsymbol{\chi})$, which means that we want $\pi_{T_j}(\boldsymbol{\chi})$ to explore and jump around more than $\pi_{T_{j-1}}(\boldsymbol{\chi})$. This indicates that we would like $\sigma_{j-1} < \sigma_j$. In our experiments, we set $\sigma_1 = 0.025$, $\sigma_D = 0.25$ and spaced all other values in a geometric sequence between them, so that $\frac{\sigma_j}{\sigma_{j-1}}$ is constant for all $j \in \{2, ..., D\}$.

In line 13 of Algorithm 4, the new temperature will depend on $k(i)$, which depends on parameters v and t_0. In order for the adaptive MCMC sampler to converge to the target distribution, we must have $\lim_{i\to\infty} k(i) = 0$ (Roberts and Rosenthal (2007)). And indeed, if $i \gg t_0$, $k(i) \approx 0$. In practice, we must therefore choose t_0 and v appropriately, so that the algorithm is stable at the beginning, has enough time to change the temperatures and also converges to the target distribution. The suggested values are $v = \frac{100}{D}$ and $t_0 = \frac{1000}{D}$, where D is the number of chains running in parallel (Vousden et al. (2016)).

Now that we have fully defined our parallel tempering algorithm, we can implement it and test it for the same test problem as previously. We use the default parameters to sample from the EI function in Figure 2.3. The results can be seen in Figure 2.8.

In Figure 2.8, we have only run the algorithm for 1000 iterations compared to 2000 in Figure 2.6 to demonstrate its superior performance. As we see, the algorithm does a

great job at sampling from both modes accurately and achieves accurate results even with small number of samples.

At every step in our sequential design algorithm, we sample from a different $\pi(\boldsymbol{\chi})$ function. However, in practice, at any step, EI function (and therefore also $\pi(\boldsymbol{\chi})$) often looks somewhat similar to the EI function at the previous step. This means that we do not need to run the entire parallel tempering algorithm again, but can instead use the previously sampled points to sample new points. We use particle filtering algorithm to do that (Del Moral (1996); Liu and Chen (1998)). This algorithm can be seen in Algorithm 5.

Algorithm 5: Particle filtering algorithm

Require: An old sample $\boldsymbol{\Psi}$ of length F' and a new proposal distribution π

1 Calculate a weight vector $\boldsymbol{w}$, where $\boldsymbol{w}_i = \frac{\pi(\boldsymbol{\Psi}_i)}{\sum_{i=1}^{F} \pi(\boldsymbol{\Psi}_i)}$;

2 Get a new sample $\boldsymbol{\Psi}'$ of size F' by sampling with replacement from $\boldsymbol{\Psi}$, with probability $\boldsymbol{w}$;

With these two algorithms we can then sample $\frac{F'}{2}$ points from $\pi_{\min}(\boldsymbol{\chi})$ and $\frac{F'}{2}$ points from $\pi_{\max}(\boldsymbol{\chi})$, which means we can discretise our sample space into F' points. However, we need F points for approximating $p_*(\chi_{\min}, \chi_{\max})$ and $F' > F$. We can do this, by taking the unique values from our MCMC samples and uniformly sampling F values without replacement from those values.

Now that we have our set of points $\zeta^F = (\boldsymbol{\chi}_1, ..., \boldsymbol{\chi}_F)$, we can reliably approximate $p_*(\chi_i, \chi_j)$ for $i, j = 1, ..., F$ by using Algorithm 2. We can then calculate entropy for this distribution by using Equation 2.2.

Since entropy is a measure of uncertainty and we wish to minimise uncertainty, we want to minimise entropy. In other words, we wish to choose the next point in our sequential design, to minimise the entropy at the next step. However, so far we've only covered how to approximate $H(p_*(\chi_{\min}, \chi_{\max}))$ for the current step.

2.3.2 Estimating entropy loss for future observations

As we have seen, we can calculate entropy by using Algorithm 2, if we have the mean vector as well as the variance-covariance matrix from the fitted Gaussian process. Therefore, to calculate the expected entropy at the next step, after including a point $\boldsymbol{\chi}_*$, we need to find the value of the mean and variance-covariance matrices at the next step. Fortunately, Gaussian processes have convenient equations for updating the mean and variance-covariance matrices after adding in an extra observation. After observing y^{n+1} at $\mathbf{x}_{n+1}$, the mean and variance of the Gaussian process become:

$$\hat{m}_{n+1}(\mathbf{x}) = \hat{m}_n(\mathbf{x}) + \hat{\Sigma}_0(\mathbf{x}, \mathbf{x}_{n+1})\hat{\Sigma}_0^{-1}(\mathbf{x}_{n+1}, \mathbf{x}_{n+1})(y^{n+1} - \hat{m}_n(\mathbf{x}_{n+1})) \tag{2.7}$$

$$\hat{\Sigma}_{n+1}(\mathbf{x}) = \hat{\Sigma}_n(\mathbf{x}) + \hat{\sigma}^2(\hat{\Sigma}_0(\mathbf{x}, \mathbf{x}_{n+1})\hat{\Sigma}_0^{-1}(\mathbf{x}_{n+1}, \mathbf{x}_{n+1})\hat{\Sigma}_0(\mathbf{x}_{n+1}, \mathbf{x})), \tag{2.8}$$

where $\hat{\Sigma}_0(X_1, X_2) = \hat{k}_n(X_1, X_2) - \hat{k}_n(X_1, X_n)\hat{K}_n^{-1}(X_n, X_n)\hat{k}_n(X_n, X_2)$ and X_n is the current design.

We notice that while $\hat{\Sigma}_{n+1}(\mathbf{x})$ is deterministic, $\hat{m}_{n+1}(\mathbf{x})$ is not and depends on the unknown value y^{n+1}. Therefore, to account for all of the uncertainty about $\hat{m}_{n+1}(\mathbf{x})$, we sample a different value of y^{n+1} from $N(\hat{m}_n(\mathbf{x}_{n+1}), \hat{\Sigma}_n(\mathbf{x}_{n+1}))$ at every iteration. This leads us to Algorithm 6, which is a slightly modified version of Algorithm 2:

Algorithm 6: Estimating $p_*(\boldsymbol{\chi}_{\min}, \boldsymbol{\chi}_{\max})$ after including a new observation

Require: A set of F points $\zeta^F = (\boldsymbol{\chi}_1, ..., \boldsymbol{\chi}_F)$, a GP mean vector $\hat{\boldsymbol{m}}_n(\zeta^F)$, a variance-covariance matrix $\hat{\Sigma}_n(\zeta^F)$, G as the number of iterations in the MC integration, a zero-matrix L with size $F \times F$, a point $\boldsymbol{\chi}_*$ to be added to the design, GP mean at $\boldsymbol{\chi}_*$ $\hat{m}_n(\boldsymbol{\chi}_*)$ and GP variance at $\boldsymbol{\chi}_*$ $\hat{\Sigma}_n(\boldsymbol{\chi}_*)$

Calculate $\hat{\Sigma}_{n+1}(\zeta^F)$, by using Equation 2.8;
for $g = 1, ..., G$ **do**
  Sample $\mathrm{Y}^{n+1} \sim N(\hat{m}_n(\boldsymbol{\chi}_*), \hat{\Sigma}_n(\boldsymbol{\chi}_*))$;
  Calculate $\hat{\boldsymbol{m}}_{n+1}(\zeta^F)$, by using Equation 2.7;
  Sample $\mathbf{Y} = (\mathrm{Y}_1, ..., \mathrm{Y}_F) \sim N(\hat{\boldsymbol{m}}_{n+1}(\zeta^F), \hat{\Sigma}_{n+1}(\zeta^F))$;
  Let $a = \arg\min_{1 \le i \le F}(\mathrm{Y}_i)$ be the index of the minimum sampled value in $\mathbf{Y}$;
  Let $b = \arg\max_{1 \le i \le F}(\mathrm{Y}_i)$ be the index of the maximum sampled value in $\mathbf{Y}$;
  $L_{a,b} = L_{a,b} + 1$;
end

Output : A probability mass function, where $p_*(\boldsymbol{\chi}_i, \boldsymbol{\chi}_j) = L_{i,j}/G$ for $i, j = 1, ..., F$

We now have everything we need to define our full entropy search algorithm. This is shown in Algorithm 7.

We can also generalise this algorithm to work in the single-task case. For that, see Appendix B.

2.3.3 Example

In this part we will be evaluating the performance of the multi-task entropy search algorithm on a number of test functions. The dimensionality of these test functions ranges from $d = 2$ to $d = 10$. We will compare the entropy search algorithm against a simple benchmark algorithm based on EI. For the benchmark algorithm, at every iteration, we choose a point that maximises $EI_{\min}(\mathbf{x})$ and a point that maximises $EI_{\max}(\mathbf{x})$ and add

Algorithm 7: Multi-task entropy search algorithm

Require: Size of the final design matrix N, an initial design matrix $X_n = (\mathbf{x}_1, ..., \mathbf{x}_n)$ for $n < N$, a computer model f

Evaluate the computer model at X_n to get $\mathbf{Y}_n = f(X_n)$;
Use X_n and $\mathbf{Y}_n$ to fit a Gaussian process model;
Use Algorithm 4 to sample $\frac{F'}{2}$ points from $\pi_{\min}(\boldsymbol{\chi})$ (Equation 2.5) and $\frac{F'}{2}$ points from $\pi_{\max}(\boldsymbol{\chi})$ (Equation 2.6) and call it $\zeta^{F'}$;
Sample F values without replacement from unique values of $\zeta^{F'}$ to get $\zeta^F = (\boldsymbol{\chi}_1, ..., \boldsymbol{\chi}_F)$;
while $n \leq N$ **do**
    for $i = 1, ..., F$ **do**
        Use Algorithm 6 to approximate $p_*(\boldsymbol{\chi}_{\min}, \boldsymbol{\chi}_{\max})$ after adding $\boldsymbol{\chi}_i$ to the design;
        Calculate the entropy $H_i(p_*(\boldsymbol{\chi}_{\min}, \boldsymbol{\chi}_{\max}))$ for point i, by using Equation 2.2;
    end
    Let $n = n + 1$ and choose point $\boldsymbol{\chi}_m$ to the design, where

$$m = arg \min_{i=1,...,F} H_i(p_*(\boldsymbol{\chi}_{\min}, \boldsymbol{\chi}_{\max}))$$

    ;
    Let $y_n = f(\boldsymbol{\chi}_m)$;
    Let $X_n = (X_{n-1}, \boldsymbol{\chi}_m)$ and $\mathbf{Y}_n = (\mathbf{Y}_{n-1}, y_n)$;
    Use X_n and $\mathbf{Y}_n$ to re-fit a Gaussian process model;
    if *Effective sample size of* $\zeta^{F'}$ *is less than* F **then**
        Use Algorithm 4 to sample $\frac{F'}{2}$ points from $\pi_{\min}(\boldsymbol{\chi})$ (Equation 2.5) and $\frac{F'}{2}$ points from $\pi_{\max}(\boldsymbol{\chi})$ (Equation 2.6) and call it $\zeta^{F'}$
    else
        Use Algorithm 5 to sample a new set of $\zeta^{F'}$
    end
    Sample F values without replacement from unique values of $\zeta^{F'}$ to get ζ^F
end

Output : $\hat{\mathbf{x}}^*_{\min} = arg\min_{\mathbf{x}\in\mathbb{X}} \hat{m}_n(\mathbf{x})$ and $\hat{\mathbf{x}}^*_{\max} = arg\max_{\mathbf{x}\in\mathbb{X}} \hat{m}_n(\mathbf{x})$

both of these points into the design matrix. All the code found in this Section can be found on GitHub at `https://github.com/Hendriico/ESMultiTask`.

2.3.3.1 Test functions

We will be evaluating the performance of the algorithms on four test functions. The first function f_1 is a 2-dimensional function defined as

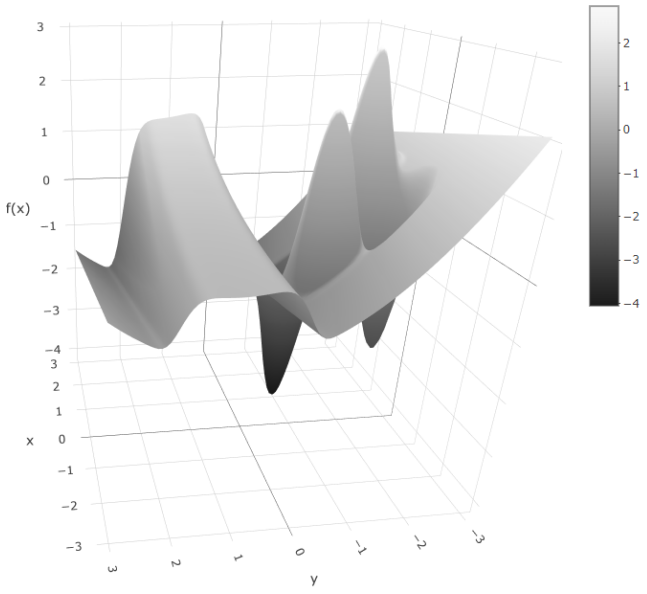

FIGURE 2.9: 2-dimensional test function f_1 evaluated in $[-3, 3]$

$$f_1(\mathbf{x}) = 5x_1 \sin(2x_2) \exp\left(-(x_1)^6 - (x_2 + 1.5)^6\right) + 2\cos\left(\frac{x_2}{5}\right) \exp\left(\frac{x_1}{5} - (x_2 - 1)^6\right) - 0.2x_1 + 0.2x_2^2 - 0.4x_2 - 1.5.$$

We evaluate this function on $x_i \in [-3, 3]$ for $i = 1, 2$, which can be seen in Figure 2.9. It has multiple local minima and maxima and is therefore suitable for multi-task optimisation.

We set the computer model noise to be $Var(f_1(\mathbf{x})) = 0.1$ for all $\mathbf{x} \in \mathbb{X}$.

The second test function we have is the Langermann function (Al-Roomi, 2015b), which is defined for a d-dimensional problem as

$$f_2(\mathbf{x}) = \sum_{i=1}^{m} c_i \exp\left(-\frac{1}{\pi}\sum_{j=i}^{d}(x_j - A_{ij})^2\right) \cos\left(\sum_{j=1}^{d}(x_j - A_{ij})^2\right).$$

We choose to evaluate this function on $x_i \in [0, 10]$ for all $i = 1, ..., d$. We also let $m = 5$, $\boldsymbol{c} = (1, 2, 5, 2, 3)$ and

$$A = \begin{pmatrix} 3 & 5 & 3 & 5 \\ 5 & 2 & 5 & 2 \\ 2 & 1 & 2 & 1 \\ 1 & 4 & 1 & 4 \\ 7 & 9 & 7 & 9 \end{pmatrix}$$

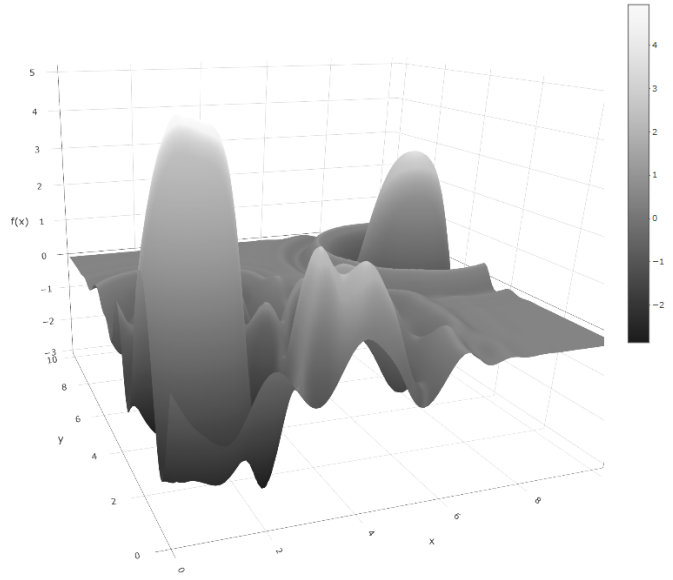

FIGURE 2.10: Langermann function evaluated in $[0, 10]$

In $d = 2$ case, this function can be seen in Figure 2.10.

It again has multiple local minima and maxima and steep drops and hills, making this a suitable test function for multi-task optimisation. In our simulations, we evaluated this function on $d = 4$. We also set the computer model noise to be $Var(f_2(\mathbf{x})) = 0.2$ for all $\mathbf{x} \in \mathbb{X}$.

The third test function we have is the Michalewicz function (Al-Roomi (2015b)), which is defined for a d-dimensional problem as

$$f_3(\mathbf{x}) = -\sum_{i=1}^{d} \sin(x_i) \sin^2\left(\frac{ix_i^2}{\pi}\right).$$

We choose to evaluate this function on $x_i \in [-\pi, \pi]$ for all $i = 1, ..., d$. A $d = 2$ case of this function can be seen in Figure 2.11.

As previously it has multiple local minima and maxima and steep drops and hills, making this a suitable test function for multi-task optimisation. In our simulations, we evaluated this function on $d = 6$. We also set the computer model noise to be $Var(f_3(\mathbf{x})) = 0.2$ for all $\mathbf{x} \in \mathbb{X}$.

The last function we're considering is a stock allocation problem $f_4(\mathbf{x})$. Assume we have a number of stocks that we can invest in and we'd like to distribute our assets to those stocks in a certain way. We can then run computer simulations, modelling the potential gains and losses of these stocks. Based on those simulations, we will then define a risk score of that portfolio. We can define the risk score in a way, so that a high risk score means an allocation that has high risk, but also very high potential reward. Low score

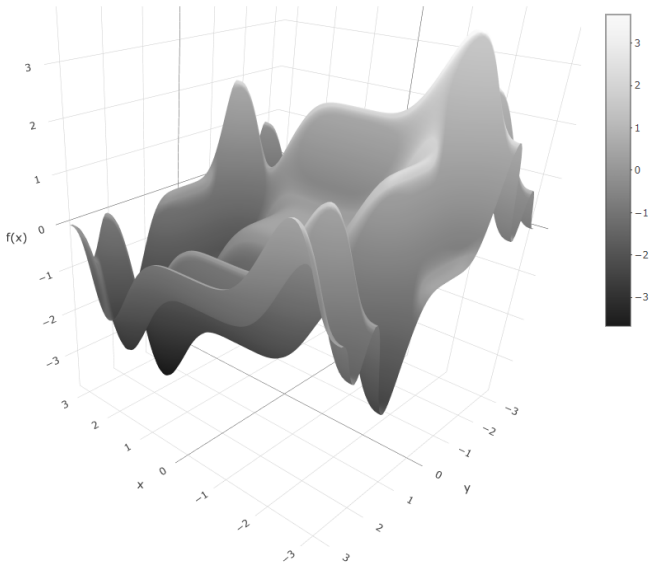

FIGURE 2.11: Michalewicz function evaluated in $[-\pi, \pi]$

will mean much lower risk but also much lower potential rewards. A score of around 0 will mean that a portfolio allocation is sub-optimal in either way and either has high risk and low reward or low risk but no reward at all. We would then like to both minimise and maximise this risk score so that we have multiple options in which we could invest in. In our simulations we are considering 10 potential stocks, so our computer model has $d = 10$. More details about the computer model can be found in Appendix A.

For each optimisation problem, we ran every algorithm 20 times. To find the final location of the optima for a specific algorithm, run and budget, we use the final design from the algorithm and use this to fit a GP. We then find the location of the minimum and the maximum of the GP mean for all evaluated points. In other words, we find $\hat{\boldsymbol{x}}^*_{\min} = \arg\min_{\mathbf{x} \in X_n} \hat{m}_n(\mathbf{x})$ and $\hat{\boldsymbol{x}}^*_{\max} = \arg\max_{\mathbf{x} \in X_n} \hat{m}_n(\mathbf{x})$. To compare the results between the two algorithms, we finally calculate $f(\hat{\boldsymbol{x}}^*_{\min})$ and $f(\hat{\boldsymbol{x}}^*_{\max})$ for each result. We then plot the cumulative improvement of the found solutions.

For the Gaussian process implementation, we used the `laGP` package (Gramacy (2016)) package in R. This package offers reliable and fast GP approximations. We also compared the performance of `laGP` to two other well-known packages, called `DiceKriging` (Roustant et al. (2012)) and `RobustGaSP` (Gu et al. (2018)). As far as we know, there's only a very limited number of papers comparing different implementations of GP modelling (e.g. see Erickson et al. (2018)) and none that would compare these from a (stochastic) Bayesian Optimisation perspective. Because of that, we have included our comparisons in Appendix C.

A 2-dimensional test problem: For our $f_1(\mathbf{x})$ function, we choose a random maximin Latin Hypercube sample (Stein (1987)) of size 20 to be our initial design matrix.

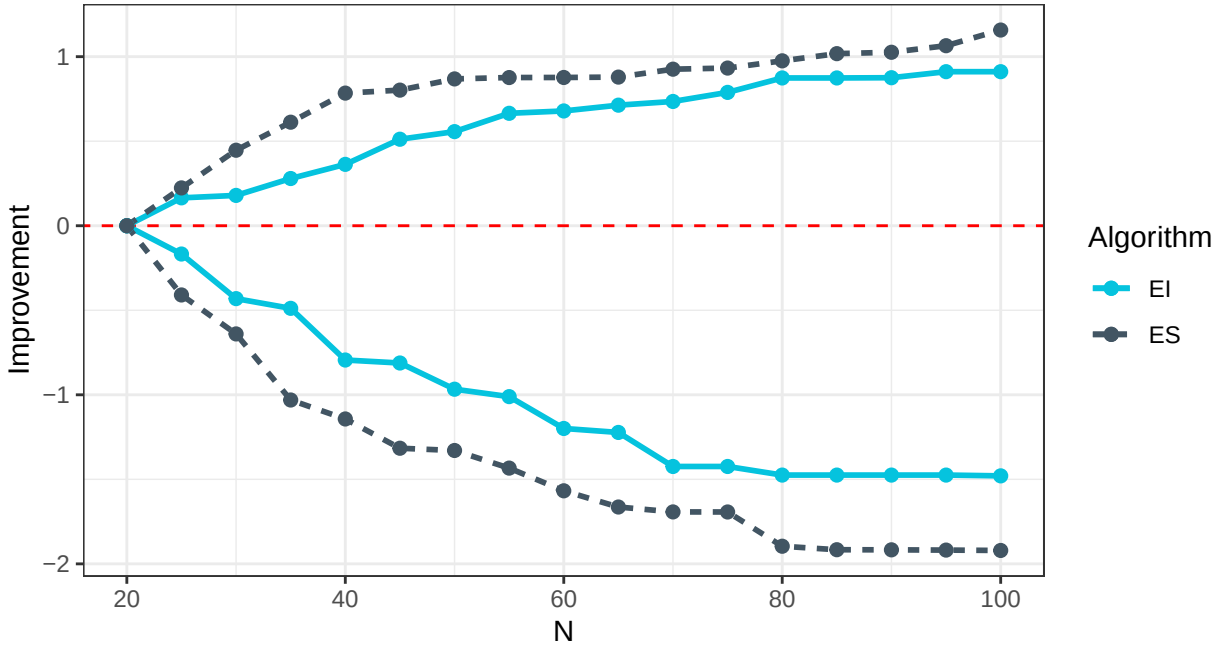

FIGURE 2.12: Comparing the performance of algorithms on f_1

We then evaluate the computer model a total of 100 times for both algorithms. We can then keep track of how much each algorithm manages to improve both the maximum and the minimum value compared to the initial design. The averages are plotted in Figure 2.12.

As we can see, entropy search comfortably outperforms the baseline algorithm for both objectives and for all N. This indicates that entropy search can easily handle simpler problems.

Langermann test problem: For our $f_2(\mathbf{x})$ function, we choose a random maximin Latin Hypercube sample (Stein, 1987) of size 20 to be our initial design matrix. We then evaluate the computer model a total of 120 times for both algorithms. We can then keep track of how much each algorithm manages to improve both the maximum and the minimum value compared to the initial design. The averages are plotted in Figure 2.13.

For the Langermann function, entropy search outperforms our baseline algorithm significantly in both tasks. The baseline algorithm can often get stuck in local optima, while entropy search does not. This seems to suggest that entropy search can balance exploration and exploitation better.

Michalewicz test problem: For our $f_3(\mathbf{x})$ function, we choose a random maximin Latin Hypercube sample (Stein (1987)) of size 30 to be our initial design matrix. We then evaluate the computer model a total of 120 times for both algorithms. We can then

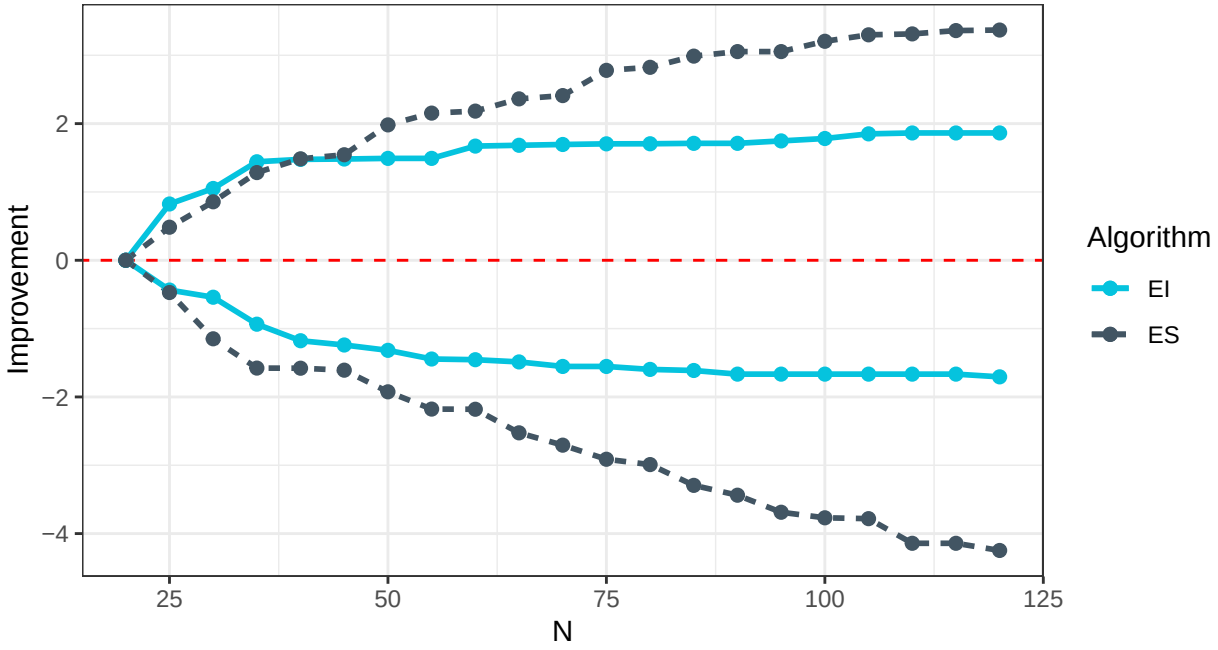

FIGURE 2.13: Comparing the performance of algorithms on f_2

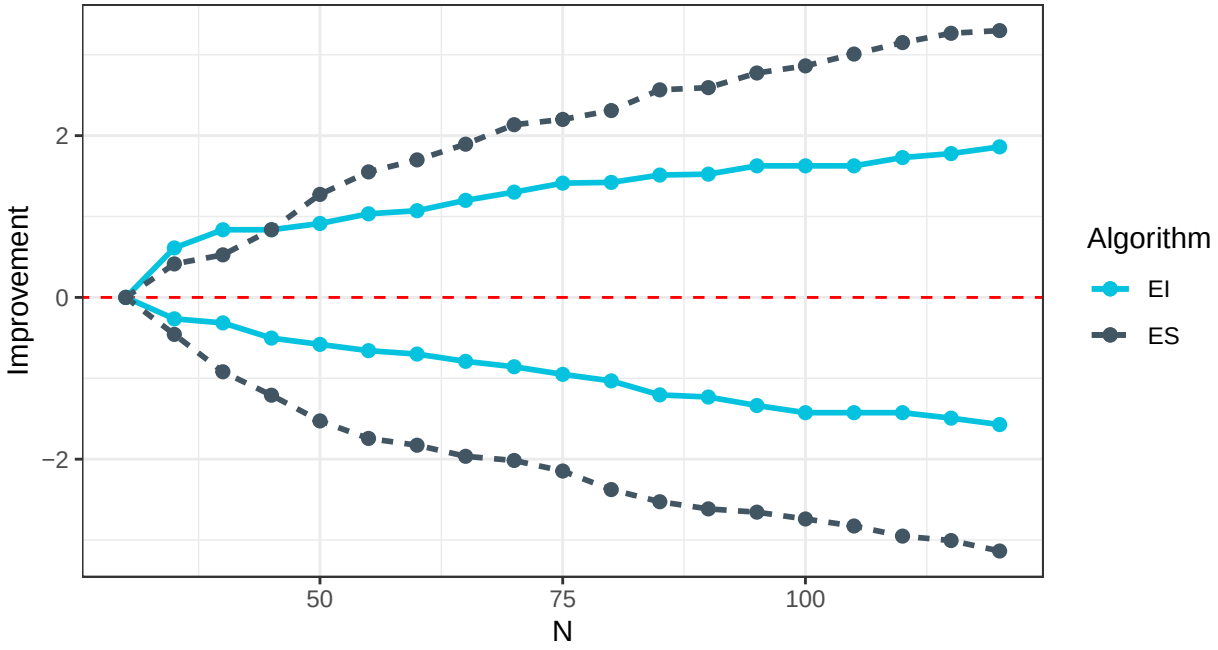

FIGURE 2.14: Comparing the performance of algorithms on f_3

keep track of how much each algorithm manages to improve both the maximum and the minimum value compared to the initial design. The averages are plotted in Figure 2.14.

For the Michalewicz function, entropy search outperforms our baseline algorithm significantly in both tasks. We also examined the performance of a single-task entropy search algorithm for the Michalewicz function and found that it also outperformed all other algorithms (see Appendix B), which means that the success of a multi-task entropy search

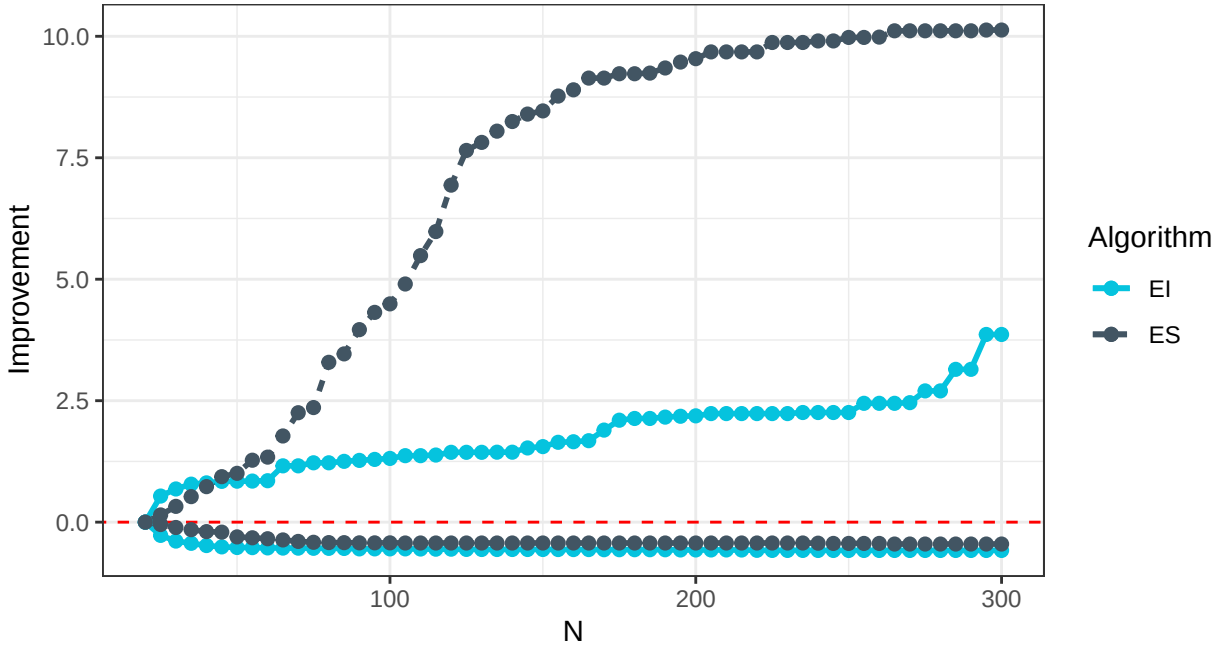

FIGURE 2.15: Comparing the performance of algorithms on f_4

algorithm on this test function is not a surprise.

Stock allocation test problem: For our $f_4(\mathbf{x})$ function, we choose a random maximin Latin Hypercube sample (Stein, 1987) of size 20 to be our initial design matrix. We then evaluate the computer model a total of 300 times for both algorithms. We can then keep track of how much each algorithm manages to improve both the maximum and the minimum value compared to the initial design. The averages are plotted in Figure 2.15.

For the stock allocation problem, neither algorithm manages to improve the minimisation task by a significant amount compared to the maximisation task. This is due to the fact that it's easier to find allocations with a low risk score, since many solutions where the portfolio is well diversified (where there's funds allocated to many different stocks) result from a low risk score. This means that the initial design already has relatively good solutions and there's not much room for improvement in the minimisation task.

However, there is a lot of room for improvement in the maximisation task. When we focus in on just the minimisation task, which can be seen in Figure 2.16, we do see that the baseline reaches a better minimum solution. However, overall it is entropy search that balances the two objectives much better and focuses more on maximisation, which is the more difficult task.

The superior performance of entropy search algorithm came at a cost of higher computational complexity. Average computational for each test problem can be seen in table 2.1.

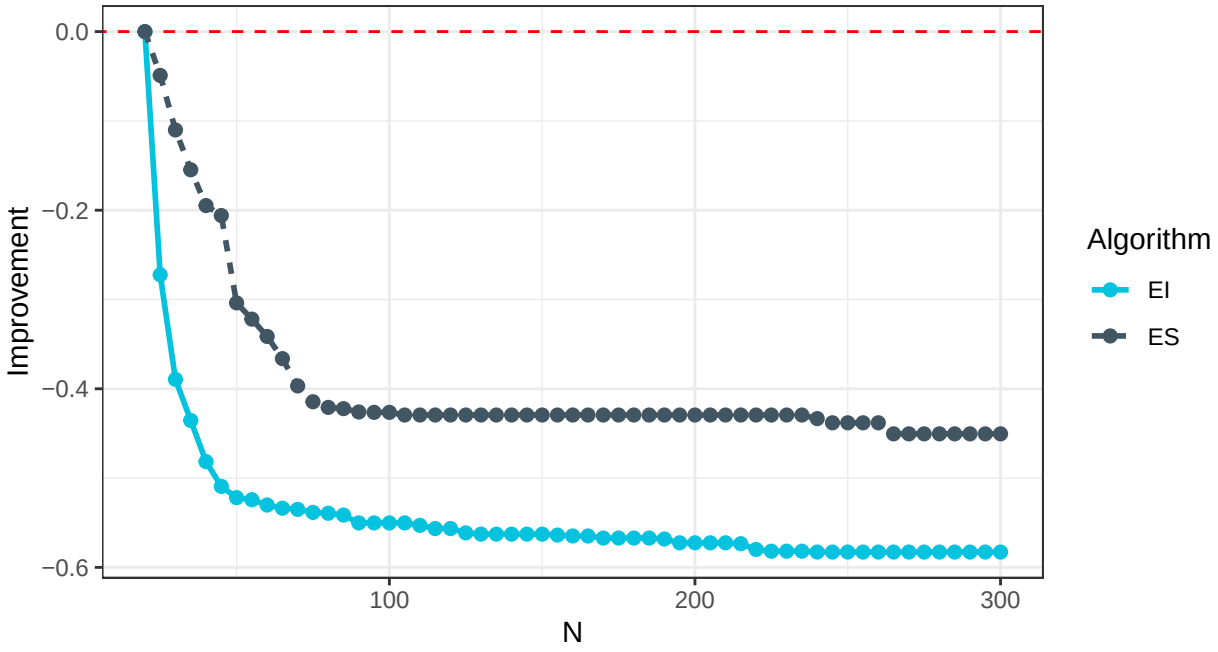

FIGURE 2.16: Comparing the performance of algorithms on f_4 for the minimisation task only

TABLE 2.1: Average time per iteration for different algorithms

Function	Task	Time per Iteration for Entropy Search	Time per Iteration for Baseline Algorithm
f_1	Parallel tempering	0.16 s	N/A
	Acquisition function evaluation	0.7 s	0.07 s
	Total	**0.86 s**	**0.07 s**
f_2	Parallel tempering	0.24 s	N/A
	Acquisition function evaluation	1.25 s	0.09 s
	Total	**1.49 s**	**0.09 s**
f_3	Parallel tempering	0.3 s	N/A
	Acquisition function evaluation	1.8 s	0.13 s
	Total	**2.1 s**	**0.13 s**
f_4	Parallel tempering	0.5 s	N/A
	Acquisition function evaluation	2.3 s	0.18 s
	Total	**2.8 s**	**0.18 s**

From Table 2.1, we see that the entropy search algorithm is on average around 15 times slower than the baseline algorithm. For the entropy search algorithm, we have also shown what tasks require the most computational time. These tasks are parallel tempering to discretise the sample space and acquisition function evaluation through MC integration. Typically, around 15%-20% of the time is spent on parallel tempering and the rest is spent on acquisition function evaluation.

2.4 Discussion

In this chapter, we presented a novel approach to multi-task Bayesian optimisation. Unlike traditional Bayesian optimisation methods that focus on a single extremum, our method is tailored to scenarios where both global extremes are of interest. Our approach is inspired by information theory as our acquisition function is using joint entropy of the location of the minimum and the maximum. This entropy-based criterion explicitly models the dependence between the two optima and naturally balances exploration and exploitation in a multi-task context.

We compared the performance of our algorithm against an Expected Improvement based baseline algorithm. The algorithms were compared on four different synthetic test functions ranging from a minimum of two-dimensional functions to a maximum of ten-dimensional functions.

In all four sets of simulations, our new algorithm outperformed the test algorithm, both in terms of convergence speed as well as the quality of solutions found. This came at a greater computational cost, taking up to 3 seconds per iteration, compared to only a fraction of a second for the baseline algorithm. However, when considering the cost of the computer model, the additional cost of the acquisition function is negligible.

There are also some limitations and open challenges. First, the entropy term involves intractable integrals which are approximated via Monte Carlo integration. While effective it offers good performance and accurate results, this can introduce significant computational overhead, particularly in high-dimensional settings. This means that additional techniques are needed when extending this algorithm to high-dimensional settings. Some possible techniques for high-dimensional Bayesian Optimisation are covered in Chapter 3.

Chapter 3

Multi-Fidelity Bayesian Optimisation in High-Dimensional Settings

3.1 Introduction

Bayesian Optimisation (BO) is a powerful framework for finding the optimum of a computer model. It offers sample-efficient global optimisation via surrogate models and acquisition functions. However, its applicability is severely constrained in high-dimensional settings, especially when we are able to tune the fidelity of the computer model.

In many real-world scenarios, multi-fidelity evaluations are available, where cheaper approximations of the expensive objective are available, with lower levels of accuracy. By leveraging these cheaper fidelities, multi-fidelity Bayesian optimisation methods aim to reduce the cost of optimisation. Despite their success in many fields, existing multi-fidelity Bayesian optimisation algorithms often struggle with scalability in high-dimensional spaces due to the curse of dimensionality and the added complexity of selecting among fidelities.

One very common example of such problems is Bayesian Design of Experiments (DoE). In Bayesian DoE, we are interested in maximising an expected utility function. This function is often high-dimensional when trying to find optimal designs for larger problems. The expected utility function is also almost never available analytically and must therefore be numerically approximated. Often times this is done by using Monte Carlo methods. This means that we can control the accuracy of the approximation by controlling the number of Monte Carlo iterations. These factors combined make it a very difficult problem to solve.

This chapter addresses these challenges by proposing a new algorithm for multi-fidelity Bayesian optimisation in high-dimensional settings. The key innovation is the combination of trust regions with subspace-based optimisation strategies. The algorithm adaptively explores dynamically updated local subsets of the input space, enabling efficient local search for global optimisation.

In section 3.2 we first start by giving an overview of high-dimensional Bayesian optimisation and introducing the main difficulties of doing so. This is then followed by a summary of existing algorithms and acquisition functions, which are described in section 3.3. In section 3.4 we formally define multi-fidelity optimisation and give a brief overview of existing literature on that topic and explain the limitations of these algorithms. We propose a new algorithm in section 3.5, which uses a combination of multiple techniques. In section 3.5.1, we introduce trust regions, which are used for local modelling. Section 3.5.2 covers Thompson sampling, which we use as our acquisition function. Finally section 3.5.3 is used to define the Optimal Computing Budget Allocation algorithm, which we use to decide the fidelities at which we evaluate our computer model. The combined algorithm is then tested against an existing baseline algorithm in section 3.6. For our simulation problem, we consider Bayesian DoE, where we are interested in finding a D-optimal design of size 150×4 for a logistic regression model. The baseline algorithm is ACE, which was specifically created to find optimal Bayesian designs. Our algorithm is shown to find better designs by using a significantly smaller number of computer model evaluations.

3.2 High-dimensional Bayesian Optimisation

High-dimensional computer models are often found in various applications, such as hyper-parameter tuning in machine learning (Turner et al. (2021)), engineering (Kohira et al. (2018)) and chemical design (Griffiths and Hernandez-Lobato (2020)). These computer models are often costly to evaluate and are not available in closed form. Because of that, Bayesian optimisation (BO) is often used for sample-efficient optimisation of these computer models. Despite many advances in the recent years, Bayesian optimisation in high-dimensional settings, especially for noisy computer models, remains a challenging task for multiple reasons.

Firstly, Bayesian optimisation requires fitting a surrogate model to the collected data and fitting accurate global surrogate models in high-dimensional settings is difficult due to the curse of dimensionality and the sparsity of the search space. The most common surrogate model used in Bayesian optimisation is Gaussian Process (GP). Gaussian Process relies on the pairwise distances between points to make predictions. However, when the dimensionality of the computer model increases, the pairwise distances also increase and become more concentrated around a single value (Köppen (2000)). We

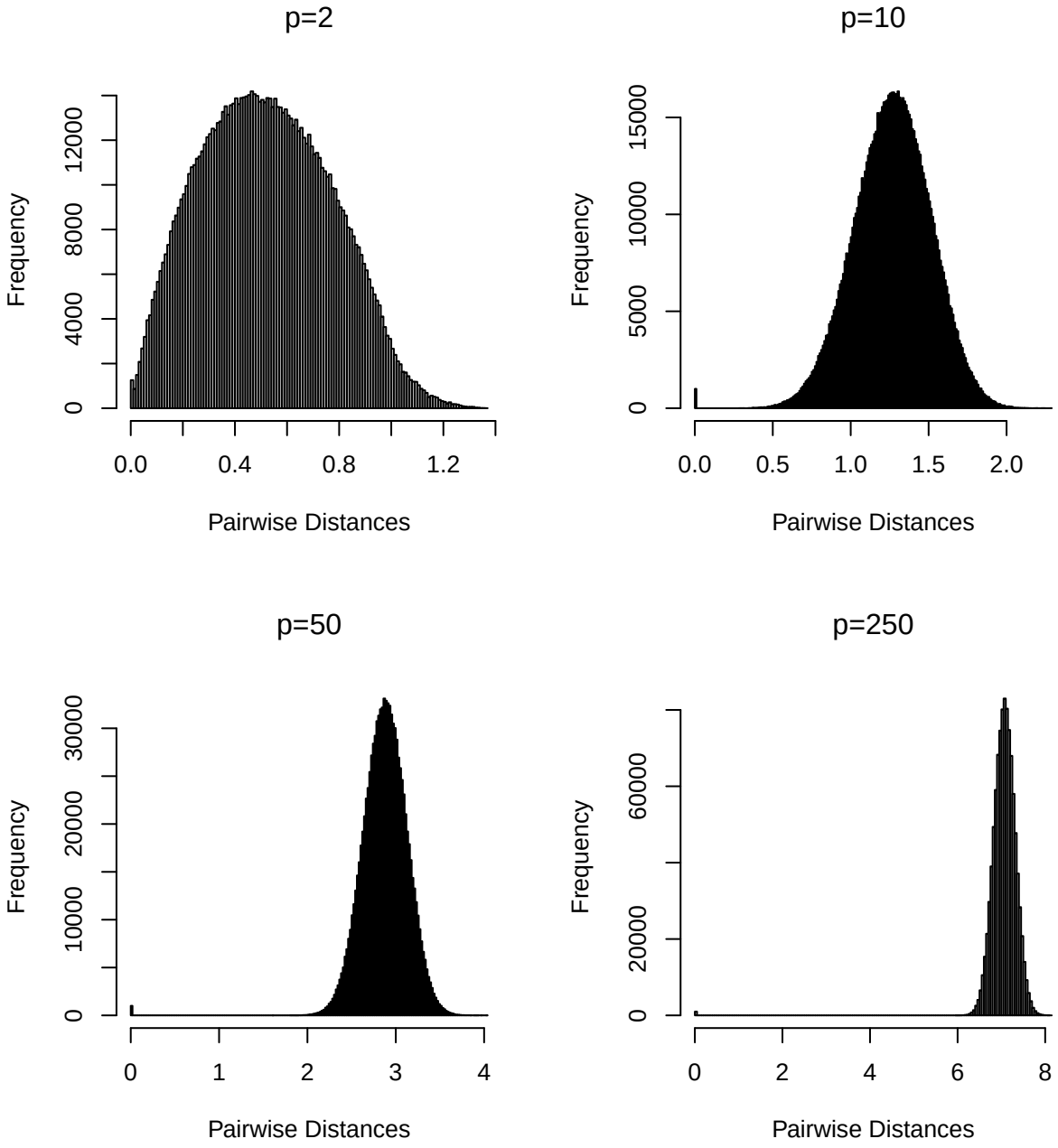

FIGURE 3.1: Pairwise distances for different $p-$dimensional designs

demonstrate this phenomenon in Figure 3.1. This makes inference more difficult and also causes GPs to often severely overestimate the uncertainty.

In Figure 3.1, we created four space filling designs in the hypercube $[0, 1]^p$. The number of rows was kept constant at $n = 1000$ for all designs and the number of dimensions considered were $p = 2$, $p = 10$, $p = 50$ and $p = 250$. We then calculate all pairwise distances between different points and plot a histogram of the distances. As we see, the distances in lower dimensions are spread further apart and vary more. For example, when $p = 10$, the lowest pairwise distances are around 0.5 and highest is around 2.0.

This means the highest distance is around 4 times bigger than the lowest. However, once we get to $p = 250$, the lowest distance is around 6 and highest is around 8, which means the highest distance is only 1.33 times bigger than the smallest.

Standard Gaussian Process also assumes stationarity. In other words, the GP hyper-parameters are assumed to be constant everywhere, which is often not true in practice (Eriksson et al. (2019)). Even though non-stationary Gaussian Processes have been introduced (Assael et al. (2015), Plagemann et al. (2008), Sauer et al. (2021)), these methods are often only suitable for low-dimensional computer models and small design matrices, since they are too computationally expensive to be used with large matrices.

Secondly, balancing exploration and exploitation is difficult in higher dimensions. Usually it is the acquisition function that balances both objectives, using the mean and the variance of the Gaussian Process. However, in higher dimensions, the search space is very sparse, which means that the predictive variance is very similar for different points, making standard acquisition functions unsuitable for such problems (Tian et al. (2019)). To solve this, various changes have been made to the standard Bayesian optimisation framework.

3.3 Previous Work in High-dimensional Bayesian Optimisation

Due to the fact that GPs are difficult to fit in higher dimensions, alternative models have been proposed. These include random forests (Hutter et al. (2011)) and Bayesian neural networks (Snoek et al. (2015)). However, these methods are often lacking in their ability to quantify uncertainty or simply lack the level of model flexibility that GPs offer. Because of that, we will focus on GPs as our surrogate models.

When using GPs for Bayesian optimisation in high dimensions, additional techniques are needed. A common technique is dimensionality reduction. Some of the earlier works in that field focused on variable selection. In these algorithms, it is assumed that only a small number of variables are important, which reduces the dimensionality of the problem. An early work in that field is (Chen et al. (2012)).

Since in reality it's highly unlikely that some variables are redundant, researchers have applied other methods of dimensionality reduction to BO. For example, Wang et al. (2013) used linear combinations of existing variables to create new variables. They used the fact that given $x \in \mathbb{R}^p$ and a random matrix $X_A \in \mathbb{R}^{p \times p'}$, there exists a point $w \in \mathbb{R}^{p'}$, so that $f(x) = f(X_A \cdot w)$, where $p' < p$. This allowed them to run Bayesian optimisation in a lower-dimensional setting than the original one.

Another method for fitting GPs for high-dimensional functions, is assuming an additive structure of the underlying function. This assumes that the true high-dimensional function is a sum of functions that each depend on only a subset of the parameters. Examples of work in this area include Gardner et al. (2017) and Kandasamy et al. (2015).

A third method for high dimensional optimisation is local optimisation using trust regions, which is a technique commonly used in mathematical optimisation (Yuan (1999)). It has also been used in a number of high-dimensional Bayesian Optimisation algorithms, first of which was the Trust Region Bayesian Optimisation (TuRBO) algorithm, introduced by Eriksson et al. (2019). While local modelling does not decrease the dimensionality of the problem, it does reduce the search space by only focusing on small areas around promising regions. It selects promising regions to be centered around the best set of already evaluated points.

While dimensionality reduction methods have been shown to be often useful, they make strong assumptions about the underlying structure and have therefore been shown to fail in cases where their assumptions about the underlying problem structure don't hold. They also often need to fit a large number of GPs to learn about the underlying structure, which makes them scale poorly to large designs. We therefore focus mostly on local modelling, using trust regions.

3.3.1 Acquisition functions

In high dimensions, evaluation budget is usually large and parallel computations are often available, which means that batch acquisition functions are used. There have been a lot of batch acquisition functions proposed (e.g. Chevalier and Ginsbourger (2013), González et al. (2015), Palma et al. (2019), Wang et al. (2019)). Most of them are either extensions of q-EI (Ginsbourger et al. (2008)) or require sampling different hyperparameters for the Gaussian Process model. This means that they often do not scale well to large batch sizes (Eriksson et al. (2019)). A common batch acquisition function for large batch sizes is Thompson sampling (Thompson (1933)), which has been used, for example, in Eriksson et al. (2019) and Hernández-Lobato et al. (2017). We will come back to this in section 3.5.

3.4 Multi-fidelity optimisation

In many occasions, the computer model is noisy. This means that it is of the form $f(\mathbf{x}) = \tilde{f}(\mathbf{x}) + \epsilon(\mathbf{x})$, where $\epsilon(\mathbf{x})$ is a random variable. For the rest of this paper, we will assume that $E[\epsilon(\mathbf{x})] = 0$ and $Var[\epsilon(\mathbf{x})] = v^2(\mathbf{x})$, which means that $f(\mathbf{x})$ is an unbiased estimator of the true function $\tilde{f}(\mathbf{x})$.

The source of noise for the computer model depends on the mathematical and computational methods used in its implementation. With noisy computer models, we are often able to make the output more or less accurate by increasing or decreasing the computational time. This is also known as fidelity of the computer model, where high fidelity means high accuracy and high computational time and vice versa. For example, when using Monte Carlo methods in computer models, the output can be represented as

$$\overline{f}_M(\mathbf{x}) = \frac{1}{M}\sum_{m=1}^{M} f_m(\mathbf{x}) = \frac{1}{M}\sum_{m=1}^{M} (\tilde{f}(\mathbf{x}) + \epsilon_m(\mathbf{x})).$$

This means that the uncertainty of the computer model $\overline{f}_M(\mathbf{x})$ is $Var\left[\overline{f}_M(\mathbf{x})\right] = \frac{v^2(\mathbf{x})}{M}$ and depends on M. We can also apply this principle to other noisy computer models that do not depend on Monte Carlo methods. We can simply evaluate the computer model multiple times at the same location and use the mean of those evaluations as the more accurate output. This type of optimisation problem, where we are able to tune the precision of the computer model is sometimes referred to as multi-fidelity optimisation.

As an example, consider the following function

$$f_F(x) = (3.6x - 1.4)\sin(22x) + \epsilon(x),$$

where $\epsilon(\mathbf{x}) \sim N(0, 1)$. We can choose a design X_n of size $n = 20$ and evaluate it at four different fidelities. We choose $M = 1$, $M = 4$, $M = 10$ and $M = 20$. We then fit a GP to each of those four designs and compare the results in Figure 3.2.

As seen in Figure 3.2, with the low fidelity evaluations, the output is much more uncertain and more inaccurate, however it is also much faster. On the other hand when we increase fidelity to $M = 20$, the GP is much more accurate, almost perfectly matching the true function. Unfortunately, it is also 20 times as expensive as the cheap option.

The main question in multi-fidelity optimisation is how to choose which points we should evaluate at only lower fidelities and which ones we should also evaluate at higher fidelities to allow for optimisation as efficiently as possible.

There has been a considerable amount of work carried out in the field of multi-fidelity optimisation. Some of the earlier works in the field were focused on creating new acquisition functions, based on expected improvement (EI) (Schonlau (1997)), which could choose a new point as well as the fidelity at which to evaluate it. For example Huang et al. (2006a) created a new acquisition function called augmented expected improvement (AEI) and Forrester (2013) created expected quantile improvement (EQI). TSSO (Quan et al. (2013)) and eTSSO (Pedrielli et al. (2020)) algorithms have two stages at each iteration of the algorithm. They have an initial exploration or search stage, where they use modified EI to find new areas with high potential. After the initial stage, they

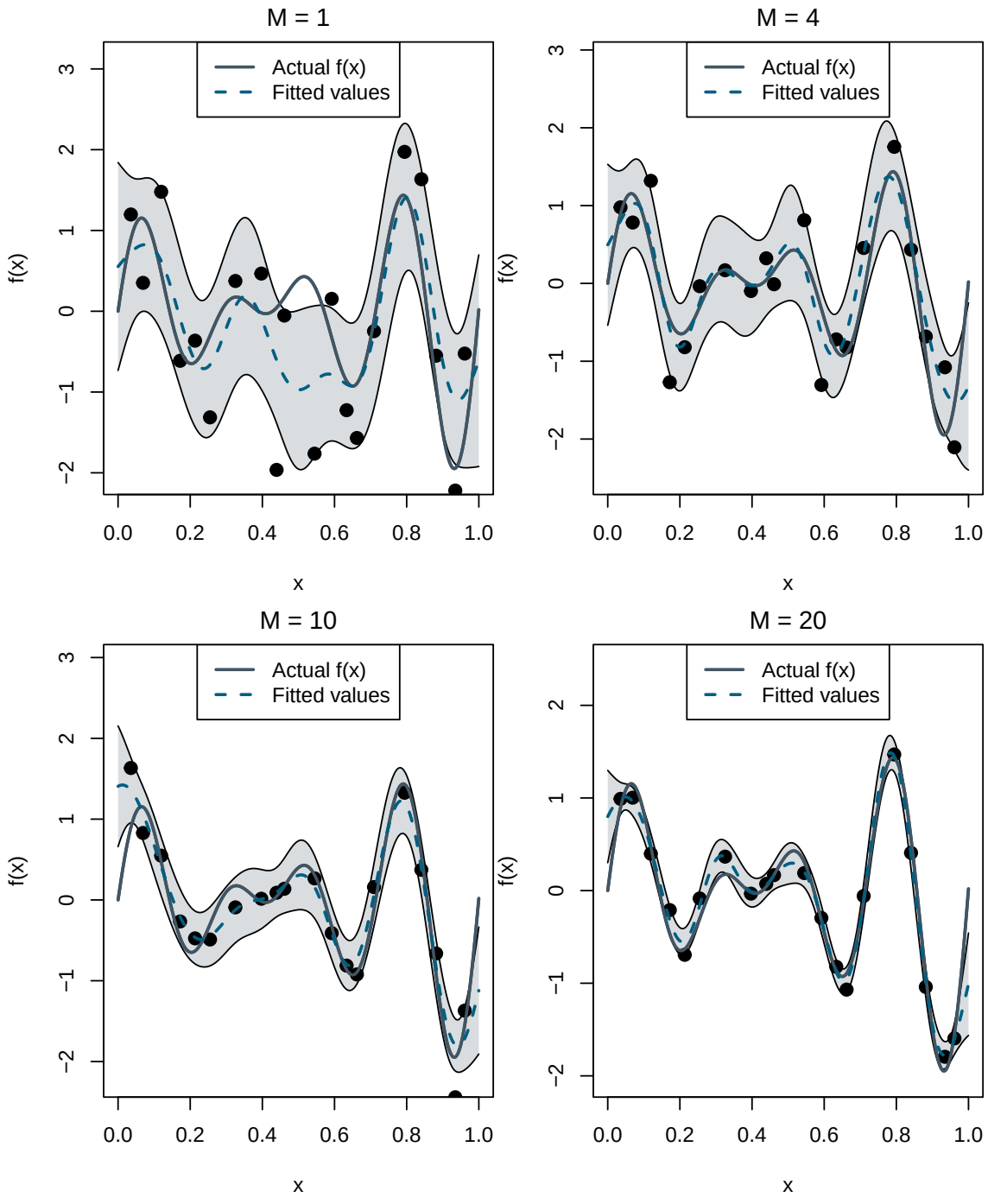

FIGURE 3.2: Design and Gaussian Process fits under different computer model fidelities M

also have an allocation stage, where they allocate more resources to the already evaluated promising points to reduce uncertainty. This is achieved by using the Optimal Computing Budget Allocation algorithm (Chen et al. (2000)).

In more recent works, multiple GPs are usually fitted for different fidelities of the computer model. They can either fit separate and independent GPs for each of the fidelities, which is done in the MF-GP-UCB algorithm (Kandasamy et al. (2016)), or fit more complex models, which also model the correlation between different fidelities. A popular model for such tasks is Co-Kriging (Myers Donald (1982)). One of the first algorithms to use Co-Kriging was Forrester et al. (2007). More recently, Co-Kriging has been used in entropy-based search algorithms. Of these, one of the most popular ones is the max-value entropy search (Takeno et al. (2020)). In addition to Co-Kriging, other complex models have also been considered. For example, Li et al. (2020) uses deep neural networks as the surrogate model and an entropy-based acquisition function.

Even though these algorithms have been shown to be useful for solving many problems, they do not scale well for high-dimensional problems. They are either unsuitable for batch acquisition functions or rely on complex surrogate models that are computationally expensive to evaluate in high-dimensional settings. This leads us to wanting to develop a new algorithm for multi-fidelity optimisation in high-dimensional settings.

3.5 Algorithm

We have assumed that our computer model is stochastic and its accuracy can be improved by evaluating it multiple times at the same location $\mathbf{x}$ and calculating $\overline{f}_M(\mathbf{x})$. Assume now that we are interested in maximising our computer model $f(\mathbf{x})$. Note that it's possible to generalise it for the minimisation tasks by maximising $-f(\mathbf{x})$. In the noiseless setting, it is easy to define the best solution. One can simply look at the best observed value so far. However, this is unsuitable in the noisy setting, as we may choose a point that by pure chance had a very high and very noisy observation. Choosing that point as the best would lead to slowing down our algorithm as it would not choose a good centre point. To take into account all of the uncertainty, we use the lower confidence bound (LCB) measure to define the best solution in the noisy setting. The same measure has also been used by Huang et al. (2006a), Forrester (2013) Picheny et al. (2013), etc. We will define this as

$$f_M^{LCB}(\mathbf{x}) = \overline{f}_M(\mathbf{x}) - \Phi^{-1}(\alpha)\sqrt{\frac{\hat{v}^2(\mathbf{x})}{M}},$$

where $\Phi^{-1}()$ is the inverse CDF of a standard normal distribution and $\hat{v}(\mathbf{x})$ is an estimate of v at the point $\mathbf{x}$, calculated as

$$\hat{v}(\mathbf{x}) = \sqrt{\frac{\sum_{m=1}^{M}(f_m(\mathbf{x}) - \overline{f}_M(\mathbf{x}))^2}{M-1}}.$$

Finally, $\alpha \in [0.5, 1)$ is a tuning parameter, which controls the risk tolerance. If we choose $\alpha = 0.5$, then $f_M^{LCB}(\mathbf{x}) = \overline{f}_M(\mathbf{x})$ and we are ignoring the variance of $\overline{f}_M(\mathbf{x})$ completely and are therefore taking a risk. If we choose a larger value of α (e.g. $\alpha = 0.9$), then we favour points that we've evaluated more times and whose variance is smaller, which means we are less willing to take a risk when it comes to choosing the best point. To be able to calculate $f_M^{LCB}(\mathbf{x})$ robustly, we require the computer model to be evaluated at least $M_{\min}$ number of times at each $\mathbf{x}$.

The algorithm will then be divided into two stages. Each stage will use a different local modelling technique as well as a different acquisition function. In both stages, we will first use an acquisition function to propose new points from the local model to be evaluated cheaply. We then allocate more budget to more promising points, which will allow us to use computational resources as efficiently as possible. These steps are written out in Algorithm 8.

Algorithm 8: Outline of an individual stage in the global algorithm

```
while Computational budget > 0 do
    Fit a local model;
    Use an acquisition function A to propose new points to be evaluated cheaply;
    Allocate more resources to the more promising points;
    Update the local model;
end
```

3.5.1 Local modelling

In high-dimensional settings, fitting global GPs that fit well is an extremely difficult task. We are therefore not aiming to fit global models that fit well everywhere, but rather local models that fit well in a local region. The computational complexity of fitting a Gaussian Process is usually $O(N^3)$, where N is the number of rows in the design matrix (Park and Huang, 2016). When fitting local models, we need a smaller number of points $n < N$ to fit an accurate model, which means that the overall computational complexity of the local model is much lower. For example when $N = 2n$, then the local model would be around 8 times faster.

The first method that we will be using for local modelling is fitting surrogate models in a local trust region (TR). Trust region is a small subset of the entire design space, where

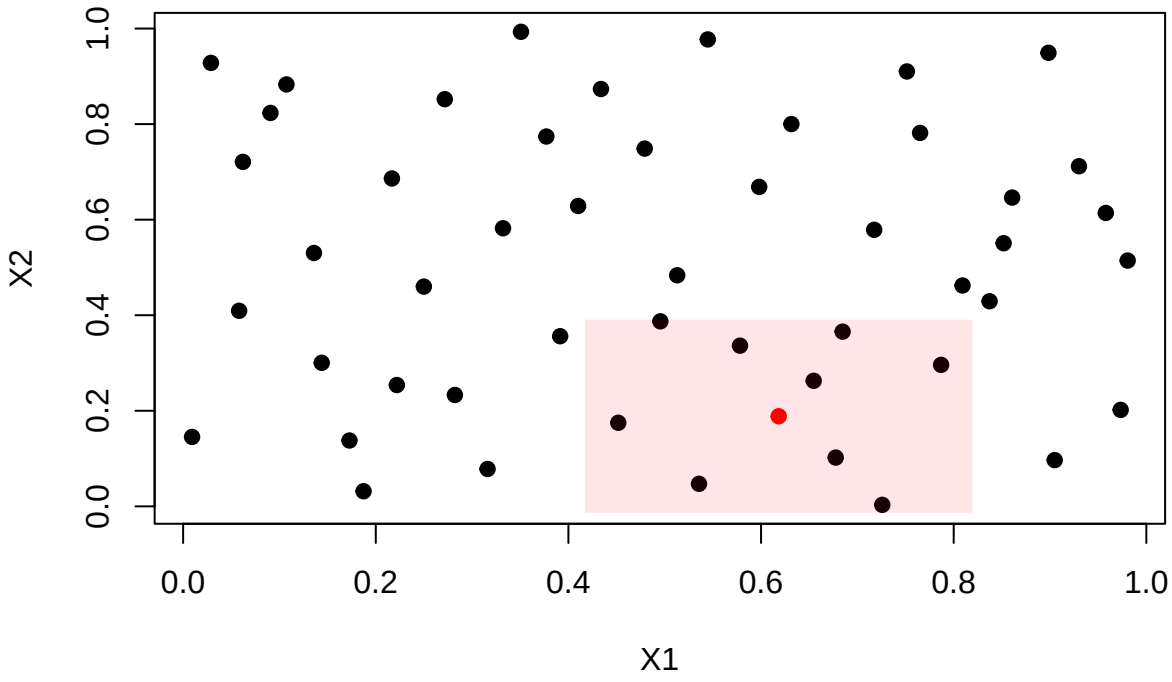

FIGURE 3.3: Local Trust Region

we limit our optimisation problem to take place. The TR can have various shapes. The more common ones include hyperspheres (Bader, 2009), hyperrectangles (Vogklis and Lagaris, 2019) and polyhedrals (Bodur et al., 2021). An example of a rectangular trust region can be seen in Figure 3.3.

Similarly to Eriksson et al. (2019), we choose the trust regions to be a hyperrectangle, centered around the point with the highest observed LCB value. The size of the hyperrectangle depends on its side length L. A larger value of L means that we are able to explore solutions in a larger area and find new promising regions. A smaller value of L means that we can focus on exploitation and find the local optimum as accurately as possible. We therefore want to change the value of L depending on the success of the algorithm. If the algorithm keeps making progress and finding good solutions, then it means that there is much reward in exploring, as we still have not found a single best area with the most promising set of solutions. We can therefore increase L to explore an even larger area of the design space. If the opposite is true and the algorithm fails to find any new good solutions, then we have done enough exploring and found a single area that has a high probability of containing the best solution. We then want to start focusing on that single area and therefore want to reduce L to do more exploitation. In reality this means increasing L when we have φ_{succ} number of consecutive improvements

to the highest LCB value and decreasing L when we fail to improve the highest LCB value more than φ_{fail} number of times.

In TuRBO, GPs are fitted to the entire design, which can be inefficient in terms of computational resources. This can be derived from Equation 1.2. The correlation between points depends on their distance. When we include points that are far away from the TR, then the effect they have on predictions is very small. Therefore, only using points closest to the TR to fit the model, makes the most amount of sense. A similar strategy has been used in other types of Bayesian Optimisation algorithms, e.g. see Daulton et al. (2022), as well as packages for implementing GPs, e.g. see `laGP` (Gramacy (2016)) in `R` (R Core Team (2023)).

3.5.2 Thompson Sampling

After we choose an initial design matrix $X_{n_{init}}$ of size $n_{init} \times p$ and evaluate the computer model at all points $M_{\min}$ times, we choose the point with the highest LCB value to be the best point so far. In other words, we choose

$$\mathbf{x}_{centre} = \arg \max_{\mathbf{x} \in X_{n_{init}}} f^{LCB}_{M_{\min}}(\mathbf{x})$$

We then use Thompson sampling to propose new points in the hyperrectangle, centred around $\mathbf{x}_{centre}$.

In Thompson sampling, we are drawing random samples from a GP posterior. Then for each sample, we choose the point which had the highest or lowest sampled value. This method is quick and scales linearly with the size of the batch. As an example, consider the one-dimensional function

$$f_G(x) = (3.6x - 1.4)\sin(22x),$$

which we are interested in maximising. We choose an initial design of size $n = 5$ and fit a GP. This is shown in Figure 3.4 alongside with the expected improvement function for reference.

To now get a batch of new points to evaluate, we use Thompson sampling and draw ten samples from the GP posterior. For each sample we choose the maximum sampled value to add to the design. This process is shown in Figure 3.5.

In Figure 3.5, every light blue line is a single sample and the red dots are the values selected by Thompson sampling. This is more formally defined in Algorithm 9.

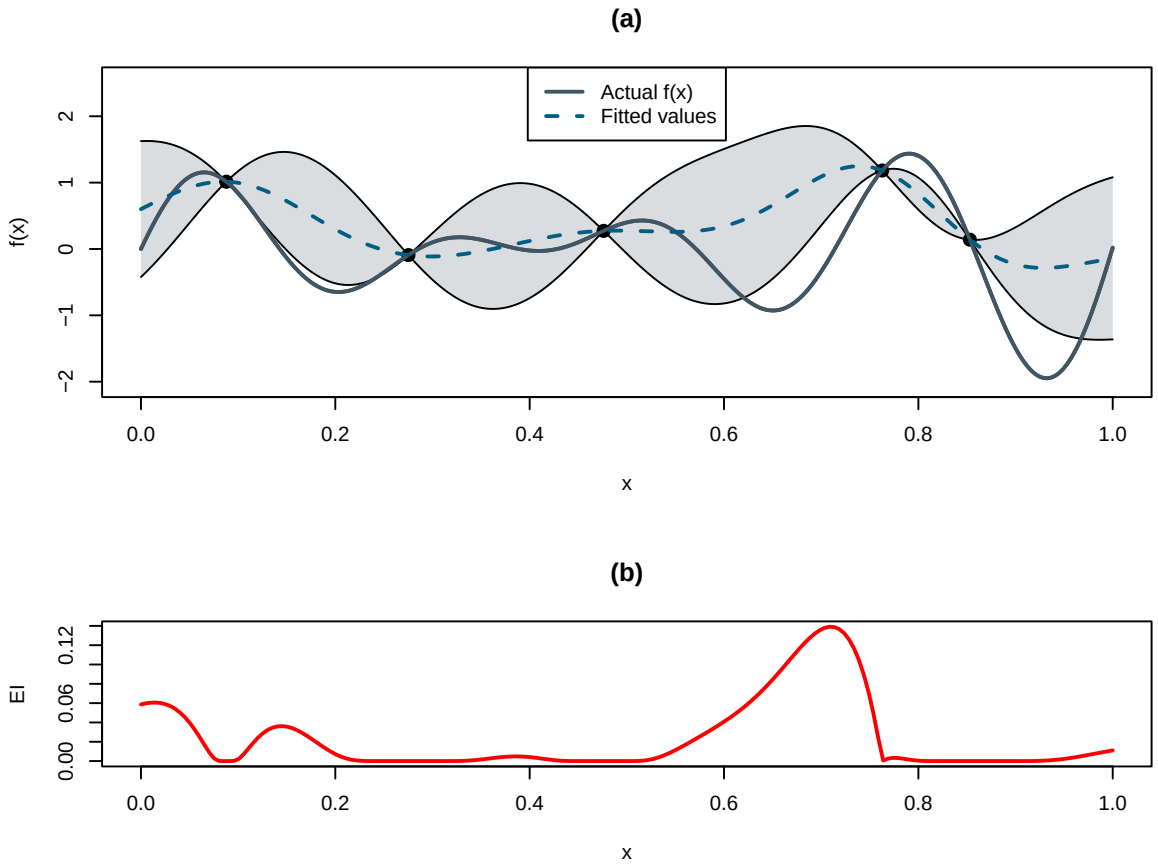

FIGURE 3.4: (a) Example of a GP fitted to $f_G(x)$ (b) Expected Improvement function evaluated

Algorithm 9: Thompson sampling

Require: A centre point $\mathbf{x}_{centre}$ of dimension p, side length of the hyperrectangle L, a fitted GP, number of candidate points n_{cand} and the size of the batch B

Output : A $B \times p$ matrix $\Xi_B = (\boldsymbol{\xi}_1, ..., \boldsymbol{\xi}_B)^T$ of new points to be evaluated by the computer model

Create an $n_{cand} \times p$ matrix $\zeta = (\boldsymbol{\chi}_1, ..., \boldsymbol{\chi}_{n_{cand}})^T$, where all points lie in the hyperrectangle, which has a side length of L and is centered around $\mathbf{x}_{centre}$;
Calculate $\hat{\boldsymbol{m}}_n(\zeta)$ and $\hat{\Sigma}_n(\zeta)$ by using Equations 1.4 and 1.5 respectively;
for $b = 1, ..., B$ **do**
    Sample $\mathbf{Y} = (\mathrm{Y}_1, ..., \mathrm{Y}_{n_{cand}}) \sim N(\hat{\boldsymbol{m}}_n(\zeta), \hat{\Sigma}_n(\zeta))$;
    Let $a = \arg\min_{1 \leq i \leq n_{cand}}(\mathrm{Y}_i)$ be the index of the maximum sampled value in $\mathbf{Y}$;
    $\boldsymbol{\xi}_b = \boldsymbol{\chi}_a$;
end

After we get B new points from the Thompson sampling algorithm, we evaluate each of those points $M_{\min}$ times to get $\overline{f}_{M_{\min}}(.)$. Since this is only a cheap approximation

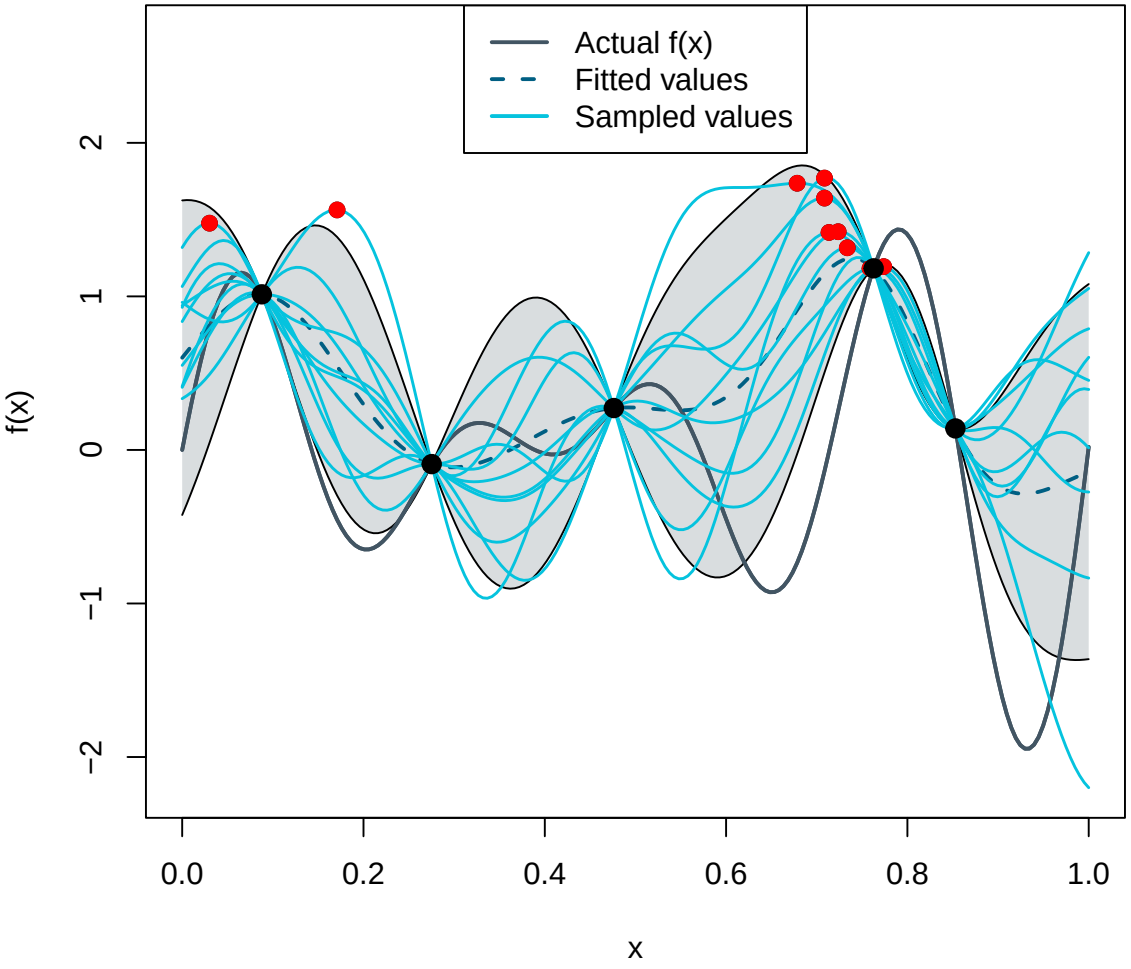

FIGURE 3.5: Thompson sampling acquisition function

of $\tilde{f}(.)$, we are now interested in allocating more resources to the points in Ξ that are more promising.

3.5.3 Optimal Computing Budget Allocation

In the first stage of the algorithm, we proposed B new points for the computer model to evaluate cheaply. We are now interested in allocating more computational resources to the promising solutions. To do that, we use the optimal computing budget allocation (OCBA) algorithm (Chen et al., 2000).

Assume that we have B points with a sample mean $\left(\overline{f}_M(\boldsymbol{x_1}), ..., \overline{f}_M(\boldsymbol{x_B})\right)$ and variances $(v^2(\mathbf{x}_1), ..., v^2(\mathbf{x}_B))$, then we are interested in allocating T replications among these B points, such that the approximate probability of correct selection (APCS) after these extra replications is maximised. APCS is the probability that choosing the solution with

the highest sample mean leads to choosing the solution with the highest actual function value. So in other words

$$APCS = P\left[\tilde{f}(\boldsymbol{x_b}) = \max_{i=1,\dots,B} \tilde{f}(\mathbf{x}_i) \middle| \overline{f}_{M+t_b}(\mathbf{x}_b) = \max_{i=1,\dots,B} \overline{f}_{M+t_i}(\mathbf{x}_i)\right],$$

where $\sum_{i=1}^{B} t_i = T$.

According to Theorem 1 in Chen et al. (2000), this can be achieved by choosing

1) $\frac{t_i}{t_j} = \left(\frac{\hat{v}(\mathbf{x}_i)/\delta_{b,i}}{\hat{v}(\mathbf{x}_j)/\delta_{b,j}}\right)^2$ for $i, j \in \{1, ..., B\}$ and $i \neq j \neq b$

2) $t_b = \hat{v}(\mathbf{x}_b)\sqrt{\sum_{i=1, i\neq b}^{B} \frac{t_i^2}{\hat{v}^2(\mathbf{x}_i)}}$,

where t_i is the number of extra replications allocated to point $\mathbf{x}_i$, $\delta_{b,i} = \overline{f}_M(\mathbf{x}_b) - \overline{f}_M(\mathbf{x}_i)$ and $b = \arg\max_{i\in\{1,\dots,B\}} \overline{f}_M(\mathbf{x}_i)$.

We can then use these rules in an algorithm, which is defined in Algorithm 10.

Algorithm 10: OCBA algorithm

Require: B sample means $\left(\overline{f}_M(\boldsymbol{x_1}), ..., \overline{f}_M(\boldsymbol{x_B})\right)$ and variances $(\hat{v}^2(\mathbf{x}_1), ..., \hat{v}^2(\mathbf{x}_B))$, number of replications to add T

Let $b = \arg\max_{i \in \{1,...,B\}} \overline{f}_M(\mathbf{x}_i)$;
Initialise $t_1 = t_2 = ... = t_B = 1$;
if $b = 1$ **then**
  $start = 2$
else
  $start = 1$
end
for $i = start + 1, ..., B$ **do**
  if $i \neq b$ **then**
    $t_i = t_{start} \left(\frac{\hat{v}(\mathbf{x}_i)/\delta_{b,i}}{\hat{v}(\mathbf{x}_{start})/\delta_{b,start}} \right)^2$;
  end
end
Let $t_b = \hat{v}(\mathbf{x}_b) \sqrt{\sum_{i=1, i \neq b}^{B} \frac{t_i^2}{\hat{v}^2(\mathbf{x}_i)}}$;
Let $S = \sum_{i=1}^{B} t_i$;
Let $t_i = \left\lfloor t_i \frac{T}{S} \right\rceil$ for all $i = 1, ..., B$, where $\lfloor . \rceil$ means rounding to the nearest integer;
Let $S = \sum_{i=1}^{B} t_i$;
if $S < T$ **then**
  Let $t_i = t_i + 1$ for $T - S$ number of $i \in \{1, ..., B\}$ with the largest values of t_i;
end
if $S > T$ **then**
  Let $t_i = t_i - 1$ for $S - T$ number of $i \in \{1, ..., B\}$ with the smallest values of t_i, where $t_i > 0$;
end

Output : Return a vector $(t_1, ..., t_B)$

After running the OCBA algorithm and allocating T replications among points in Ξ_B, we can run the OCBA algorithm again for all points in Ξ_B and X_n. This is especially important in the later stages of the algorithm, when we're trying to focus more on exploitation and are really trying to focus on reducing uncertainty about $\overline{f}_M(.)$.

With all that put together, we can define a full algorithm for multi-fidelity high dimensional optimisation algorithm. This can be seen in Algorithm 11.

Algorithm 11: Multi-Fidelity Bayesian Optimisation Algorithm

Require: A computer model f, computational budget N, initial design matrix $X_{n_0} = (\mathbf{x}_1, ..., \mathbf{x}_{n_0})^T$ of size $n_0 \times p$, the minimum number of replications for sampling a new point $M_{\min}$, batch size for new locations B, initial side length of the local TR L_0, maximum and minimum side length of the local TR $L_{\max}$ and $L_{\min}$ respectively, maximum amount of consecutive successes and fails φ_{succ} and φ_{fail} respectively and number of replications allocated to the OCBA algorithm T

Evaluate f $M_{\min}$ times at each point in X_n and calculate $\boldsymbol{Y}_n = (\overline{f}_{M_{\min}}(\mathbf{x}_1), ..., \overline{f}_{M_{\min}}(\mathbf{x}_n))$, $\boldsymbol{v}_n = (\hat{v}(\mathbf{x}_1), ..., \hat{v}(\mathbf{x}_n))$ as well as $\boldsymbol{Y}_n^{LCB} = (f_{M_{\min}}^{LCB}(\mathbf{x}_1), ..., f_{M_{\min}}^{LCB}(\mathbf{x}_n))$;
Set $n = n_0$ and calculate the initial cost $G = n \cdot M_{\min}$;
Set $L = L_0$;
while $G < N$ *and* $L > L_{\min}$ **do**
  Let $\delta_C = \arg\max_{\mathbf{x} \in X_n} f_M^{LCB}(\mathbf{x})$;
  Use only X_n that is within $2L$ of δ_C and the associated $\boldsymbol{Y}_n$ to fit a Gaussian Process;
  With δ_C use Algorithm 9 to choose B new points $\Xi_{B;S} = (\boldsymbol{\xi}_{1;S}, ..., \boldsymbol{\xi}_{B;S})$;
  Let $\Xi_B = (\boldsymbol{\xi}_1, ..., \boldsymbol{\xi}_B)$ be a $B \times p$ matrix where columns S come from $\Xi_{B;S}$ and the other columns come from $\delta_{C;-S}$;
  Calculate $\boldsymbol{Y}_{new} = (\overline{f}_{M_{\min}}(\boldsymbol{\xi}_1), ..., \overline{f}_{M_{\min}}(\boldsymbol{\xi}_B))$ as well as $\boldsymbol{v}_{new} = (\hat{v}(\boldsymbol{\xi}_1), ..., \hat{v}(\boldsymbol{\xi}_B))$;
  Let $G = G + B \cdot M_{\min}$;
  Use Algorithm 10 to allocate T replications among points in Ξ_B;
  Update $\boldsymbol{Y}_{new}$ and $\boldsymbol{v}_{new}$ and let $G = G + T$;
  if $\max \boldsymbol{Y}_{new}^{LCB} > \max \boldsymbol{Y}_n^{LCB}$ **then**
    $success = success + 1$ and $fail = 0$;
    if $success > \varphi_{succ}$ **then**
      $L = \min\{2L, L_{\max}\}$ and $success = 0$;
    end
  else
    $success = 0$ and $fail = fail + 1$;
    if $fail > \varphi_{fail}$ **then**
      $L = \max\left\{\frac{L}{2}, L_{\min}\right\}$ and $fail = 0$;
    end
  end
  Let $X_n = (X_n, \Xi_B)$, $\boldsymbol{Y}_n = (\boldsymbol{Y}_n, \boldsymbol{Y}_{new})$ and $\boldsymbol{v}_n = (\boldsymbol{v}_n, \boldsymbol{v}_{new})$;
  Use Algorithm 10 to allocate T replications among points in X_n;
  Update $\boldsymbol{Y}_n$ and $\boldsymbol{v}_n$, recalculate $\boldsymbol{Y}_n^{LCB}$ and let $G = G + T$;
end

Output : Return the best found solution $\mathbf{x}^* = \arg\max_{\mathbf{x} \in X_n} f_M^{LCB}(\mathbf{x})$

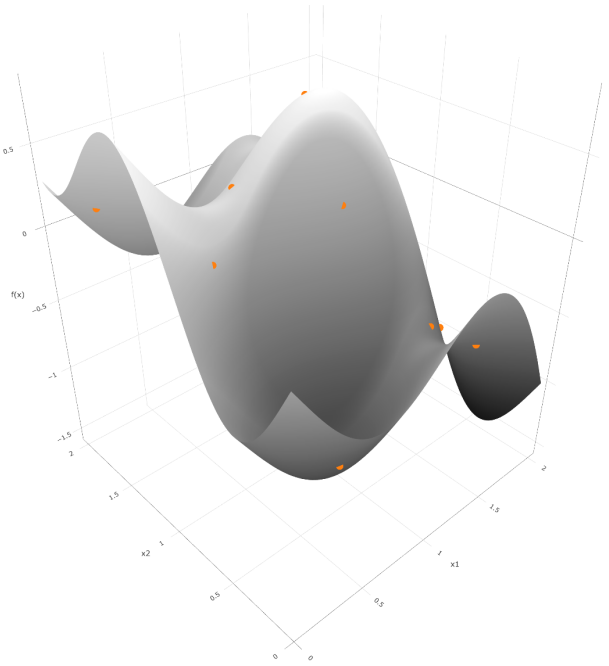

FIGURE 3.6: Michalewicz function in $x_1, x_2 \in (0, 2)$

3.5.4 Problems with Algorithm 11

While using trust regions can reduce the number of observations in the design, we still have to often deal with a large number of dimensions at the same time, which can still cause computational issues and inaccurate GP fits. The largest problem considered by TuRBO is the Ackley test function (Al-Roomi, 2015a) in 200 dimensions. However, the size of the problem in Bayesian DoE can be much larger than that, which TuRBO as well as Algorithm 11 will struggle with in practice.

One way to address those concerns is to increase $M_{\min}$ to make the computer model output more accurate, which makes it easier to fit an accurate GP. Another method to increase the accuracy of the surrogate model is to decrease the dimensionality. It is easier to fit an accurate GP in lower dimensions. An extreme example of that is the Approximate Coordinate Exchange (ACE) algorithm (Overstall and Woods, 2017), which is used for finding optimal designs for Bayesian Design of Experiments. The ACE algorithm fits a series of one-dimensional GPs. It does so by choosing the currently best observed solution and then only varying one dimension at a time, while keeping all others constant. That allows us to fit a GP in just a single dimension, which can be very accurate and fast. Consider Michalewicz function, which we introduced in Section 2.3.3.1. We have chosen an initial design of size $n = 10$ for $x_1, x_2 \in (0, 2)$, which is plotted in Figure 3.6.

To consider optimisation over a single dimension, we first choose the best observed point, which in this example is $(x_1, x_2) = (1.37, 1.42)$. We then treat one dimension as a constant and fix it, while optimising the function over the other dimension. For example,

when we consider optimisation along x_1, we fix $x_2 = 1.42$ and do the optimisation along the hyperplane, which is shown in Figure 3.7. If we plot this hyperplane in one dimension, then it looks like the graph in Figure 3.8.

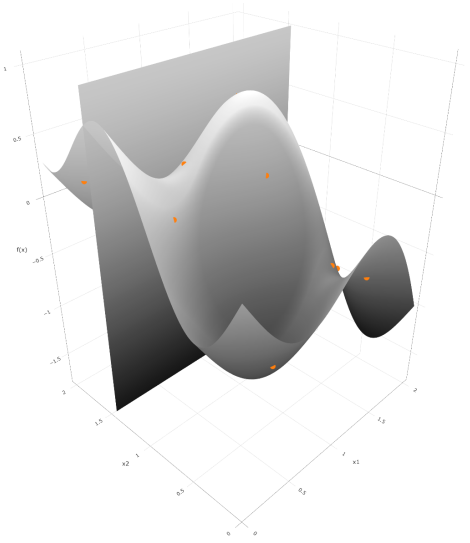

FIGURE 3.7: One dimensional optimisation over x_1 for the Michalewicz function

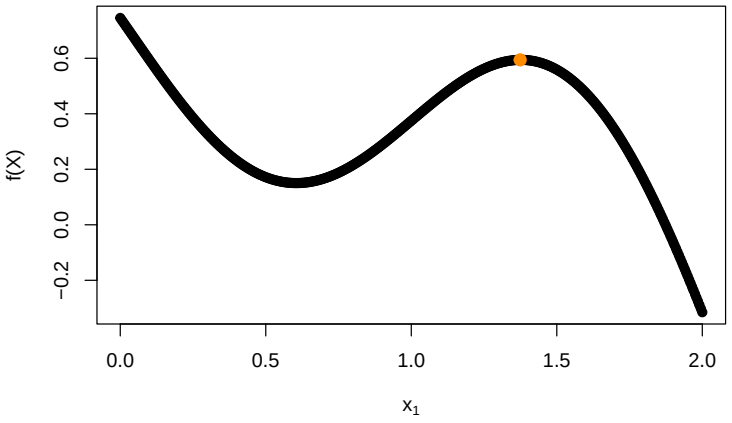

FIGURE 3.8: One dimensional cut of Michalewicz function over x_1

Similarly when we consider optimisation along x_2, we fix $x_1 = 1.37$ and only vary x_2. This means we only consider the hyperplane, which is shown in Figure 3.9. Plotting this hyperplane in one dimension can be seen in Figure 3.10.

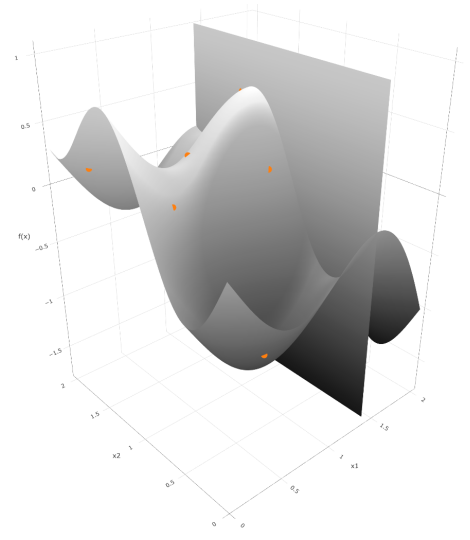

FIGURE 3.9: One dimensional optimisation over x_2 for the Michalewicz function

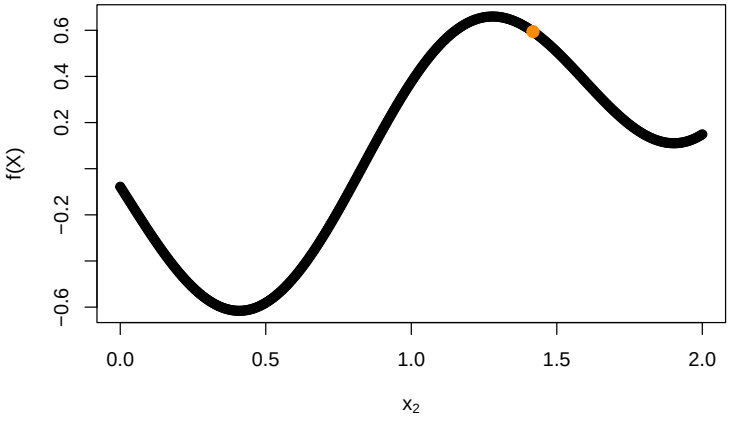

FIGURE 3.10: One dimensional cut of Michalewicz function over x_2

While this approach provides accurate results and is even guaranteed to converge to the global optimum under certain conditions (Luo and Tseng, 1992), it also has some limitations. In general, there are no guarantees that this method will converge to a global optimum instead of a local one, even after a large number of restarts. More importantly, it is also quite inefficient in terms of computer model evaluations. If we have a 200-dimensional problem, then we would have to create 200 different local models, each of which require an initial design. This causes the total number of computer model evaluations to be very large and make the entire algorithm inefficient. Instead we can fit p_0-dimensional GPs, where p_0 is smaller than the original model p, but is still larger than 1. This choice is very important, if p_0 is too large then we have the same problems as before. However, if p_0 is too small, then we are potentially making the algorithm more inefficient by optimising fewer numbers of dimensions at a time with the same computational budget. In our experiments we have found that starting out with $p_0 = 100$ is a good starting point.

The further we get, the more difficult it becomes to keep making progress. This would suggest that we both keep increasing $M_{\min}$ and decreasing p_0 throughout the run of the algorithm as we stop making progress by using the old values.

As we keep decreasing p_0, there comes a point where p_0 becomes smaller than some cut-off point p_0^* and our GP is no longer considered high-dimensional and we can start using standard Bayesian optimisation techniques. This means we no longer have to use trust regions and we can also use EI as an acquisition function, rather than Thompson sampling. In our experiments, we've found that choosing $p_0^* = 30$ provides good results.

In Equation 1.8, $y_{\max}$ was defined as the currently observed maximum from the computer model. However, this is only appropriate in the noiseless setting, where $f(\mathbf{x}) = \tilde{f}(\mathbf{x})$. Using this approach in our noisy setting could lead to a scenario where only a single very noisy observation would cause us to severely overestimate $y_{\max}$, which would also have a major impact on the shape of the EI function. Instead, we let $y_{\max} = m_n(\delta_C)$, where $\delta_C = \arg\max_{\mathbf{x} \in X_n} f_M^{LCB}(\mathbf{x})$ is the point with the highest observed LCB value. This is also known as the plugin method (Forrester, 2013). By using this, we can avoid having to estimate any further parameters that come with other noisy acquisition functions, such as the noise of any future observations (Huang et al. (2006a); Forrester (2013); Frazier et al. (2009)).

We now have enough to define our full two stage algorithm. We first run Algorithm 11 for a p_0 number of variables. After this algorithm converges, we move on to new variables that haven't yet been considered for optimisation and run it again. If we are unable to make any further progress, we reduce p_0 and increase $M_{\min}$. We keep doing this until $p_0 \leq p_0^*$. At that point we move on to stage two of the algorithm. In stage two, we fit a GP to p_0 variables, but this time without using trust regions. We then find a point that maximises EI for that GP and evaluate it. We then again move on to new

variables that haven't yet been considered for optimisation and repeat the same steps. If we are unable to make any further progress, we reduce p_0 and increase $M_{\min}$. These steps are described in Algorithm 12.

3.6 Simulations

For our test problem, we consider Bayesian Design of Experiments (DoE). In Bayesian DoE, we are designing an experiment where we are interested in the effect of d different variables on an outcome y. To model this effect, some sort of a statistical model is fitted (e.g. a linear regression model, logistic regression model, etc.). We are then interested in finding quantities $\boldsymbol{\beta} = (\beta_1, ..., \beta_k)$ from that model (such as model parameters, predicted

Algorithm 12: Partial Local Bayesian Optimisation Algorithm

Require: A computer model f, computational budget N, initial design matrix δ_0 of size $1 \times p$, the number of dimensions to be considered for local modelling at the start p_0, the minimum number of dimensions to use for trust regions p_0^* and the minimum number of replications for sampling a new point $M_{\min}$

Define : Let A_n be an $n \times p$ matrix, then if $S = (s_1, ..., s_{p_0})$ is a vector of column indexes, then $A_{n;S}$ is a subset of A_n, where only the columns in S are found and $A_{n;-S}$ is a subset of A_n, where only the columns not in S are found

Let $G = 0$;
Set $L = L_0$;
Set $\boldsymbol{W} = (w_1, ..., w_p) = \mathbf{1}_p = (1, ..., 1)$ be a vector of ones with length p;
while $G < N$ *and* $p_0 > p_0^*$ **do**
  Sample $S = (s_1, ..., s_{p_0})$ without replacement from $(1, ..., p)$ with weights $\boldsymbol{W}$;
  Let $w_{s_i} = 0$ for all $i \in \{1, ..., p_0\}$;
  Let X_n be a design of size $n \times p$ where $X_{n;S}$ is a new space-filling design of size $n \times p_0$ and $X_{n;-S}$ are the values from $\delta_{0;-S}$ replicated n times for each row;
  Evaluate f $M_{\min}$ times at each point in X_n and calculate $\boldsymbol{Y}_n = (\overline{f}_{M_{\min}}(\mathbf{x}_1), ..., \overline{f}_{M_{\min}}(\mathbf{x}_n))$, $\boldsymbol{v}_n = (\hat{v}(\mathbf{x}_1), ..., \hat{v}(\mathbf{x}_n))$ as well as $\boldsymbol{Y}_n^{LCB} = (f_{M_{\min}}^{LCB}(\mathbf{x}_1), ..., f_{M_{\min}}^{LCB}(\mathbf{x}_n))$;
  Let $y_{\max}^{old} = \max \boldsymbol{Y}_n^{LCB}$;
  Let $G = G + n \cdot M_{\min}$;
  Use columns S from X_n to run Algorithm 11 until convergence;
  Let $\delta_0 = \arg\max_{\mathbf{x} \in X_n} f_M^{LCB}(\mathbf{x})$ and $y_{\max}^{new} = \max_{\mathbf{x} \in X_n} f_M^{LCB}(\mathbf{x})$;
  if $p_0 > \sum_{i=1}^{p} w_i$ **then**
    Let $\boldsymbol{W} = \mathbf{1}_p$;
    if $y_{\max}^{new} \leq y_{\max}^{old}$ **then**
      Let $p_0 = \lceil \frac{p_0}{2} \rceil$ and $M_{\min} = \lceil 1.5 \cdot M_{\min} \rceil$;
    end
  end
end

Let $y_{\max}^{old} = \max \boldsymbol{Y}_n^{LCB}$;
while $G < N$ **do**
  Sample $S = (s_1, ..., s_{p_0})$ without replacement from $(1, ..., p)$ with weights $\boldsymbol{W}$;
  Let $w_{s_i} = 0$ for all $i \in \{1, ..., p_0\}$;
  Let X_n be a design of size $n \times p$ where $X_{n;S}$ is a new space-filling design of size $n \times p_0$ and $X_{n;-S}$ are the values from $\delta_{0;-S}$ replicated n times for each row;
  Evaluate f $M_{\min}$ times at each point in X_n and calculate $\boldsymbol{Y}_n = (\overline{f}_{M_{\min}}(\mathbf{x}_1), ..., \overline{f}_{M_{\min}}(\mathbf{x}_n))$, $\boldsymbol{v}_n = (\hat{v}(\mathbf{x}_1), ..., \hat{v}(\mathbf{x}_n))$ as well as $\boldsymbol{Y}_n^{LCB} = (f_{M_{\min}}^{LCB}(\mathbf{x}_1), ..., f_{M_{\min}}^{LCB}(\mathbf{x}_n))$;
  Let $y_{\max}^{new} = \max \boldsymbol{Y}_n^{LCB}$;
  Let $G = G + n \cdot M_{\min}$;
  Fit a GP by using $X_{n;S}$ and $\boldsymbol{Y}_n$;
  Find a point $\boldsymbol{\xi}_{new;S}$ that maximises EI for the fitted GP, evaluate $y_{new} = \overline{f}_{M_{\min}}(\boldsymbol{\xi}_{new;S})$ and calculate $y_{new}^{LCB} = f_{M_{\min}}^{LCB}(\boldsymbol{\xi}_{new;S})$;
  if $y_{new}^{LCB} > y_{\max}^{new}$ **then**
    $y_{\max}^{new} = y_{new}^{LCB}$;
  end
  if $p_0 > \sum_{i=1}^{p} w_i$ **then**
    Let $\boldsymbol{W} = \mathbf{1}_p$;
    if $y_{\max}^{new} < y_{\max}^{old}$ **then**
      Let $p_0 = \lceil \frac{p_0}{2} \rceil$ and $M_{\min} = \lceil 1.5 \cdot M_{\min} \rceil$;
    else
      $y_{\max}^{old} = y_{\max}^{new}$
    end
  end
end

Output : Return the best found solution $\mathbf{x}^* = \arg\max_{\mathbf{x} \in X_n} f_M^{LCB}(\mathbf{x})$

future responses, etc.) and we want those quantities to be as accurate as possible. Given a design $\chi \in \mathbb{R}^{n \times d}$ with n observations and d variables, collected data $\boldsymbol{y}$ and the quantities $\boldsymbol{\beta}$, we can measure the success of our experiment by using a utility function $u(\chi, \boldsymbol{y}, \boldsymbol{\beta})$.

Since the observed values $\boldsymbol{y}$ and $\boldsymbol{\beta}$ are unknown before running the experiment, one is interested in the expected utility instead, which is given as

$$\begin{aligned} U(\chi) =& E_{\boldsymbol{y},\boldsymbol{\beta}|\chi}[u(\chi, \boldsymbol{y}, \boldsymbol{\beta})] \\ =& \int u(\chi, \boldsymbol{y}, \boldsymbol{\beta}) \pi(\boldsymbol{y}, \boldsymbol{\beta}|\chi) d\boldsymbol{y} d\boldsymbol{\beta} \\ =& \int u(\chi, \boldsymbol{y}, \boldsymbol{\beta}) \pi(\boldsymbol{\beta}|\boldsymbol{y}, \chi) \pi(\boldsymbol{y}|\chi) d\boldsymbol{y} d\boldsymbol{\beta} \\ =& \int u(\chi, \boldsymbol{y}, \boldsymbol{\beta}) \pi(\boldsymbol{y}|\boldsymbol{\beta}, \chi) \pi(\boldsymbol{\beta}|\chi) d\boldsymbol{y} d\boldsymbol{\beta}, \end{aligned}$$

where $\pi(.|.)$ are the prior distributions for different variables.

The problem then becomes to maximise this expected utility function. Since these integrals are not usually available in closed form, Monte Carlo methods and other numerical methods are used to find an approximation $\tilde{U}(\chi)$ of the expected utility function instead. See Overstall and Woods (2017) and Overstall et al. (2020) for more information about Bayesian DoE and difficulties of evaluating the expected utility functions.

The resulting expected utility function is stochastic, high-dimensional and its accuracy can usually be controlled by increasing the computational cost of the function. Maximising this expected utility function is a very complex task with only a very limited number of algorithms being able to tackle it. A state of the art algorithm for finding optimal Bayesian designs for any generic utility functions, is the ACE algorithm (Overstall and Woods, 2017).

In the rest of this section we will demonstrate the performance of Algorithm 12 and compare it to ACE. We will use both algorithms to find a design that maximises Shannon Information Gain for a logistic regression model.

3.6.1 Shannon information gain

Shannon information gain is defined as

$$\begin{aligned} u^S(\chi, \boldsymbol{y}, \boldsymbol{\beta}) &= \log \pi(\boldsymbol{\beta}|\boldsymbol{y}, \chi) - \log \pi(\boldsymbol{\beta}|\chi) \\ &= \log \pi(\boldsymbol{y}|\boldsymbol{\beta}, \chi) - \log \pi(\boldsymbol{y}|\chi) \end{aligned}$$

An optimal design is then one that maximises $U^S(\chi, \boldsymbol{y}, \boldsymbol{\beta}) = E_{\boldsymbol{\beta},\boldsymbol{y}}[u^S(\chi, \boldsymbol{y}, \boldsymbol{\beta})]$. As mentioned earlier, this expectation is not available in closed form and has to be approximated.

A straightforward method is to first sample $\beta^B = (\boldsymbol{\beta}_1, ..., \boldsymbol{\beta}_B)^T$, where each $\boldsymbol{\beta}_i$ for $i = 1, ..., B$ is sampled from the prior $\pi(\boldsymbol{\beta}|\chi)$. Then using β^B, we can sample $y^B = (\boldsymbol{y}_1, ..., \boldsymbol{y}_B)$, where $\boldsymbol{y}_i$ is sampled from $\pi(\boldsymbol{y}|\boldsymbol{\beta}_i, \chi)$ for all $i = 1, ..., B$. With those samples, we can approximate $\pi(\boldsymbol{y}|\boldsymbol{\beta}, \chi)$ and $\pi(\boldsymbol{y}|\chi)$ as follows:

$$\begin{aligned} \tilde{\pi}(\boldsymbol{y}|\boldsymbol{\beta}, \chi) &= \frac{1}{B} \sum_{i=1}^{B} \pi(\boldsymbol{y}_i|\boldsymbol{\beta}_i, \chi) \\ \tilde{\pi}(\boldsymbol{y}|\chi) &= \frac{1}{B} \sum_{i=1}^{B} \frac{1}{B} \sum_{j=1}^{B} \pi(\boldsymbol{y}_i|\boldsymbol{\beta}_j, \chi) \end{aligned}$$

which means

$$\tilde{U}^S(\chi) = \log \tilde{\pi}(\boldsymbol{y}|\boldsymbol{\beta}, \chi) - \log \tilde{\pi}(\boldsymbol{y}|\chi).$$

This is the approximation used in Overstall and Woods (2017). Unfortunately, this approximation is biased for $\tilde{U}^S(\chi)$ and this bias has been shown to be of order B^{-1} (Ryan, 2003). Since our algorithm assumes that the computer model is unbiased, we are unable to use this approximation in our simulations. We therefore demonstrate how we can use power posteriors to better approximate $\tilde{U}^S(\chi)$. This follows the approach introduced by Friel and Pettitt (2008).

Power posterior is given as

$$\pi_t(\boldsymbol{\beta}|\boldsymbol{y},\chi) = \pi(\boldsymbol{y}|\boldsymbol{\beta},\chi)^t\pi(\boldsymbol{\beta}|\chi),$$

where $t \in [0,1]$ is a temperature variable.

We now define

$$z(\boldsymbol{\beta}|t) = \int_{\boldsymbol{\beta}} \pi(\boldsymbol{y}|\boldsymbol{\beta},\chi)^t\pi(\boldsymbol{\beta}|\chi)d\boldsymbol{\beta}.$$

We are interested in $\log\pi(\boldsymbol{y}|\chi)$, which can be shown to be

$$\log\pi(\boldsymbol{y}|\chi) = \log\left(\frac{z(\boldsymbol{\beta}|t=1)}{z(\boldsymbol{\beta}|t=0)}\right) = \int_0^1 E_{\boldsymbol{\beta}|\boldsymbol{y},t}[\log\pi(\boldsymbol{y}|\boldsymbol{\beta},\chi)]dt,$$

where

$$E_{\boldsymbol{\beta}|\boldsymbol{y},t}[\log\pi(\boldsymbol{y}|\boldsymbol{\beta},\chi)] = \int_{\boldsymbol{\beta}} \frac{\pi(\boldsymbol{y}|\boldsymbol{\beta},\chi)^t p(\boldsymbol{\beta}|\chi)}{z(\boldsymbol{y}|t)} \log\pi(\boldsymbol{y}|\boldsymbol{\beta},\chi)d\boldsymbol{\beta}$$

We can discretise this integral over $t \in [0,1]$ to approximate it. If we have $0 = t_0 < t_1 < ... < t_{A-1} < t_A = 1$, we get

$$\log\pi(\boldsymbol{y}|\chi) \approx \sum_{i=0}^{A-1}(t_{i+1}-t_i)\frac{E_{\boldsymbol{\beta}|\boldsymbol{y},t_{i+1}}[\log\pi(\boldsymbol{y}|\boldsymbol{\beta})] + E_{\boldsymbol{\beta}|\boldsymbol{y},t_i}[\log\pi(\boldsymbol{y}|\boldsymbol{\beta})]}{2}$$

To approximate $E_{\boldsymbol{\beta}|\boldsymbol{y},t_i}[\log\pi(\boldsymbol{y}|\boldsymbol{\beta})]$, we first sample B points $(\boldsymbol{\beta}_1,...,\boldsymbol{\beta}_B)$ from $\pi_{t_i}(\boldsymbol{\beta}|\boldsymbol{y},\chi)$ by using MCMC sampling and then simply having

$$E_{\boldsymbol{\beta}|\boldsymbol{y},t_i}[\log\pi(\boldsymbol{y}|\boldsymbol{\beta})] \approx \frac{1}{N}\sum_{j=1}^{N}\log\pi(\boldsymbol{y}|\boldsymbol{\beta}_i,\chi)$$

3.6.2 Shannon information gain for a logistic regression model

Consider a logistic regression model with d explanatory variables. This means that $y_i \sim \text{Bernoulli}(p_i)$, where

$$\log\left(\frac{p_i}{1-p_i}\right) = \beta_0 + \sum_{j=1}^{d}\beta_j x_{ij}$$

for $i = 1, ..., n$. This can then be rewritten for p_i as

$$p_i = \frac{1}{1 + \exp(-\beta_0 - \sum_{j=1}^{d} \beta_j \chi_{ij})}$$

For logistic regression, the data likelihood $p(\boldsymbol{y}|\boldsymbol{\beta}, \chi)$ is given as

$$p(\boldsymbol{y}|\boldsymbol{\beta}, \chi) = \prod_{i=1}^{n} p_i^{y_i} (1 - p_i)^{1-y_i}$$

To ensure computational stability, we often find it useful to also define the log-likelihood function, which is given as

$$\begin{aligned}
\log p(\boldsymbol{y}|\boldsymbol{\beta}, \chi) &= \log \left(\prod_{i=1}^{n} p_i^{y_i} (1 - p_i)^{1-y_i} \right) \\
&= \sum_{i=1}^{n} y_i \log p_i + (1 - y_i) \log(1 - p_i) \\
&= \sum_{i=1}^{n} \log(1 - p_i) + \sum_{i=1}^{n} y_i \log \frac{p_i}{1 - p_i} \\
&= \sum_{i=1}^{n} \log \left(1 - \frac{1}{1 + \exp(-\beta_0 - \sum_{j=1}^{d} \beta_j \chi_{ij})} \right) + \sum_{i=1}^{n} \left(y_i \beta_0 + \sum_{j=1}^{d} y_i \beta_j x_{ij} \right)
\end{aligned}$$

Following Overstall and Woods (2017), we will restrict our design to $x_i \in [-1, 1]$ for $i = 1, ..., d$ and choose $d = 4$. We specify the following prior distributions for $\pi(\boldsymbol{\beta}|\chi)$:

$$\begin{aligned}
&\beta_0 \sim U[-3, 3] = \pi(\beta_0|\chi), \qquad \beta_1 \sim U[4, 10] = \pi(\beta_1|\chi), \qquad \beta_2 \sim U[5, 11] = \pi(\beta_2|\chi), \\
&\beta_3 \sim U[-6, 0] = \pi(\beta_3|\chi), \qquad \beta_4 \sim U[-2.5, 3.5] = \pi(\beta_4|\chi).
\end{aligned} \tag{3.1}$$

We'll also define $\pi(\boldsymbol{\beta}|\chi) = \prod_{i=0}^{4} \pi(\beta_i|\chi)$.

To now approximate $\tilde{U}^S(\chi)$, we can use Algorithm 13.

Algorithm 13: Approximating Shannon information gain for a logistic regression model

Require: A design matrix χ, a number of outer samples B_1, number of inner samples B_2, a set of A temperatures $0 = t_0 < t_1 < ... < t_{A-1} < t_A = 1$, step size for the Metropolis Hastings algorithm $\boldsymbol{\sigma}^2$

for $b = 1, ..., B_1$ **do**

Draw a random sample $\boldsymbol{\beta}^{(b)} = \left(\beta_0^{(b)}, ..., \beta_4^{(b)}\right)$, where $\beta_i^{(b)} \sim \pi(\beta_i|\chi)$ for all $i = 0, ..., 4$

Calculate $\boldsymbol{p}^{(b)} = (p_1^{(b)}, ..., p_n^{(b)})$, where $p_i^{(b)} = \frac{1}{1+\exp(-\beta_0^{(b)} - \sum_{j=1}^{4} \beta_j^{(b)} \chi_{ij})}$ for all $i = 1, ..., n$

Draw a sample $\boldsymbol{y}^{(b)} = (y_1^{(b)}, ..., y_n^{(b)})$, where $y_i^{(b)} \sim \text{Bernoulli}(p_i^{(b)})$

Calculate $\pi(\boldsymbol{y}^{(b)}|\boldsymbol{\beta}^{(b)}, \chi)$

Let $\boldsymbol{\beta}_0^{(0)} = (E[\pi(\boldsymbol{\beta}_0|\chi)], ..., E[\pi(\boldsymbol{\beta}_4|\chi)]) = (0, 7, 8, -3, 0.5)$

for $a = 1, ..., A$ **do**

for $i = 1, ..., B_2$ **do**

$\boldsymbol{\beta}_i^{(0)} \sim N(\boldsymbol{\beta}_{i-1}^{(0)}, \boldsymbol{\sigma}^2)$

Calculate

$$Accept = t_a \cdot \log p\left(\boldsymbol{y}^{(b)}|\boldsymbol{\beta}_i^{(0)}, \chi\right) + \log \pi\left(\boldsymbol{\beta}_i^{(0)}|\chi\right) - t_a \cdot \log p\left(\boldsymbol{y}^{(b)}|\boldsymbol{\beta}_{i-1}^{(0)}, \chi\right) - \log \pi\left(\boldsymbol{\beta}_{i-1}^{(0)}|\chi\right) \tag{3.2}$$

Draw $p_{accept} \sim U[0, 1]$

if $\log p_{accept} > Accept$ **then**

$\boldsymbol{\beta}_i^{(0)} = \boldsymbol{\beta}_{i-1}^{(0)}$

end

end

Let $\hat{E}_{\boldsymbol{\beta}|\boldsymbol{y}^{(b)}, t_a}[\log \pi(\boldsymbol{y}|\boldsymbol{\beta})] = \frac{1}{B} \sum_{i=1}^{B} \log p(\boldsymbol{y}^{(b)}|\boldsymbol{\beta}_i^{(0)}, \chi)$

Let $\boldsymbol{\beta}_0^{(0)} = \boldsymbol{\beta}_B^{(0)}$

end

$$log\hat{\pi}(\boldsymbol{y}^{(b)}|\chi) = \sum_{a=0}^{A-1} (t_{a+1} - t_a) \frac{\hat{E}_{\boldsymbol{\beta}|\boldsymbol{y}^{(b)}, t_{a+1}}[\log \pi(\boldsymbol{y}|\boldsymbol{\beta})] + \hat{E}_{\boldsymbol{\beta}|\boldsymbol{y}^{(b)}, t_a}[\log \pi(\boldsymbol{y}|\boldsymbol{\beta})]}{2}$$

end

Calculate

$$\hat{U}^S(\chi) = \frac{1}{B} \sum_{i=1}^{B} \log \pi(\boldsymbol{y}^{(b)}|\boldsymbol{\beta}^{(b)}, \chi) - \frac{1}{B} \sum_{i=1}^{B} log\hat{\pi}(\boldsymbol{y}^{(b)}|\chi).$$

Output : $\hat{U}^S(\chi)$

The computer model we get from Algorithm 13 is unbiased, but subject to Monte Carlo error, which means it is noisy. However, we can control the noise level and make the output more accurate by increasing B_1, which is equivalent to evaluating the computer model at the same input location multiple times. The implementation of Algorithms 12 and 13 are both found on Github at `https://github.com/Hendriico/PLBO`.

For our experiments, we aim to find an optimal Bayesian design χ of size 150×4, such that its Shannon Information gain is maximised. To find an optimal Bayesian design means to choose every element of the matrix

$$\chi = \begin{pmatrix} \chi_{1,1} & \chi_{1,2} & \chi_{1,3} & \chi_{1,4} \\ \chi_{2,1} & \chi_{2,2} & \chi_{2,3} & \chi_{2,4} \\ \dots & \dots & \dots & \dots \\ \chi_{150,1} & \chi_{150,2} & \chi_{150,3} & \chi_{150,4} \end{pmatrix}.$$

This means that we need to choose $150 \cdot 4 = 600$ different values and hence the underlying optimisation problem will be 600-dimensional. We will maximise this computer model by using our proposed Partial Local Bayesian Optimisation (PLBO) algorithm as well as the ACE algorithm. We compare the two algorithms by controlling the number of MCMC iterations available to each algorithm. We run each algorithm 10 times with different starting locations. The results can be seen in Figure 3.11.

As seen in Figure 3.11, our proposed PLBO algorithm manages to outperform the ACE algorithm in terms of both speed as well as the quality of the final result. This can be further emphasised by zooming in on the top part of the graph, which can be seen in Figure 3.12.

From Figure 3.12 it is clear to see that PLBO converges to an optimum 50%-100% faster than the ACE algorithm, while also finding better optimal designs for every single run. This shows that the extra efficiency we gained from fitting GPs to more than a single dimension at a time allowed us to find better results faster.

3.7 Discussion

In this chapter we presented a new algorithm for multi-fidelity Bayesian optimisation in high-dimensional settings. This addresses a significant gap in the existing literature where the simultaneous challenges of high dimensionality and computer model evaluation costs are often treated separately or inadequately. The proposed PLBO algorithm integrates trust regions and OCBA with a local subspace-based optimisation strategy, enabling scalable and efficient search in scenarios where full-fidelity evaluations are expensive and the input space is of high dimension.

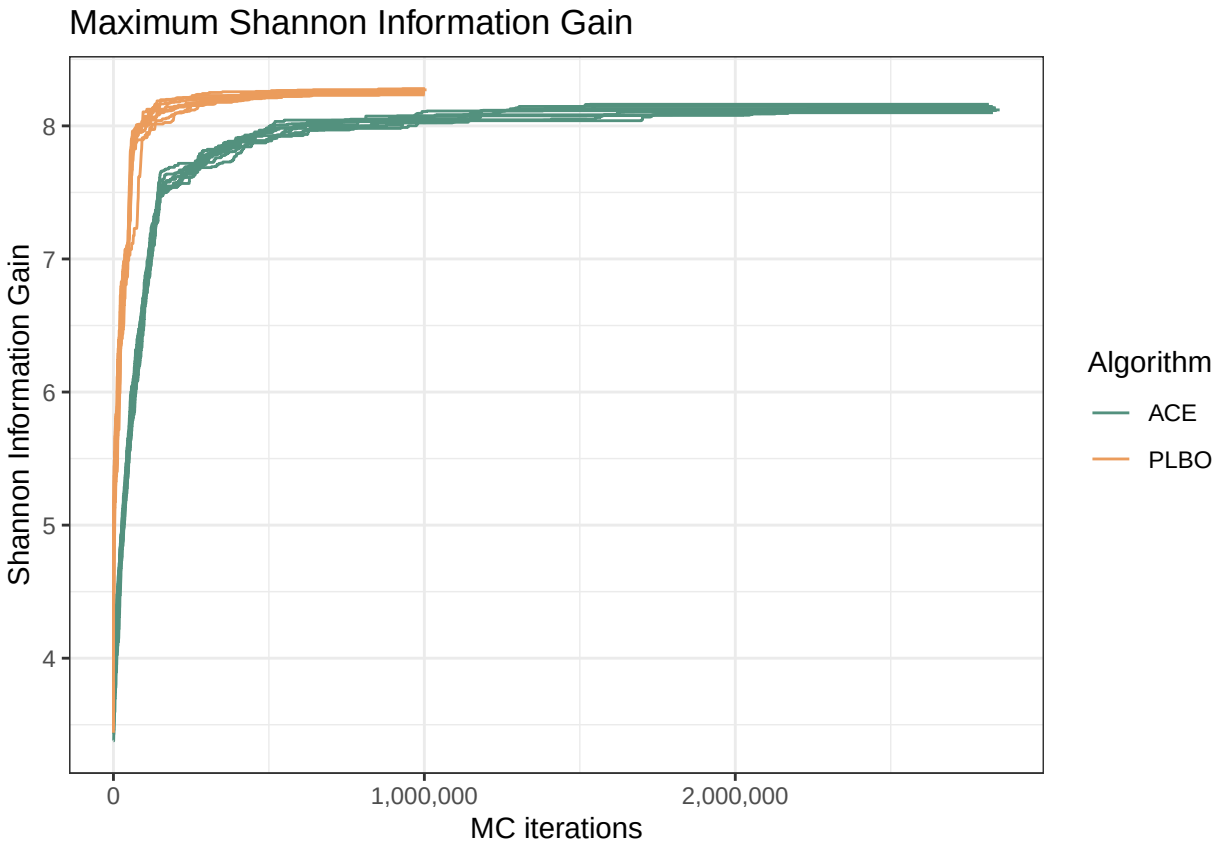

FIGURE 3.11: Comparison of ACE and PLBO algorithms for Shannon Information Gain

Our algorithm's performance was demonstrated by successfully finding optimal Bayesian designs. We were interested in finding a D-optimal design of size 150×4 for a logistic regression model. This means that the test problem was 600-dimensional. We compared our algorithm against an existing state-of-the-art algorithm called ACE. We kept the total number of computer model evaluations equal for both algorithms to more accurately compare their performance. We found that PLBO found better results while only requiring about a third of the number of evaluations compared to the ACE algorithm. This demonstrated our algorithm's superior performance.

Despite its success, the proposed method has certain limitations that suggest directions for future work. The fitting of Gaussian Processes at the start of the algorithm is still slightly limiting due to lack of GP implementations that can be fitted quickly and accurately for moderate to large number of dimensions. The algorithm itself is also fairly complicated and further simplification of it could be possible, but this is outside of scope for this current work.

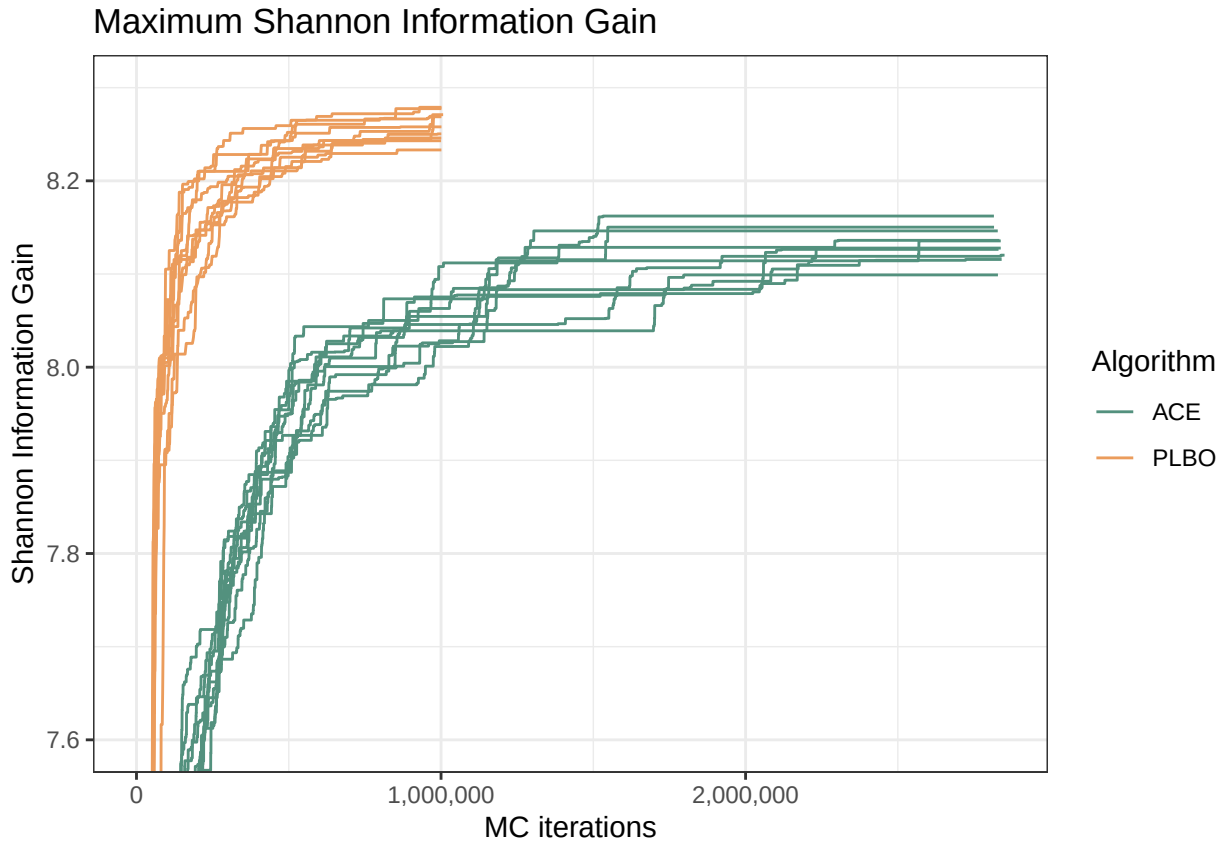

FIGURE 3.12: Comparison of ACE and PLBO algorithms for Shannon Information Gain (zoomed in to better visualise the differences between the results of the two algorithms)

Chapter 4

Sequential Design for Non-Stationary Computer Models

4.1 Introduction

Modern scientific and engineering investigations frequently rely on computationally intensive simulators to model complex physical systems. In many such cases, surrogate modelling is used to cheaply approximate these simulators. Gaussian Processes are are one of the most popular surrogate models due to their flexibility and their ability to both provide predictions as well as quantify uncertainty. When combined with sequential design strategies, GPs enable efficient exploration of expensive simulators by iteratively selecting new input locations that improve the surrogate model in targeted ways.

A standard Gaussian Process, which we have been using throughout this paper, assumes stationarity, meaning that its properties do not change throughout the design space. However, often times, this assumption does not hold true in real-world settings. In these scenarios, stationary GPs can perform very poorly, underfitting regions of high-variance, while over-exploring areas of low-variance.

To remedy the situation, non-stationary GPs along with appropriate acquisition functions have been proposed. While offering greater flexibility and more accurate predictions, they are computationally very expensive to fit. They usually also require a large number of evaluated points to learn about the underlying structure and fit accurately. These reasons mean that sequential design strategies are extremely inefficient when combined with non-stationary GPs, potentially defeating the purpose of sequential design altogether. As a result, we are interested in a method that can adapt to non-stationary behaviour using simpler stationary GP models.

In this chapter, we present a novel acquisition function designed specifically for finding sequential designs for non-stationary computer models that avoids having to fit non-stationary GPs at every iteration. The acquisition function intelligently finds areas of greater interest and allocates more computational resources into exploring those regions.

In section 4.1 we give a brief overview of existing non-stationary GPs and their relation to sequential design process. Section 4.3 a few acquisition functions that are used in sequential design. It shows how they are calculated and demonstrates some of the weaknesses that arise from using them with non-stationary computer models. In section 4.4 we define our new acquisition function for a 1-dimensional GP. We use pairwise slopes between all the points to measure how quickly the function changes at any given location. Regions with a greater rate of change are given more emphasis and areas where function changes less rapidly are given less emphasis. In section 4.4.2 we extend this acquisition function to multi-dimensional problems. Finally we perform our simulation studies in section 4.5. We compare our new approach against a few other existing methods, the most advanced method being a fully non-stationary sequential design using deep GPs. We demonstrate how our new method either achieves similar results to the deep GPs or even outperforms them. All while using only a fraction of the computational resources compared to the non-stationary GP.

4.2 Non-Stationary Computer Models

Despite being a useful tool, GPs assume the computer model is stationary, which limits its ability to model many real-life processes accurately. The stationarity assumption is mostly there for computational convenience, making the likelihood calculations and uncertainty quantification be available in closed form.

Stationary GPs assume that the correlation function only depends on $\boldsymbol{\theta}$ and pairwise distances between points. It does not take into account the fact that the computer model could be changing much more rapidly in certain areas compared to others. Common approaches to modelling non-stationarity are given in the next section.

4.2.1 Non-Stationary Gaussian Processes

Many surrogates have been introduced in the past for non-stationary computer models, such as Schmidt and O'Hagan (2003), Paciorek and Schervish (2003) and Rasmussen and Ghahramani (2001). These mostly focused on geospatial modelling, which meant that the input dimensions were very low and the amount of training data was also small.

More recently, Bornn et al. (2011) proposed an algorithm based on dimension expansion. A small number of dimensions is expanded into a higher number of dimensions

and transformed so that the resulting process is again stationary. Katzfuss (2013) also focused on modelling spatial datasets, but focused on scenarios with a large number of available datapoints.

Treed Gaussian processes were proposed by Gramacy and Lee (2009) to fit to models where the stationarity changes alongside each coordinate direction. It combines treed partitioning with Gaussian processes. The design space is split into different regions and within each region, a separate GP is fitted.

Most recently, a lot of attention has been given to deep GPs. Deep GPs are multi-layered models, where every layer has a multivariate normal distribution. These hidden layers can warp the design in ways so that some points are pushed closed together and some are spread further away, which allows us to use stationary correlation functions. Deep GPs were first proposed by Damianou and Lawrence (2013) and later expanded on by many authors, such as Bui et al. (2016), Salimbeni and Deisenroth (2017), Havasi et al. (2018). Most of the work carried out has been in the context of inference and fitting deep GPs. Only a few works have focused on using deep GPs for sequential design. For example, Rajaram et al. (2021) used a simple criterion with deep GPs, that chose a point with the highest posterior variance to be added to the design. A more complex acquisition function was given by Sauer et al. (2021), who introduced an acquisition function for deep GPs that measures how much variance can be reduced over the entire design, by adding a new point. This acquisition function will be introduced in the next section.

Sequential design using non-stationary computer models requires us to refit the surrogate model at every iteration, which can be computationally very expensive. Non-stationary computer models also often need a moderately large amount of training data to learn about accurately about the structure of the problem (Sauer et al., 2023). Because of those reasons, we might be interested in how we can create sequential designs for non-stationary computer models without being affected by the drawbacks of non-stationary GPs. We propose a novel acquisition function that can be used with regular stationary GPs and will find designs for non-stationary GPs by putting more points in the more interesting regions.

4.3 Acquisition Functions

Often times our goal is to find a design that maximises the accuracy of the fitted model or in other words, minimises the mean squared error (MSE) of the model. In the context of GPs, this is equivalent to reducing the uncertainty/variance of the GP. The simplest method to do that sequentially, is to simply choose the point with the highest posterior variance (MacKay, 1992). This is also known as ALM (Active Learning MacKay), named

after Mackay, who first proposed using posterior variance as an acquisition function in sequential design. More formally, we choose a new point $\mathbf{x}_{n+1}$, so that

$$\mathbf{x}_{n+1} = \arg\max_{\mathbf{x}\in\mathbb{X}} \hat{\Sigma}_n(\mathbf{x}).$$

However, this ignores the fact that gaining information at a specific location also reduces uncertainty in other locations and we are interested in minimising uncertainty over the entire design space. After adding a new point $\mathbf{x}_{n+1}$ to the design, variance of the Gaussian process at point $\mathbf{x}$ becomes

$$\hat{\Sigma}_{n+1}(\mathbf{x}) = \hat{\Sigma}_n(\mathbf{x}) + \hat{\sigma}^2(\hat{\Sigma}_0(\mathbf{x}, \mathbf{x}_{n+1})\hat{\Sigma}_0^{-1}(\mathbf{x}_{n+1}, \mathbf{x}_{n+1})\hat{\Sigma}_0(\mathbf{x}_{n+1}, \mathbf{x})),$$

where $\hat{\Sigma}_0(X_1, X_2) = \hat{k}_n(X_1, X_2) - \hat{k}_n(X_1, X_n)\hat{K}_n^{-1}(X_n, X_n)\hat{k}_n(X_n, X_2)$ and X_n is the current design. We are interested in how much we can reduce the variance, which means we are interested in the measure $\hat{\Sigma}_n(\mathbf{x}) - \hat{\Sigma}_{n+1}(\mathbf{x})$. To measure how much the variance reduces over the entire design space, we simply integrate over all possible values of $\mathbf{x} \in \mathbb{X}$. So we get

$$\begin{aligned}
\Delta\Sigma_n(\mathbf{x}_{n+1}) &= \int_{\mathbf{x}\in\mathbb{X}} \hat{\Sigma}_n(\mathbf{x}) - \hat{\Sigma}_{n+1}(\mathbf{x}) dx \qquad (4.1)\\
&= \int_{\mathbf{x}\in\mathbb{X}} \hat{\Sigma}_n(\mathbf{x}) - \left(\hat{\Sigma}_n(\mathbf{x}) + \hat{\sigma}^2(\hat{\Sigma}_0(\mathbf{x}, \mathbf{x}_{n+1})\hat{\Sigma}_0^{-1}(\mathbf{x}_{n+1}, \mathbf{x}_{n+1})\hat{\Sigma}_0(\mathbf{x}_{n+1}, \mathbf{x}))\right) dx\\
&= \int_{\mathbf{x}\in\mathbb{X}} \hat{\sigma}^2(\hat{\Sigma}_0(\mathbf{x}, \mathbf{x}_{n+1})\hat{\Sigma}_0^{-1}(\mathbf{x}_{n+1}, \mathbf{x}_{n+1})\hat{\Sigma}_0(\mathbf{x}_{n+1}, \mathbf{x}))dx
\end{aligned}$$

The goal is to minimise variance, which means we want to maximise $\Delta\Sigma_n(\mathbf{x}_{n+1})$. More formally, at step n, we want to add point $\mathbf{x}_{n+1}$ to the design, so that

$$\mathbf{x}_{n+1} = \arg\max_{\mathbf{x}\in\mathbb{X}} \Delta\Sigma_n(\mathbf{x}).$$

While the integral in Equation 4.1 is not available in closed form in general, it can be calculated analytically when $\mathbb{X}$ is a hyper-rectangle. Despite this, numeric optimisation of this function can still be a challenge (Sauer et al., 2021).

Because of this, more commonly variance reduction over a reference set X_{ref} is measured. This is also known as ALC (Active Learning Cohn), named after Cohn, who proposed it (Cohn, 1996) and it is given as:

$$
\begin{aligned}
ALC(\mathbf{x}_{n+1}|X_{\text{ref}}) &\propto \sum_{\mathbf{x}\in X_{\text{ref}}} \hat{\Sigma}_n(\mathbf{x}) - \hat{\Sigma}_{n+1}(\mathbf{x}) \\
&= \sum_{\mathbf{x}\in X_{\text{ref}}} \hat{\sigma}^2(\hat{\Sigma}_0(\mathbf{x}, \mathbf{x}_{n+1})\hat{\Sigma}_0^{-1}(\mathbf{x}_{n+1}, \mathbf{x}_{n+1})\hat{\Sigma}_0(\mathbf{x}_{n+1}, \mathbf{x}))
\end{aligned}
$$

The optimisation is then also carried out by calculating ALC for a discrete set of candidate points X_{cand} (this can be the same as X_{ref}) and choosing the point with the highest ALC value. In other words, we choose

$$
\mathbf{x}_{n+1} = \arg\max_{\mathbf{x}\in X_{\text{cand}}} ALC(\mathbf{x}|X_{\text{ref}}).
$$

Discretising the integral and the optimisation problems allows for simpler and more stable implementation as well as allowing us to easily parallelise the computer code.

To quickly demonstrate how ALC works in practice, consider the following computer model over $x \in [0, 1]$:

$$
f_H(x) = \begin{cases} 1.35cos(24\pi x), & \text{if } x \leq 0.33 \\ 1.35, & \text{if } 0.33 < x \leq 0.66 \\ 1.35cos(12\pi x), & \text{if } x > 0.66 \end{cases} \tag{4.2}
$$

We choose a simple design of size $n = 15$ and evaluate the computer model there. We then fit a Gaussian Process to that data. This can be seen in Figure 4.1 (a).

We then choose our $\boldsymbol{X}_{\text{ref}}$ to be a set of 100 uniformly distributed points and also set $X_{\text{cand}} = X_{\text{ref}}$. ALC values can then be seen in Figure 4.1 (b).

As we can see, ALC is higher in areas where adding a new point will lead to greater reduction in GP posterior variance. However, in this example, our computer model is highly non-stationary. Even though the computer model is completely flat between $x \in [0.33, 0.66]$, there are still many points in that region, where the ALC value is high. In reality, evaluating a point there will not give as much more information. Likewise, ALC is also high around $x \approx 0.85$, but evaluating the function there will not give as much information as some other points, since the computer model does not change very quickly and the predictive mean is already quite accurate. Instead we would want to focus more on $x \in [0, 0.33]$, where the computer model changes more rapidly and we can gain more information about the behavior of the computer model. One way to fix that, is to fit a non-stationary surrogate model to the data. However, fitting a new model at every stage in our sequential design algorithm can be very expensive and make the design process inefficient. Non-stationary computer models also often require moderately large

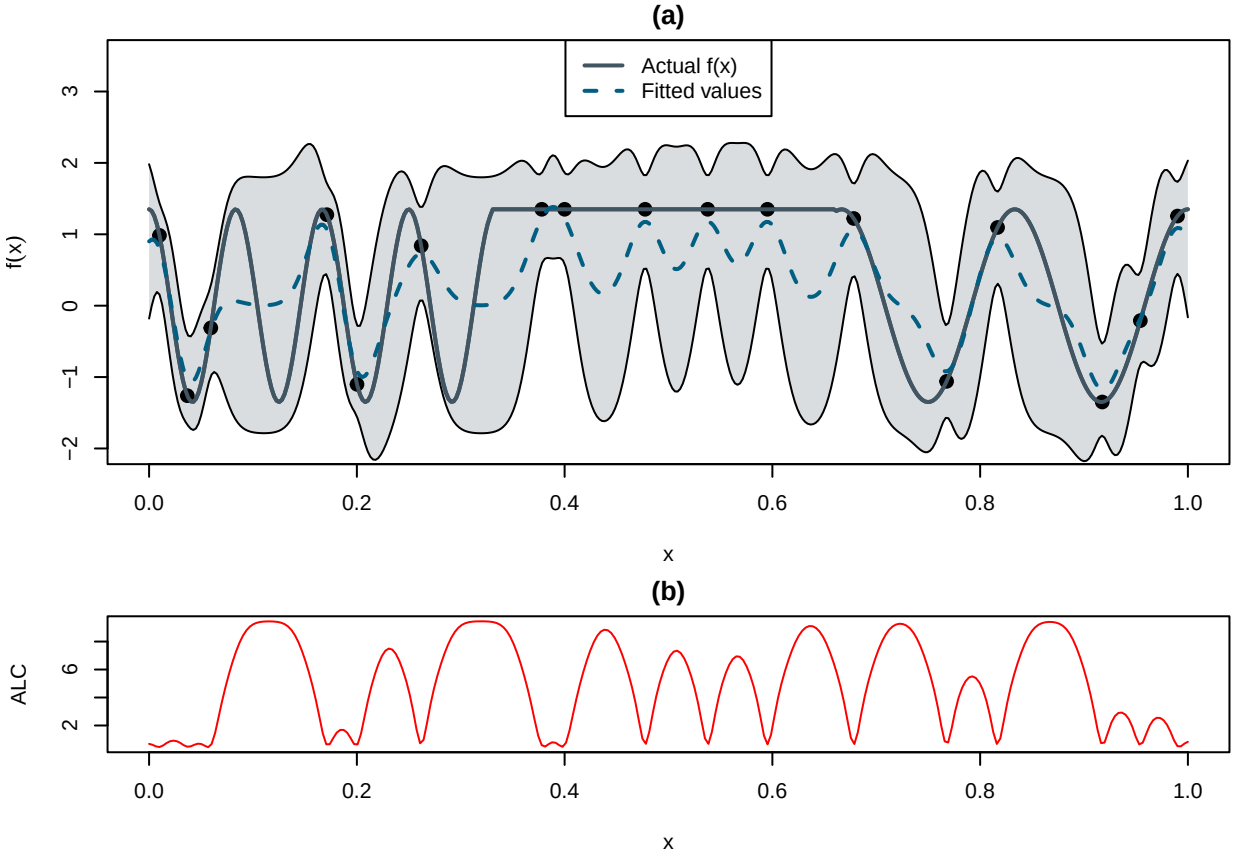

FIGURE 4.1: (a) An example of a fitted GP to a computer model $f_H(x)$ (b) ALC values from the GP

designs to be able to learn about the structure of the computer model accurately. This is shown in Figure 4.2 (a), where we fit a deep GP to the same design as we did in Figure 4.1.

In Figure 4.2 we can see that the deep GP is unable to accurately fit the data with the amount of training data available to it. This will also limit the performance of any sequential design strategies judging from the shape of the ALC function in Figure 4.2 (b).

In the next section, we demonstrate how we can create sequential designs for non-stationary computer models, by only fitting stationary GPs.

4.4 Stationarity Score

If our computer model is non-stationary, then fitting stationary GPs means that we are unable to learn about the structure of the computer model accurately. This means that ALC will put equal weights to the "interesting" regions as well as the regions that are

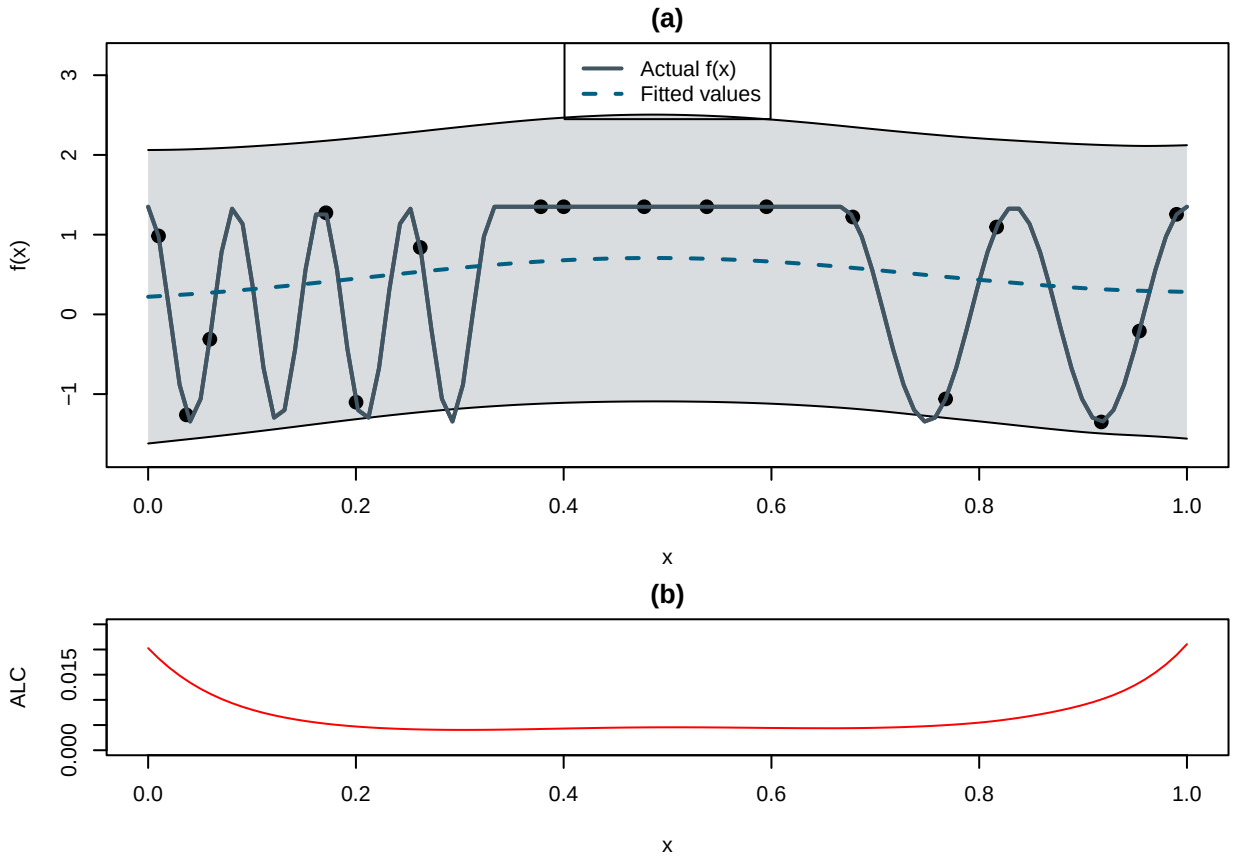

FIGURE 4.2: (a) An example of a fitted deep GP to a computer model $f_H(x)$ (b) ALC values from the deep GP

completely flat. Ideally we would like to put more weight to the more interesting regions where the function changes more rapidly.

This could be modelled with non-stationary GPs. However, fitting non-stationary GPs accurately can be quite a time consuming task and as seen in Figure 4.2, a fairly large design is usually needed to fit the model accurately. In this section we demonstrate how we can use stationary GPs and extend ALC to put more weight on the more interesting regions.

4.4.1 Measuring lengthscale

In a non-stationary Gaussian Process, the lengthscale variable is able to change within the design space. This lengthscale variable θ is used in correlation function $r(x_1, x_2) = \exp\left(-\frac{(x_1-x_2)^2}{\theta}\right)$ and it controls the wiggliness of a GP.

Lengthscale is related to the number of level-u upcrossings N_u. N_u is the number of times a GP crosses level u from left to right on the y-axis. It can be shown that for a Gaussian kernel $E[N_u] \propto \theta^{-\frac{1}{2}}$ (Adler and Taylor, 2007).

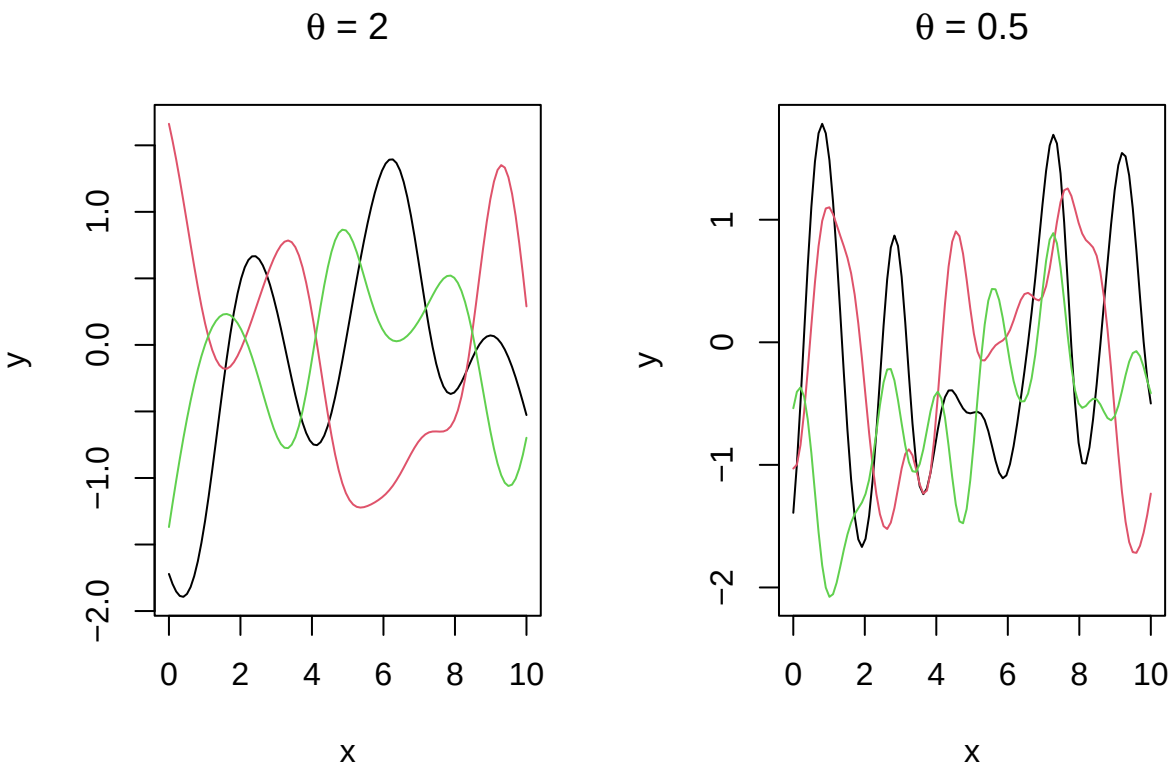

FIGURE 4.3: Random draws from two different GP priors

To demonstrate this, let us first sample a number of random draws from a GP prior with $\theta = 2$ and $\theta = 0.5$. The paths drawn from a GP prior with $\theta = 0.5$ should be twice as wiggly and have twice as many upcrossings as the ones from a GP prior with $\theta = 2$.

The number of upcrossings is directly related to how often a function changes directions, which is related to how much the slope of the function changes. Therefore to understand how wiggly the function is at any point x we want to know how often the slope changes in the vicinity of x.

For example, consider again the function given in Equation 4.2. It can be seen in Figure 4.4. It is clearly non-stationary. It is very wiggly between $x \in [0, 0.33]$, half as wiggly between $x \in (0.66, 1]$ and completely flat between $x \in (0.33, 0.66]$. We can choose a random space filling design of size 40 to evaluate the function at. This is used to fit a GP and find the maximum likelihood estimate of θ. This design is also shown in Figure 4.4.

Intuitively, where the function is very wiggly, then the local derivatives change very quickly and when the function is smooth, then the derivative doesn't change much at all.

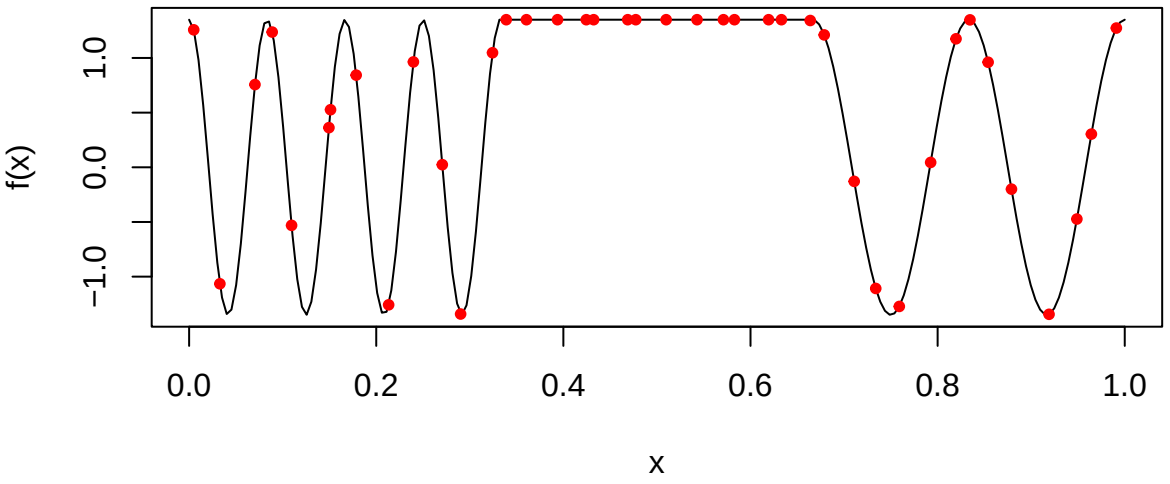

FIGURE 4.4: Example of a non-stationary computer model

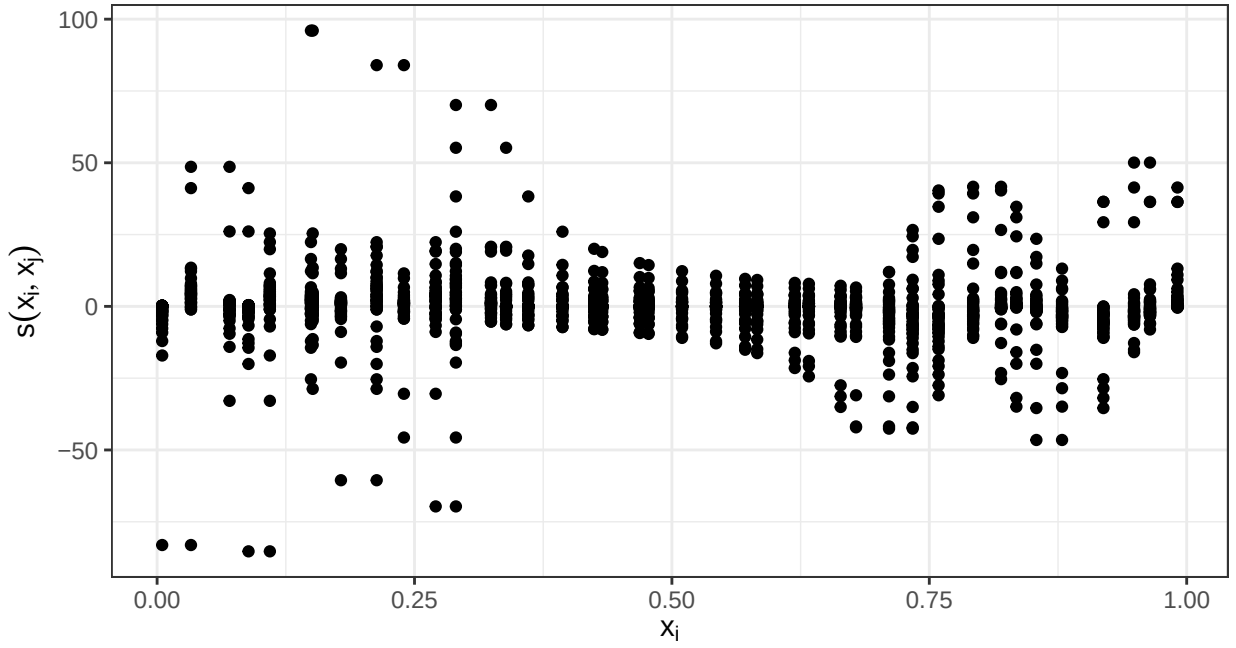

FIGURE 4.5: All pairwise slopes between different points

To get an idea of the smoothness, we calculate the pairwise slopes between all points. Given two points x_i and x_j and their respective function values $y_i = f(x_i)$ and $y_j = f(x_j)$, the slope between them is given by $s(x_i, x_j) = \frac{y_i - y_j}{x_i - x_j}$.

All pairwise slopes for the example in Figure 4.4 can be seen in Figure 4.5.

To measure how much the slopes change, we can use variance. Since we want the

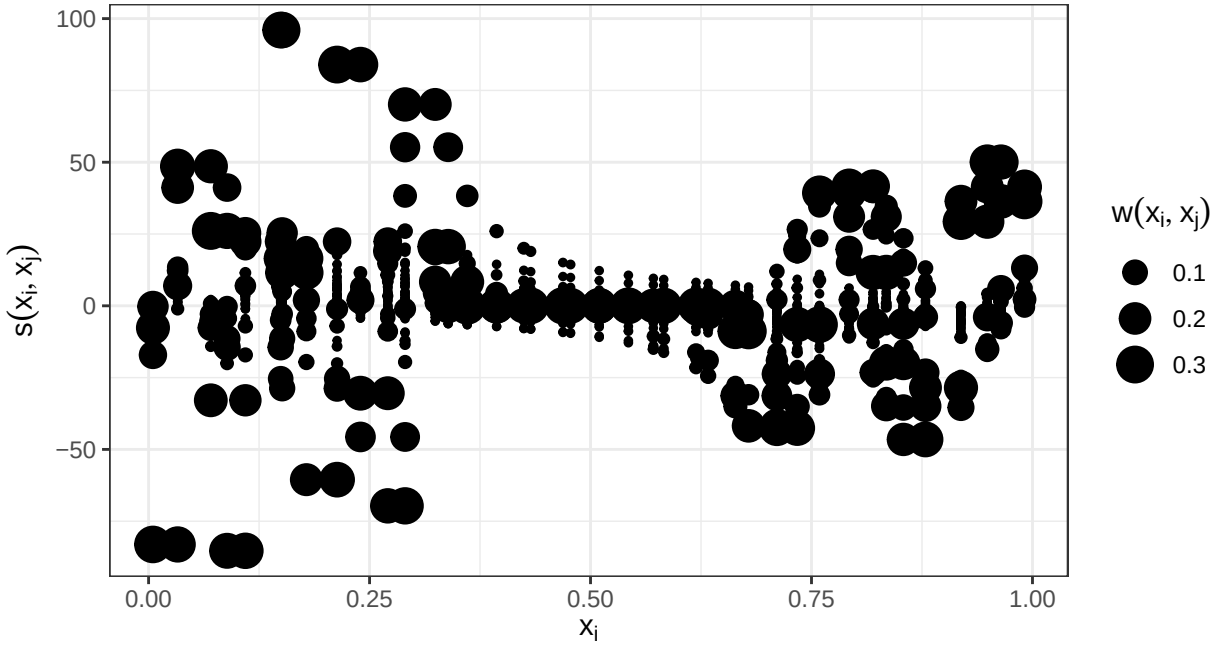

FIGURE 4.6: All pairwise slopes between points weighted by pairwise distances

points closer to x_i to have more of an effect on the variance, we will use weighted variance. Given a set of n observations $y_1, y_2, ..., y_n$ and associated weights $w_1, w_2, ...w_n$, so that $\sum_{i=1}^{n} w_i = 1$, weighted variance is given by $V_w = \frac{n\sum_{i=1}^{n} w_i(y_i - \bar{y}_w)^2}{n-1}$, where $\bar{y}_w = \frac{\sum_{i=1}^{n} w_i y_i}{\sum_{i=1}^{n} w_i}$.

We want the weights to correspond to the distances, i.e. the smaller the distance between points, the larger the weight should be. A logical choice for calculating weight like that, would be to use the Gaussian kernel function. This means that a slope $s(x_i, x_j)$ should have a corresponding weight $w(x_i, x_j) \propto r(x_i, x_j) = \exp\left(\frac{(x_i - x_j)^2}{\theta}\right)$.

Replotting all the pairwise slopes between points with weights calculated can be seen in Figure 4.6.

We can now calculate the weighted variances for all 40 points based on the equations shown so far. The weighted variance for point x_i is given as

$$V_w(x_i) = \frac{(n-1)\sum_{j=1;j\neq i}^{n} w(x_i, x_j)(s(x_i, x_j) - \bar{s}_w^i)^2}{n-2}, \tag{4.3}$$

where $\bar{s}_w^i = \frac{\sum_{j=1;j\neq i}^{n} w(x_i,x_j)s(x_i,x_j)}{\sum_{j=1;j\neq i}^{n} w(x_i,x_j)}$. These can be seen in Figure 4.7.

To now measure the smoothness, we calculate the stationarity score. We will define the stationarity score as a weighted average over all $V_w(x_i)$, where $i = 1, 2, ..., n$. This means that for a point x^*, the stationarity score is given by $S(x^*) = \frac{\sum_{i=1}^{n} w(x^*,x_i)V_w(x_i)}{\sum_{i=1}^{n} w(x^*,x_i)}$,

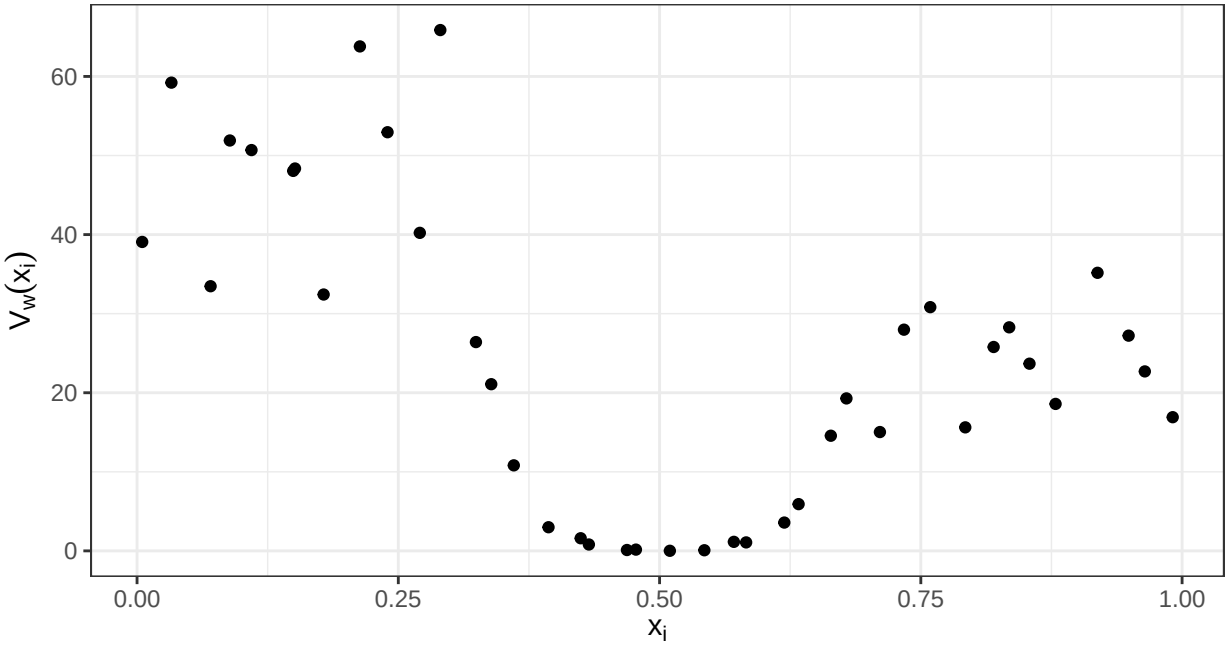

FIGURE 4.7: Weighted variances for all 40 points

where $w(x^*, x_i) =\propto r(x^*, x_i) = \exp\left(\frac{(x^*-x_i)^2}{\theta}\right)$. To interpret the stationarity score, we can normalise it. Consider the mean of all weighted variances $\bar{V}_w = \frac{\sum_{i=1}^n V_w(x_i)}{n}$. This will correspond to the overall average wiggliness and therefore also to $\theta^{-\frac{1}{2}}$ as well as the overall number of level-u upcrossings N_u. If we now define the stationarity score to be $S(x^*) = \frac{1}{\bar{V}_w} \cdot \frac{\sum_{i=1}^n w(x^*,x_i)V_w(x_i)}{\sum_{i=1}^n w(x^*,x_i)}$, then we can interpret this score. Stationarity score will then show how much more wiggly the function is at a location compared to the overall average level of wiggliness The stationarity score for the example in Figure 4.4 can be seen in Figure 4.8.

This function looks how we would expect based off of the true function. Stationarity score is around 0 between $x \in [0.33, 0.66]$, which would mean it's completely flat. It increases to around 1 between $x \in [0.66, 1]$, which indicates this region has the wiggliness corresponding to the wiggliness associated with θ. Finally, the score is around 2 between $x \in [0, 0.33]$ indicating that the function is twice as wiggly as it is between $x \in [0.66, 1]$, which is also true.

With the stationarity score, we are able to define our extended non-stationary ALC function. This will be given by multiplying the stationarity score with the ALC. So in other words, this is

$$NSALC(x|X_{\text{ref}}) = S(x) \cdot ALC(x|X_{\text{ref}}).$$

Coming back to the example in Figure 4.1, we now calculate $NSALC(x)$ as well as $ALC(x)$, which we will compare in Figure 4.9 (b).

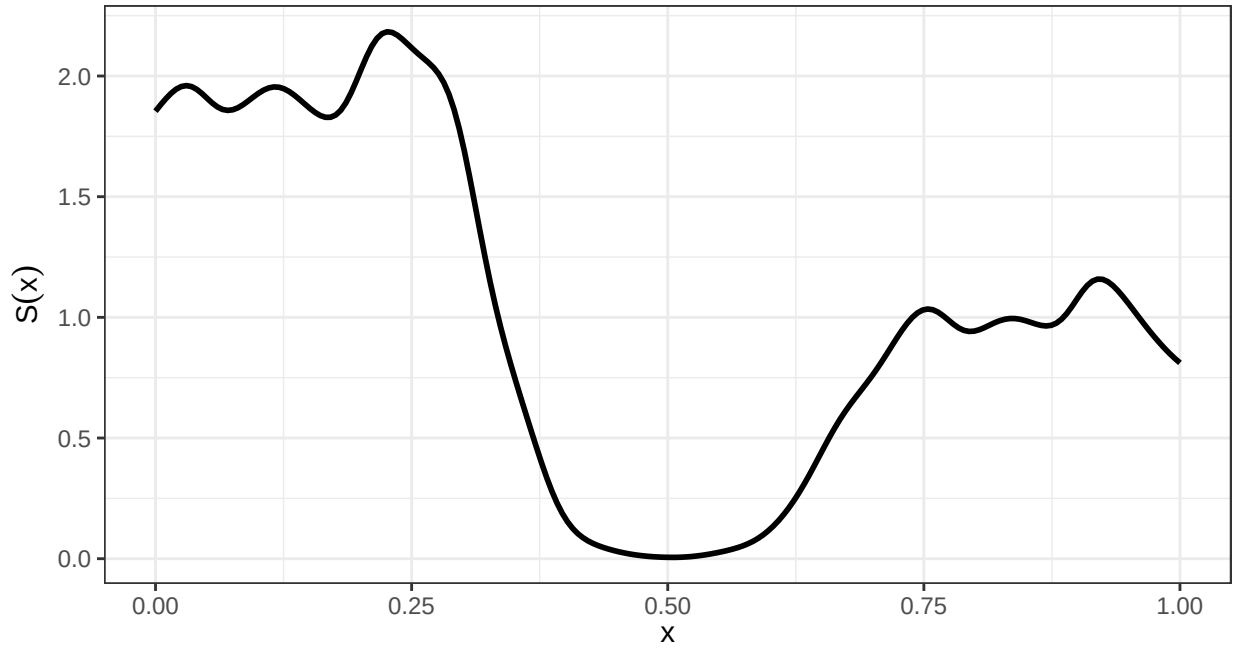

FIGURE 4.8: Stationarity score for a simple example

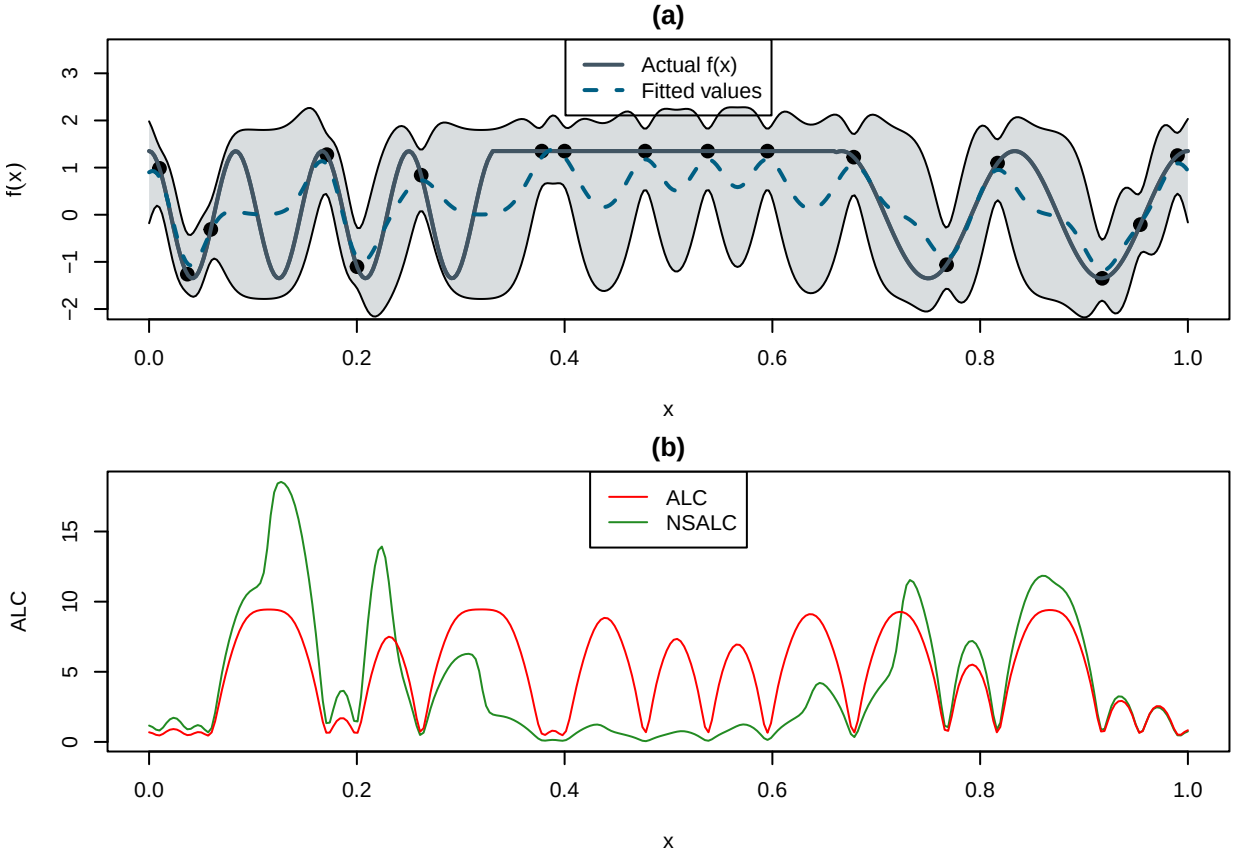

FIGURE 4.9: (a) An example of a fitted GP to a computer model $f_H(x)$ (b) ALC and NSALC values from the GP

As we see from Figure 4.9, $NSALC$ succeeds in achieving our goal. It puts more weight on the most interesting area, which is $x \in [0, 0.33]$. $NSALC$ and ALC values are roughly equal in $x \in [0.66, 1]$ and $NSALC$ is almost zero in $x \in (0.33, 0.66)$, where we are least interested in evaluating new points.

4.4.2 Multidimensional extension

So far we have only considered a 1-dimensional example. Finding slopes in one dimension is simple as it is well-defined and we have $s(x_i, x_j) = \frac{y_i - y_j}{x_i - x_j}$. An important thing to note, is that $s(x_i, x_j) = \frac{y_i - y_j}{x_i - x_j} = \frac{-(y_j - y_i)}{-(x_j - x_i)} = \frac{y_j - y_i}{x_j - x_i} = s(x_j, x_i)$. This means that the slope is always directional. However, it is slightly trickier in higher dimensions as $\mathbf{x}_i - \mathbf{x}_j$ is not a single value anymore. We could simply use any measure for distance that's defined for multidimensional data, such as Euclidean or Manhattan distance, but this would mean that $s(\mathbf{x}_i, \mathbf{x}_j) \neq s(\mathbf{x}_j, \mathbf{x}_i)$ and the slopes would no longer be directional. To demonstrate why this is an issue, consider the following two-dimensional computer model over $x_1, x_2 \in [-1, 1]$:

$$f_I(x_1, x_2) = x_1^2 + x_2^2,$$

where we choose the following design:

$$X = \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{x}_3 \\ \mathbf{x}_4 \\ \mathbf{x}_5 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ -0.5 & -0.5 \\ -0.5 & 0.5 \\ 0.5 & -0.5 \\ 0.5 & 0.5 \end{pmatrix}$$

Evaluating this gives us $\boldsymbol{y} = f_I(X) = (y_1, y_2, y_3, y_4, y_5) = (0, 0.5, 0.5, 0.5, 0.5)$.

This is visualised in Figure 4.10.

Firstly we can show that $s(\mathbf{x}_i, \mathbf{x}_j) \neq s(\mathbf{x}_j, \mathbf{x}_i)$ in this example, by calculating $s(\mathbf{x}_1, \mathbf{x}_2)$ and $s(\mathbf{x}_2, \mathbf{x}_1)$ using the Manhattan distance. We get $s(\mathbf{x}_1, \mathbf{x}_2) = \frac{y_1 - y_2}{\sum_{i=1}^{2} |x_1^i - x_2^i|} = \frac{0 - 0.5}{|0 - (-0.5)| + |0 - (-0.5)|} = -0.5$ and similarly $s(\mathbf{x}_2, \mathbf{x}_1) = \frac{y_2 - y_1}{\sum_{i=1}^{2} |x_2^i - x_1^i|} = \frac{0.5 - 0}{|-0.5 - 0| + |-0.5 - 0|} = 0.5$. Clearly $s(\mathbf{x}_1, \mathbf{x}_2) = -0.5 \neq 0.5 = s(\mathbf{x}_2, \mathbf{x}_1)$. Similar results follow when using any other distance measure.

Let us also calculate slopes between $\mathbf{x}_1$ and the other points, using the Manhattan distance, we already have $s(\mathbf{x}_1, \mathbf{x}_2) = -0.5$ and similarly we can get $s(\mathbf{x}_1, \mathbf{x}_3) = s(\mathbf{x}_1, \mathbf{x}_4) = s(\mathbf{x}_1, \mathbf{x}_5) = -0.5$. These numbers would indicate that the function doesn't change at all in the vicinity of $\mathbf{x}_1 = (0, 0)$. We would expect to see that when the function is flat, but in this case our function has a bowl shape, so this method does not work.

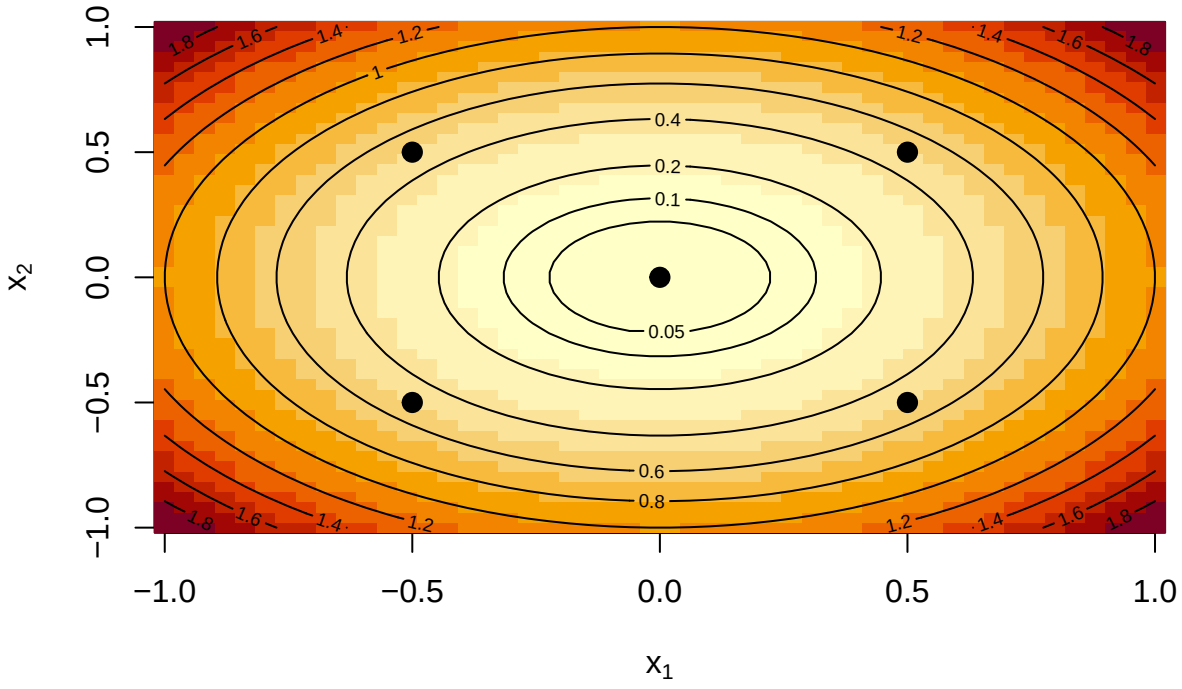

FIGURE 4.10: A contour plot of function $f_I(X)$, where the black dots mark the location of the chosen design

To address this issue, we define the slope direction for every dimension separately. Consider two p-dimensional points $\mathbf{x}_i = (x_i^1, ..., x_i^p)$ and $\mathbf{x}_j = (x_j^1, ..., x_j^p)$ and their function values y_i and y_j respectively. Then the slope for the k^{th} dimension will be defined as

$$s_k(\mathbf{x}_i, \mathbf{x}_j) = \frac{y_i - y_j}{(\sum_{l=1}^{p} |x_i^l - x_j^l|) \cdot sign(x_i^k - x_j^k)},$$

where

$$sign(a) = \begin{cases} -1, & \text{if } a < 0 \\ 1, & \text{if } a \geq 0 \end{cases}$$

Doing this will guarantee that all slopes are directional and describe the function accurately.

We now calculate the weighted variance $V_w^k(\mathbf{x}_i)$ for every point $i = 1, ..., n$ and for every dimension $k = 1, ..., p$. To calculate $V_w^k(\mathbf{x}_i)$, we use Equation 4.3, but we replace $s(x_i, x_j)$ with $s_k(\mathbf{x}_i, \mathbf{x}_j)$. We then simply define $V_w(\mathbf{x}_i) = \frac{1}{p} \sum_{k=1}^{p} V_w^k(\mathbf{x}_i)$. After calculating $V_w(\mathbf{x}_i)$, everything else will follow as defined in Section 4.4.1.

4.5 Examples

In this part we will be evaluating the performance of our $NSALC$ acquisition function. We fit a standard stationary GP and then use $NSALC$ to select a new point to be added to our design. The implementation of our code can be found on Github at `https://github.com/Hendriico/NSALC`. For our baseline, we have three different methods. Firstly, we fit a deep GP at every step and use ALC to choose a new point. For the deep GP and the ALC implementation, we use the `deepgp` package in R (Sauer, 2022). For the second method, we fit a standard stationary GP and use ALC to select a new point. Finally, we simply choose a single maximum projection design (Joseph et al., 2015). This method is not based off of any sequential design criteria. For this implementation, we use the `MaxPro` package in R (Ba and Joseph, 2018).

4.5.1 Langley Glide-Back Booster

Our first example comes from NASA. Pamadi et al. (2004b) describe a rocket booster that could be reused after being used to transport a payload into orbit. To achieve this task, a computer model was created that simulates how the booster behaves under various conditions. The computer model has three input variables and six different output variables. The computer model is deterministic and exhibits clear signs of non-stationarity. The real computer model is not available and therefore we use a dense grid of pre-evaluated points instead. This data can be downloaded at `https://bobby.gramacy.com/surrogates/lgbb.tar.gz`.

We start with an initial maximum projection design (Joseph et al., 2015) of size $n = 50$. We then run each algorithm for extra 250 steps to get a final design of size $n = 300$. We also compare the performance of these sequential designs to a non-sequential maximum projection designs.

To measure the performance of our designs, we use each design to fit a non-stationary deep GP. The fitted model is used to evaluate the root mean squared error (RMSE), which will be our measure for the goodness of fit. RMSE will be calculated will be calculated over 1000 out-of-sample points and is given as

$$RMSE = \sqrt{\frac{\sum_{i=1}^{1000}(\hat{m}_n(\mathbf{x}_i) - f(\mathbf{x}_i))^2}{1000}},$$

where $\hat{m}_n(\mathbf{x}_i)$ is the mean from the fitted non-sationary deep GP at point $\mathbf{x}_i$ and $f(\mathbf{x}_i)$ is the computer model value at $\mathbf{x}_i$. RMSE will be evaluated after every 50 iterations. Each algorithm is run 10 times. The results are seen in Figure 4.11.

From Figure 4.11, we can see that the designs found by $NSALC$ are very similar to the ones found by the deep GP. However, every iteration for $NSALC$ only takes a fraction

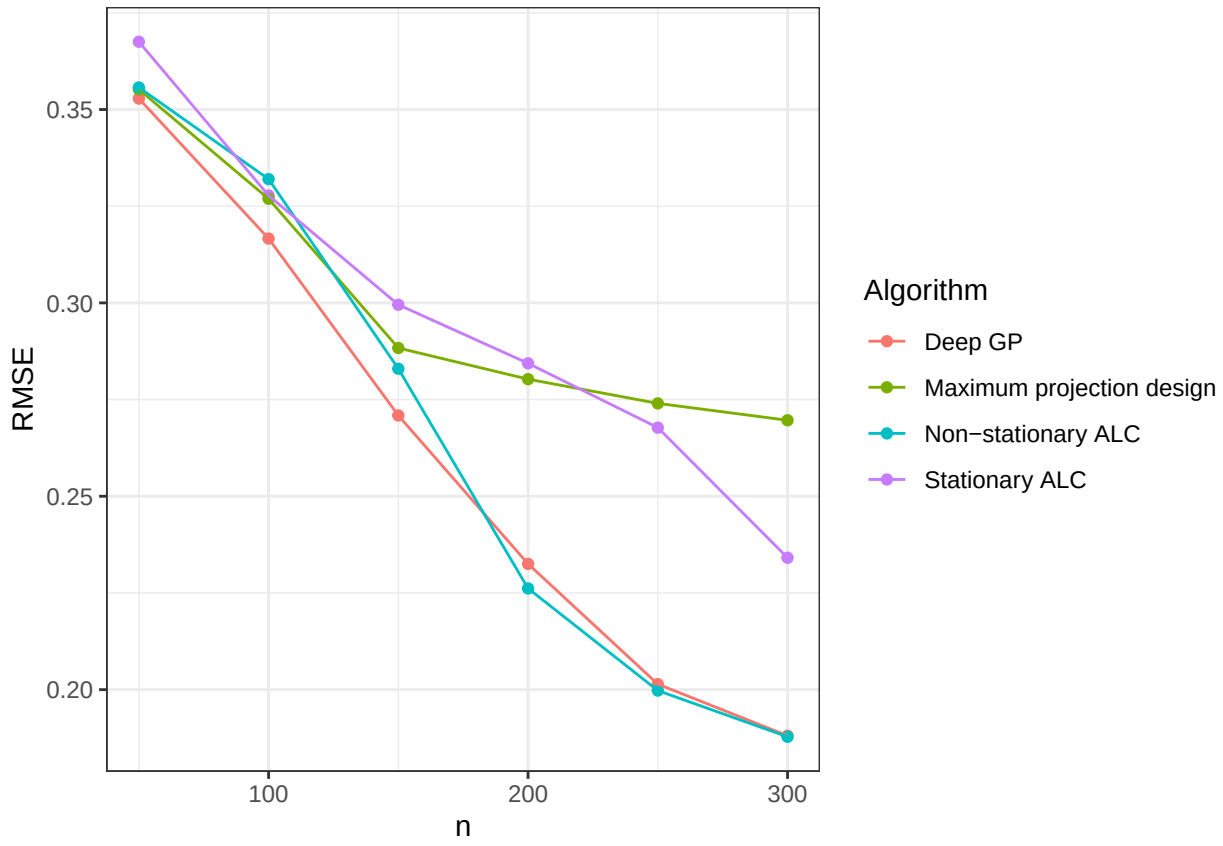

FIGURE 4.11: Simulation results for the Langley Glide-Back Booster computer model.

of time compared to fitting a deep GP at every iteration, making it more practical. An iteration of $NSALC$ only takes a few seconds, while fitting a whole deep GP accurately can take over an hour in practice. There are a few ways to make it faster, such as choosing appropriate starting positions for the different parameters. While this can make the process faster, it is still hundreds or even thousands of times slower than an iteration of $NSALC$. As expected, using a regular ALC with a stationary GP will not give us good results as it treats all areas equally and does not put more points in the more interesting regions. However, it is still slightly better than a non-sequential design strategy presented by `MaxPro`.

4.5.2 Satellite Drag

Our second example is a 7-dimensional computer model (Sun et al., 2019). It was developed by researchers at Los Alamos National Laboratory. It uses 7 input variables to predict drag coefficients for various satellites in orbit. The computer model uses Monte Carlo methods and is therefore noisy. The simulator also requires details about a specific satellite. Similarly to Sauer et al. (2021), we use the GRACE satellite. The computer model is available publicly at `https://bitbucket.org/gramacylab/tpm/`.

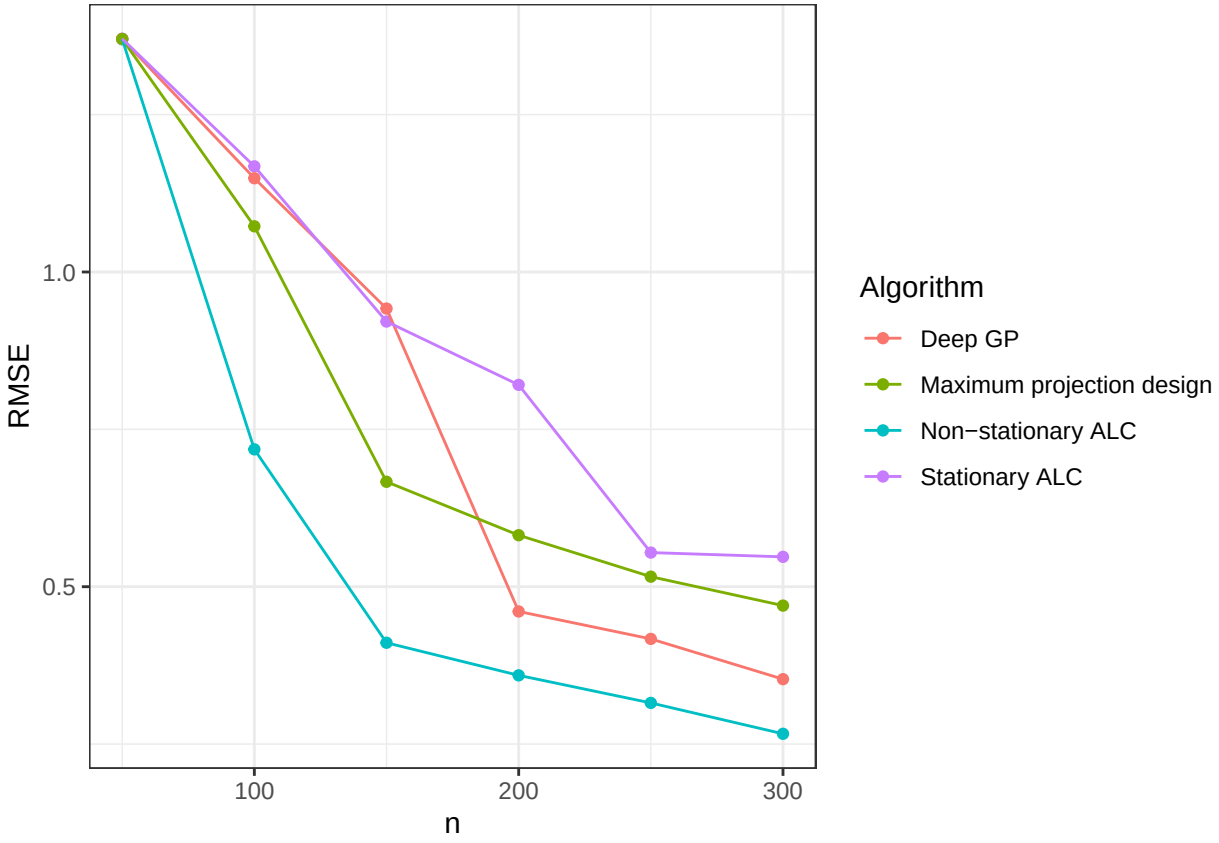

FIGURE 4.12: Simulation results for the satellite drag computer model

We start with an initial maximum projection design (Joseph et al., 2015) of size $n = 50$. We then run each algorithm for extra 250 steps to get a final design of size $n = 300$. Similarly to the last example, we measure the performance of our designs by fitting a deep GP and evaluating RMSE for every fitted model. This is again done after every 50 iterations. Each algorithm is run 10 times. The results are seen in Figure 4.12.

From Figure 4.12, we can see that the designs found by $NSALC$ are better than the ones found by the deep GP and what is more, every iteration for $NSALC$ only takes a fraction of time compared to fitting a deep GP at every iteration, making it more practical. As expected, using a regular ALC with a stationary GP will not give us good results as it treats all areas equally and does not put more points in the more interesting regions. In this scenario, it is also slightly worse than a non-sequential design strategy presented by `MaxPro`. Interestingly, designs found by Deep GP are worse than the ones found by MaxPro until $n = 200$. This could indicate that it takes a while for the Deep GP to learn the structure of the problem correctly and a lot of effort is put into less interesting areas until that point.

4.6 Discussion

The main contribution of this chapter is the proposal of a novel acquisition function which can be used with stationary GPs to build sequential designs for non-stationary computer models. This addressed an important gap in the field of sequential design, which is how to efficiently design experiments for computer models that exhibit non-stationary behaviour without having to fit computationally expensive non-stationary Gaussian Processes. While much of the existing literature assumes stationarity in the underlying process or relies on complex non-stationary GPs, our proposed acquisition function provides a practical and scalable alternative. By using only standard, stationary GPs, the method remains computationally cheap, while still adapting effectively to spatial heterogeneity in the response surface.

The proposed acquisition function intelligently finds areas of higher variance which are considered more interesting and puts more emphasis on those regions. Areas with a slower rate of change are considered less interesting and sampled less.

The efficiency of our method was demonstrated through simulation studies in section 4.5. We compared our approach against three other methods, the most complex being a fully non-stationary sequential design using deep GPs. Our algorithm reached comparable or better results for two different example models. All while only requiring a fraction of the time compared to deep GPs.

Although successful in our studies, this novel methodology still requires further testing across different computer models to validate its effectiveness. Other non-stationary surrogate models should also be considered in further studies. Occasional numerical stability issues were also noted with our method during simulations, indicating potential issues with the method which should be explored.

Chapter 5

Conclusions

This thesis explored key challenges in the design of computer experiments, a rapidly evolving field with applications in various scientific and engineering domains. Experimenters have various objectives, such as optimisation, inverse calibration, sensitivity analysis, etc. Our goal was to develop novel algorithms that enhance Bayesian optimisation techniques to tackle these challenges more effectively.

We developed three novel algorithms for sequential design in computer experiments, each addressing distinct challenges in Bayesian optimisation. The first algorithm is a Bayesian optimisation algorithm that can be used to efficiently optimise multiple objectives of the same computer model simultaneously. The second algorithm is again used for Bayesian optimisation, however, the emphasis is on high-dimensional computer models whose accuracy can be controlled by choosing the amount of computational resources allocated. The final algorithm is used to find sequential designs that maximise prediction accuracy over the entire design space. This was designed to be used with non-stationary computer models, without having to rely on expensive non-stationary surrogate models.

In Chapter 2, we introduced a new entropy-based algorithm for multi-task Bayesian optimisation. The algorithm was defined in terms of the location of the optimum solution and the goal was to reduce the uncertainty about its location by using entropy. The algorithm was able to either maximise or minimise a computer model as well as do both simultaneously. In the multi-task case, the algorithm chooses a new point so that the expected entropy of the joint probability density of the maximum and minimum is minimised at the next step. In the single-objective case, it does so by choosing a new point sequentially, so that the expected entropy of the probability density of the optimum is minimised at the next step.

We approximated the probability densities by using Monte Carlo integration. This method was robust, flexible, unbiased and asymptotically exact, but it was also computationally expensive.

The defined entropy search algorithm was then compared against a simple alternative multi-task algorithm, which was based on expected improvement. They were compared by using three different benchmark functions that ranged from two to ten dimensions. The entropy search algorithm excelled at balancing both objectives in all test cases, dynamically allocating more resources to the more challenging objective. This came at a higher computational cost. On average, the entropy search algorithm took between 0.9-2.8 seconds per iteration for the different test functions. This was around 15 times slower than the simple baseline algorithm. However, a few seconds per iteration is, in most cases, much less than the time it takes to evaluate the computer model. When a computer model takes hours to evaluate then the extra few seconds per iteration are irrelevant. Thus, the entropy search algorithm is a highly competitive approach for multi-task Bayesian optimisation.

In Chapter 3, we introduced a novel Bayesian optimisation algorithm. The algorithm was designed to be used for high-dimensional and noisy computer models whose accuracy we can change. Due to the high dimensionality, we only considered a subset of dimensions at a time, changing the size of that subset over time. At the beginning of the algorithm when the subset was still large, we first used trust regions to fit a local Gaussian Process in a hypercube centered around the best currently observed solution. We then use Thompson Sampling to propose a batch of new points to evaluate cheaply. Based on the cheap evaluations, we use the OCBA (Optimal Computing Budget Allocation) algorithm to allocate more resources to more promising solutions. Once the subset had been reduced to a manageable size, allowing for effective Gaussian Process modelling, we used expected improvement to propose new points.

The proposed algorithm was used to find optimal Bayesian designs for any generic utility function. Since the utility functions usually have to be approximated by using numerical methods and/or MCMC algorithms, means that they are noisy and their accuracy or fidelity can be controlled by allocating more or fewer computational resources to it. The Bayesian designs are also often very high-dimensional, making our algorithm particularly well-suited for this problem.

We demonstrated the efficiency of our algorithm by finding an optimal Bayesian design for a logistic regression model.The results were compared with the ACE algorithm, a state-of-the-art method for finding optimal Bayesian designs. Our algorithm managed to consistently find better solutions than ACE, requiring 50%-100% fewer number of computer model evaluations.

In Chapter 4, our aim was to create sequential designs for non-stationary computer models. Standard methods and acquisition functions do not take non-stationarity into account and put equal weights to all areas of the design space, regardless of whether they exhibit significant variation or remain relatively flat.

One approach to handling non-stationarity is through non-stationary Gaussian Processes, which allow the use of standard acquisition functions. This provides a robust solution by focusing more on interesting regions. However, fitting non-stationary GPs is computationally expensive and as the model must be refitted at every iteration in the sequential design process, the entire design process becomes extremely inefficient. To address this, we proposed an alternative method.

A novel acquisition function was proposed that could be used with standard stationary GPs. It prioritises regions where the function changes more rapidly while allocating fewer resources to flatter areas. This was achieved by calculating pairwise slopes between points and measuring how quickly the slopes change at any given location.

Our acquisition function was compared against designs found using deep GPs, a type of non-stationary GP model. Our method produced comparable designs while requiring only a fraction of the computational effort. That is due to the fact that fitting deep GPs takes much longer than fitting stationary GPs. These two methods were also compared against using standard acquisition functions with stationary GPs. As expected, this achieved worse results compared to the two methods designed for non-stationary computer models.

Each of the newly developed algorithms enables us to tackle problems more efficiently than before. However, they also come with certain limitations that we need to be aware of.

So far, we have only tested our Entropy Search algorithm with test functions that had relatively low dimensions. The most complex function we tested, was the stock allocation problem, which was evaluated in 10 dimensions. Scaling to higher dimensions would require more points to discretise the sample space accurately, leading to increased computational complexity. The computational complexity of the current algorithm increases cubically with M, making it prohibitively expensive in high dimensions. In the future we could also compare against different baseline algorithms For example, consider an algorithm identical to the existing baseline one, with the execption that it uses entropy as an acquisition function instead of EI. This could provide a more fair comparison. Furthermore, even though our proposed algorithm only takes a few seconds per iteration, it is still much more expensive than simple alternatives. This means that when the computer model itself is fairly cheap, the extra cost might be more difficult to justify. This means that our algorithm probably can not be used for high-dimensional or cheap computer models. Finally, the number of parameters that need to be chosen in our algorithm can be a challenge. While we provide recommendations and heuristics for parameter selection, they do not guarantee optimal performance. This makes simpler algorithms more stable and potentially preferable in some worst-case scenarios.

Our high-dimensional optimisation algorithm also has some limitations. The main problem is its implementation. Firstly, in the early stages of our algorithm, we are having

to fit Gaussian Processes for fairly large design matrices, both in terms of number of dimensions as well as the number of evaluations. This requires a lot of computational power as well as cleverness to be able to fit accurately. We believe there is still a lot of room for improvement for developing new Gaussian Process implementations for such scenarios as only a very small number of them provided adequate performance for our use case. Second challenge is the complexity of our own algorithm. There are a lot of steps in our algorithm and also a very large number of parameters that need to be specified. As with the first algorithm, even though we provided heuristics for parameter tuning, further research is needed to validate their robustness across diverse applications.

Our third algorithm also has some drawbacks and limitations. Firstly, as it is purely an approximation of the non-stationarity exhibited, there is no guarantee that this will be accurate in all scenarios. The approach also heavily relies on the fitted Gaussian Process, particularly the choice of lengthscale parameter. Since we know that a standard Gaussian Process is not an accurate representation of the true computer model, there is a possibility that the fitted surrogate fails to deliver good results. We did not encounter any such scenario in our research, however, this is not a guarantee of further success in every single process. Additionally, we observed occasional numerical instabilities when the number of design points was very small, suggesting potential avenues for improving the formulation.

In further research, our multi-task entropy search algorithm could be extended to high-dimensional problems. As mentioned previously, the current version is not scalable to higher dimensions, which means that alternative approximations and techniques would be needed to apply the entropy search algorithm to high-dimensional problems. Some methods from Chapter 3 could be leveraged, though further research is needed. Additionally, making the algorithm more computationally efficient would be another way to improve the algorithm. Instead of using Monte Carlo methods, perhaps some kind of an approximation could be developed. Unfortunately this was outside the scope for this thesis.

Our high-dimensional optimisation algorithm has demonstrated significant advantages over existing methods. However, fitting Gaussian Processes in high dimensions remains an ongoing challenge. Firstly, as mentioned before, the number of Gaussian Process implementations that can handle hundreds of dimensions at a time is still very low, which puts limits on any Bayesian optimisation related work. One obvious way is to simply use more powerful computers, supercomputers or even cloud computing to fit Gaussian Processes. This is again outside of the current scope for this thesis. While our current algorithm works well for the test scenarios we have explored, we believe further work is required to test its feasibility in other areas besides Bayesian DoE. Additionally, simplifying the algorithm without sacrificing performance is an open question for future work.

Our approach of using stationary Gaussian Processes with adaptive acquisition functions offers a promising alternative for non-stationary computer models. However, as a novel method, it requires further validation across diverse test cases to confirm its robustness and reliability. We could also compare our method against a larger number of existing non-stationary surrogate models. Unfortunately due to the large computational complexity of fitting such models, this is outside of the scope for our current work. We have also found our method to occasionally become numerically unstable, especially with very small design matrices where there's very few points to calculate the weighted variance. Even though efficient implementation was used, occasional errors still occurred. Therefore, addressing numerical instabilities and improving reliability in scenarios with small datasets would be valuable directions for further research.

In summary, this thesis advances the field of Bayesian optimisation by introducing novel methods for multi-objective, high-dimensional, and non-stationary computer models. While challenges remain, our contributions provide a foundation for future research in scalable and efficient optimisation techniques. We anticipate that these methods will be valuable in a wide range of applications, from engineering design to machine learning and beyond.

Appendix A

Details on the stock allocation problem

In the stock allocation problem, we were trying to create a portfolio, so that we could balance the risk and reward of that portfolio in a certain way. We wanted to distribute our assets between 9 different stocks as well as an ISA with a guaranteed interest rate of 5%. The 9 stocks we considered were:

1. Berskhire Hathaway (BRK-B)
2. Amazon (AMZN)
3. Microsoft (MSFT)
4. Tesla (TSLA)
5. Apple (AAPL)
6. BitCoin (BTC-USD)
7. Nvidia (NVDA)
8. Google (GOOG)
9. Shopify (SHOP)

For each of those stocks, we used their stock price data between 1st January 2017 and 1st June 2023 to fit an autoregressive integrated moving average (ARIMA) model in R (Hyndman and Khandakar, 2008). From the fitted model we can sample a number of scenarios where we think our portfolio value will be at 60 days after 1st June 2023. We then divide the sampled total value of the portfolio by its initial value. From those sampled values, we take the 5% and the 95% intervals and call them q_5 and q_{95} respectively. Now let $RI = 1 - q_5$ be the risk of the portfolio and $RE = q_{95} - 1$ be the reward of the portfolio. We then wish to define a risk score based on those two values. For a high risk score, we are allowing RI to be high, but then also demand that the RE be also high. For a low risk score, we are fine with a lower RE, but we also want RI to be low. We

define $RI > 0.2$ to be high risk and $RI < 0.2$ to be low risk. With that, we define the following risk score.

$$f(RE, RI) = \begin{cases} 100RE \cdot (RI - 0.2), & \text{if } RI < 0.2 \\ (200(RE - 0.2)^2 - 200(RI - 0.2)^2) \cdot \min(1 + RE - RI, 1)^{40}, & \text{if } RI \geq 0.2 \end{cases}$$

For example, when we take an equal allocation of stocks, i.e. $\mathbf{x} = (0.1, ..., 0.1)$, we get $q_5 = 0.884$ and $q_{95} = 1.159$, which means that $RI = 1 - 0.884 = 0.116$ and $RE = 1.159 - 1 = 0.159$, this gives us $f_3(\mathbf{x}) = -1.336$. If we choose a more risky investment with $\mathbf{x} = \left(\frac{1}{9}, ..., \frac{1}{9}, 0\right)$, then we get $q_5 = 0.865$ and $q_{95} = 1.171$, so $RI = 1 - 0.866 = 0.134$ and $RE = 1.171 - 1 = 0.171$, leading to $f_3(\mathbf{x}) = -1.129$.

For a high-risk portfolio, consider the case where we invest half of our money in Bitcoin and put the other half in the ISA with a guaranteed 5% interest rate. In that case, $q_5 = 0.741$ and $q_{95} = 1.32$. This means that $RI = 1 - 0.741 = 0.259$ and $RE = 1.32 - 1 = 0.32$. This gives us a computer model value of $f_3(\mathbf{x}) = 2.16$.

Appendix B

Single-task entropy search algorithm

Even though we have developed the entropy search algorithm for multi-task optimisation, we can also extend it for standard single-task Bayesian optimisation. In the multi-task setting, our goal is to minimise the uncertainty about the distribution $p_*(\boldsymbol{\chi}_{min}, \boldsymbol{\chi}_{max})$. If we are only interested in minimisation, for example, we can redefine our goal to be minimising the uncertainty about the distribution of $p_*(\boldsymbol{\chi}_{min})$ instead. This distribution is defined as

$$p_*(\boldsymbol{\chi}_{min}) = P\left[\mathbf{x}^*_{min} = \boldsymbol{\chi}_{min}\right].$$

This means that over our reference set $\zeta^F = (\boldsymbol{\chi}_1, ..., \boldsymbol{\chi}_M)$, the entropy is given as $H(p_*(\boldsymbol{\chi}_{min})) = -\sum_{i=1}^{M} p_*(\boldsymbol{\chi}_i) \log(p_*(\boldsymbol{\chi}_i))$.

We can now modify Algorithm 7 slightly to allow for single-task optimisation. Firstly, in lines 3 and 15, we only sample F' points from $\pi_{min}(\boldsymbol{\chi})$. Secondly, in lines 7, 8 and 10 we replace $p_*(\boldsymbol{\chi}_{min}, \boldsymbol{\chi}_{max})$ with $p_*(\boldsymbol{\chi}_{min})$. These are the only changes required and everything else is identical to the multi-task optimisation algorithm.

We have also compared this single-task version of the entropy search algorithm to the algorithms introduced in Section 1.2. These algorithms were MQ, SKO, CKG, EQI and TSSO. We have considered 3 different test functions: Michalewicz, Hartmann3 and Hartmann6 (Al-Roomi, 2015b). We have found the single-task version to also be highly competitive to the other algorithms.

Michalewicz function: Michalewicz function (Al-Roomi, 2015b) is a 2-dimensional function, which we will define as

$$f_4(x) = -\sum_{i=1}^{2} sin(\pi x_i) \sin^{20}(\pi i x_i^2)$$

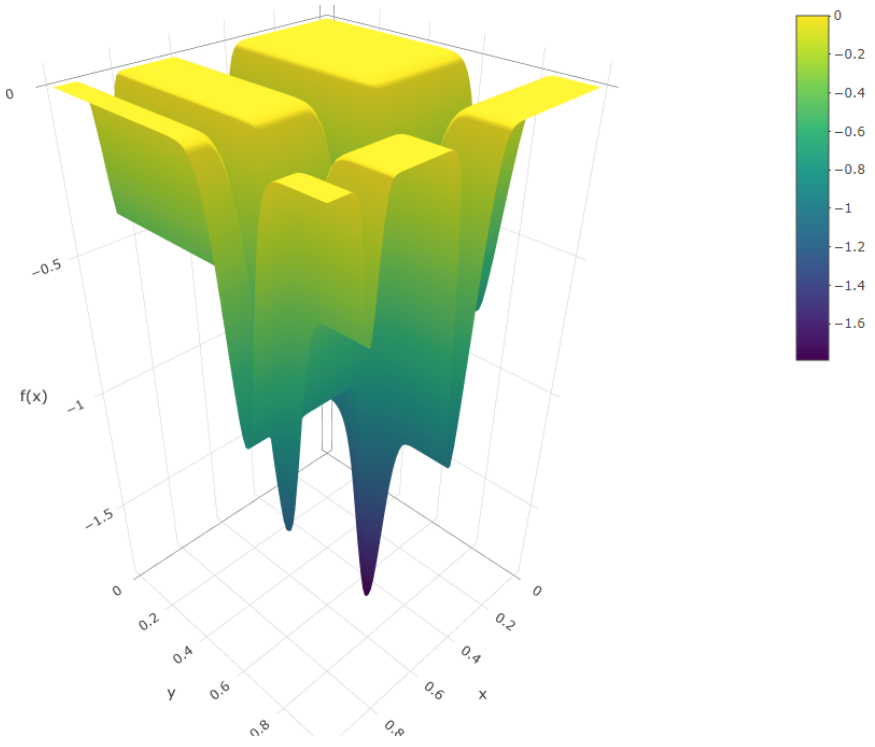

FIGURE B.1: Michalewicz function

It has 2 local minima and is evaluated for $x_1, x_2 \in [0, 1]$. The function can be seen in Figure B.1.

Hartmann3 function: Hartmann3 (Al-Roomi, 2015b) is a 3-dimensional function, which is defined as

$$f_5(x) = -\sum_{i=1}^{4} \alpha_i \exp\left(-\sum_{j=1}^{3} A_{ij}(x_j - P_{ij})^2\right),$$

where

$$\alpha = (1, 1.2, 3, 3.2)^T$$

$$A = \begin{pmatrix} 3 & 10 & 30 \\ 0.1 & 10 & 35 \\ 3 & 10 & 30 \\ 0.1 & 10 & 35 \end{pmatrix}$$

$$P = 10^{-4} \begin{pmatrix} 3689 & 1170 & 2673 \\ 4699 & 4387 & 7470 \\ 1091 & 8732 & 5547 \\ 381 & 5743 & 8828 \end{pmatrix}$$

It has 4 local minima as well as one unique global minimum and is evaluated for $x_1, x_2, x_3 \in [0, 1]$.

Hartmann6 function: Hartmann6 (Al-Roomi, 2015b) is a 6-dimensional function, which is defined as

$$f_6(x) = -\sum_{i=1}^{4} \alpha_i \exp\left(-\sum_{j=1}^{6} A_{ij}(x_j - P_{ij})^2\right),$$

where

$$\begin{aligned}
\alpha =& (1, 1.2, 3, 3.2)^T \\
A =& \begin{pmatrix} 10 & 3 & 17 & 3.5 & 1.7 & 8 \\ 0.05 & 10 & 17 & 0.1 & 8 & 14 \\ 3 & 3.5 & 1.7 & 10 & 17 & 8 \\ 17 & 8 & 0.05 & 10 & 0.1 & 14 \end{pmatrix} \\
P =& 10^{-4} \begin{pmatrix} 1312 & 1696 & 5569 & 124 & 8283 & 5886 \\ 2329 & 4135 & 8307 & 3736 & 1004 & 9991 \\ 2348 & 1451 & 3522 & 2883 & 3047 & 6650 \\ 4047 & 8828 & 8732 & 5743 & 1091 & 381 \end{pmatrix}
\end{aligned}$$

It has 6 local minima as well as one unique global minimum and is evaluated for $x_i \in [0, 1]$, for all $i = 1, ..., 6$.

For each optimisation problem, we ran every algorithm 20 times. The computational budget ranged from 25 to 55 computer model evaluations, which was enough for the algorithms to reach the true minimum in most cases. We also set the computer model noise at $\tau^2 = 0.2$ for every function.

To find the final location of the optimum for a specific algorithm, run and budget, we use the final design from the algorithm and use this to fit a GP. We then find the location of the minimum of the GP mean. In other words, we find $\hat{\mathbf{x}}^* = \arg\min_{\mathbf{x} \in X_n} \hat{m}_n(\mathbf{x})$. To compare the results between algorithms, we finally calculate the true function value for each result and plot the mean of those values for every algorithm and budget.

The simulation results can be seen in Figure B.2. As we can see, Entropy Search (marked ES in Figure B.2) performs significantly better for Michalewicz and Hartmann3 functions than other algorithms. For Hartmann6 function, Entropy Search is comparable to SKO and MQ algorithms and better than the other algorithms.

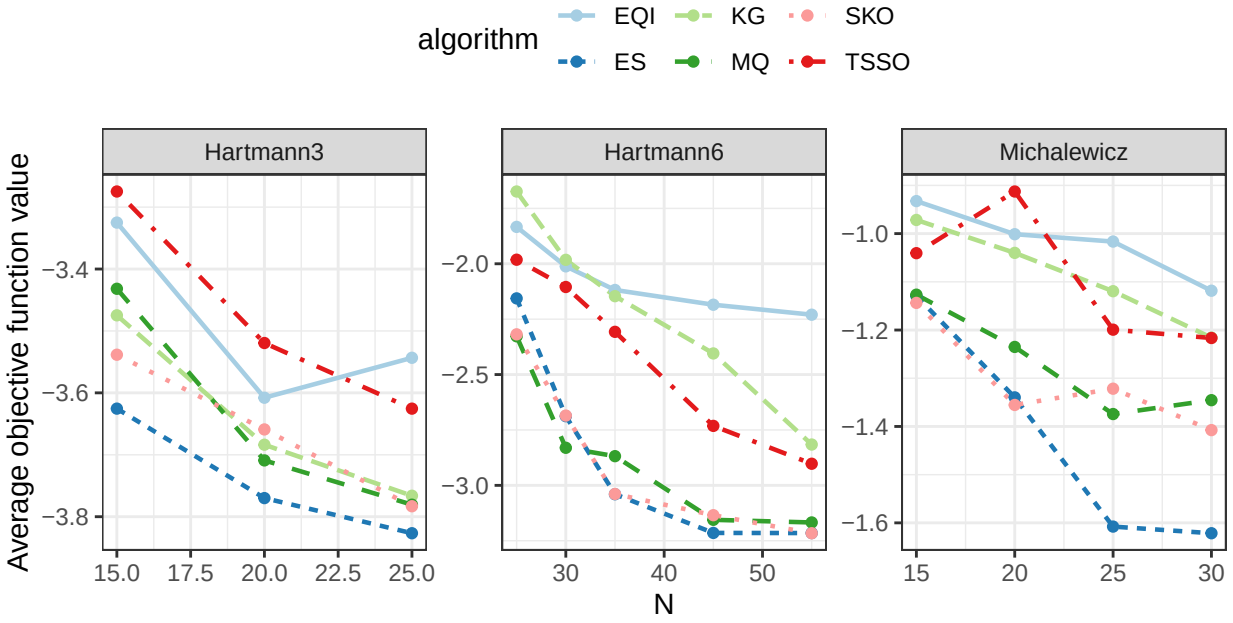

FIGURE B.2: Comparing algorithms on different test functions

Appendix C

Gaussian Process Implementation

A very important part of a Bayesian optimisation algorithm is its implementation. Even a well-designed algorithm will perform poorly when it's badly implemented. A bad implementation can lead to slow running time, numerical instabilities and errors in acquisition function evaluations, poor inner optimisation performance (i.e. not being able to find the maximum of the acquisition function and getting stuck in local optima) as well as any other unexpected errors.

The implementation of a Bayesian optimisation algorithm is reliant on being able to fit a Gaussian process reliably. There are many existing packages that are able to fit a GP. In our work, we have been using the laGP (Gramacy, 2016) package in R (R Core Team (2023)), except where otherwise mentioned. However, there are other implementation of GPs available in R as well. Some other well-known packages are DiceKriging (Roustant et al., 2012) and RobustGaSP (Gu et al., 2018). As far as we know, there's only a very limited number of papers comparing different implementations of GP modelling (e.g. see (Erickson et al., 2018)) and none that would compare these from a (stochastic) Bayesian Optimisation perspective.

We are not going to do a full review of the different packages here. However, we have implemented all of the three mentioned R packages (laGP, DiceKriging and RobustGaSP) in our entropy search algorithm. We will now explore how the choice of a GP package affects the performance of the algorithm.

One of the major differences between laGP and the other two packages is the fact that we are able to specify the correlation function r used in the GP. With laGP, we are only able to use the Gaussian correlation function. However, with both DiceKriging and RobustGaSP, we are able to specify Gaussian, Exponential, Matern3_2, Matern5_2 and the power-exponential correlation functions. In our simulations, we will be using both a Gaussian, as well as the Matern5_2 correlation function, when possible. The Matern5_2 correlation function is given as follows:

$$r(\mathbf{x}, \mathbf{x}'|\boldsymbol{\theta}) = \prod_{i=1}^{d} \left(\left(1 + \sqrt{5}\frac{(x_i - x_i')}{\theta_i} + \frac{5}{3}\left(\frac{(x_i - x_i')}{\theta_i}\right)^2 \right) \exp\left(-\sqrt{5}\frac{(x_i - x_i')}{\theta_i} \right) \right)$$

The Matern5_2 correlation function is only twice differentiable, while the Gaussian correlation function is infinitely differentiable. This means that a Gaussian Process with a Matern5_2 correlation function is less smooth and often preferable to a Gaussian correlation function in practice (Rasmussen and Williams, 2006).

We will be comparing the three packages on 4 different test functions: Michalewicz, Hartmann3, Shekel and Hartmann6.

Michalewicz function: We choose the following parameters for our computer simulations:

- Initial design: maximin Latin Hypercube sample of size 10.
- Computational budget $N \in [15, 20, 25, 30, 35, 40]$
- Computer model noise: $\tau^2 = 0.1$

We then run each algorithm 20 times and get the results that are seen in Figures C.1 and C.2 as well as Table C.1.

Table C.1: Simulation results for the Michalewicz function

Algorithm	N	Mean Optimal Value	Median Optimal Value	Time
DK Matern5_2	40	-1.71	-1.75	100.65 s
laGP	40	-1.61	-1.78	41.61 s
DK Gaussian	40	-1.58	-1.71	99.69 s
robustGaSP Gaussian	40	-1.54	-1.78	50.61 s
robustGaSP Matern5_2	40	-1.49	-1.77	52.99 s

For this function, there are no large differences between the packages. DiceKriging with the Matern5_2 correlation function has the smallest mean optimal value, while laGP has the smallest median optimal value. DiceKriging, however, is computationally much slower than laGP. It is around 2.5 times slower than laGP. The main difference comes from the parallel tempering part of the algorithm, where the package has to repeatedly calculate the GP mean and variance. On the other hand, RobustGaSP with the Matern5_2 correlation function performs the worst, especially at the start of the algorithm.

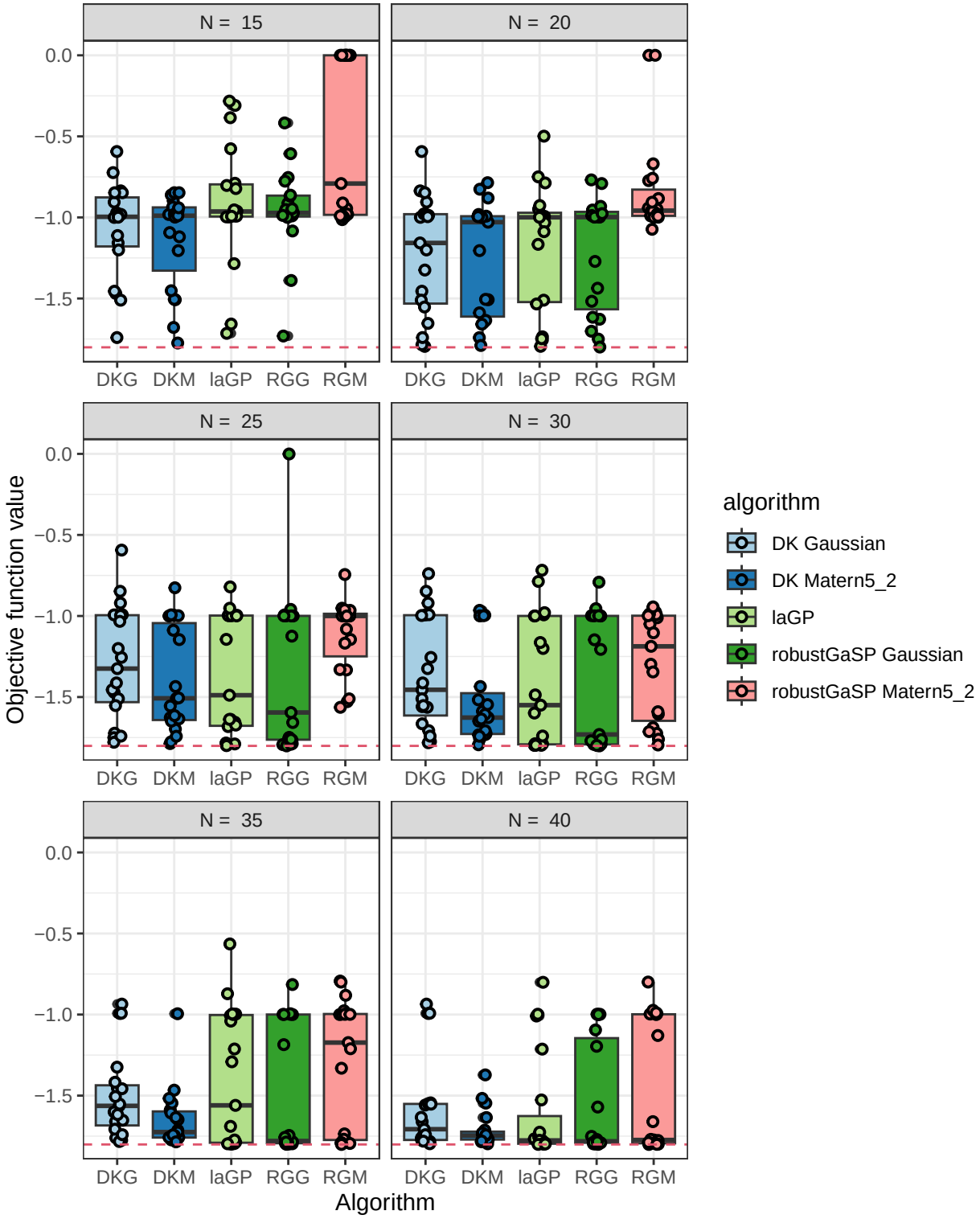

FIGURE C.1: Results for the Michalewicz function

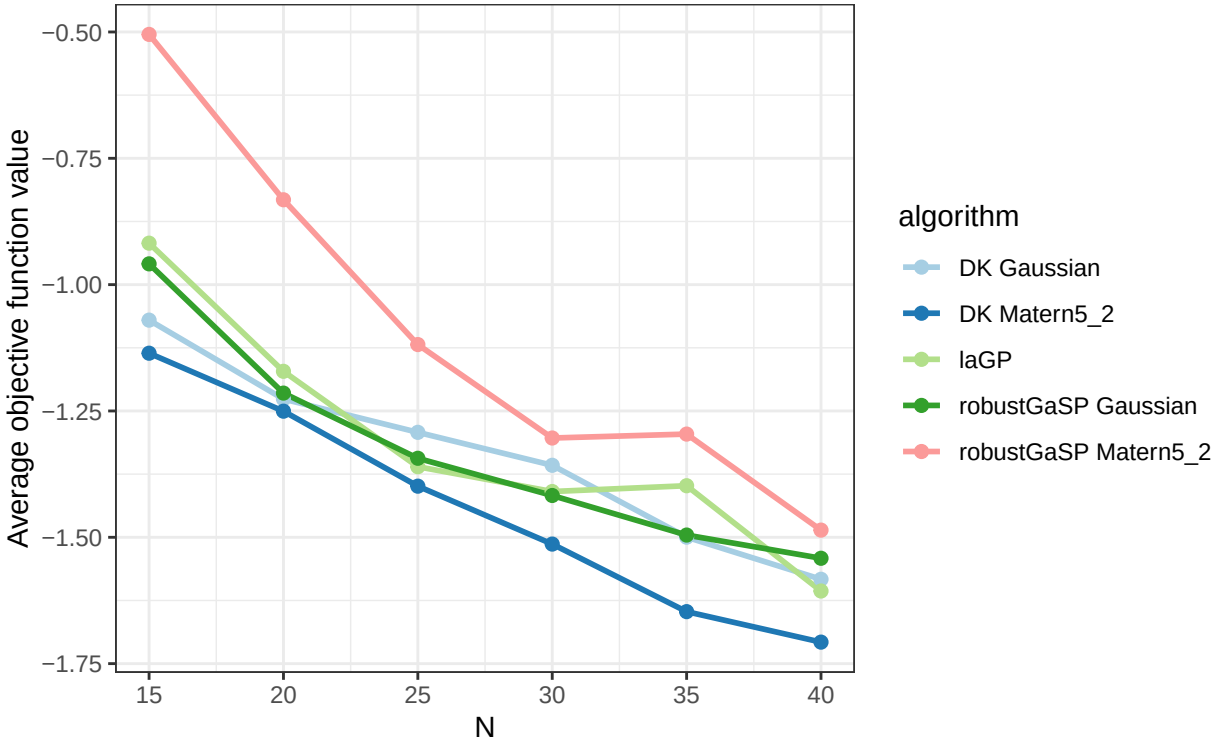

FIGURE C.2: Results for the Michalewicz function

Hartmann3 function: We choose the following parameters for our computer simulations:

- Initial design: maximin Latin Hypercube sample of size 7.
- Computational budget $N \in [10, 15, 20, 25, 30]$
- Computer model noise: $\tau^2 = 0.1$

We then run each algorithm 20 times and get the results that are seen in Figures C.3 and C.4 as well as Table C.2.

TABLE C.2: Simulation results for the Hartmann3 function

Algorithm	N	Mean Optimal Value	Median Optimal Value	Time
robustGaSP Matern5_2	30	-3.85	-3.85	46.41 s
laGP	30	-3.84	-3.85	34.02 s
robustGaSP Gaussian	30	-3.84	-3.85	43.14 s
DK Matern5_2	30	-3.83	-3.83	85.82 s
DK Gaussian	30	-3.78	-3.80	84.57 s

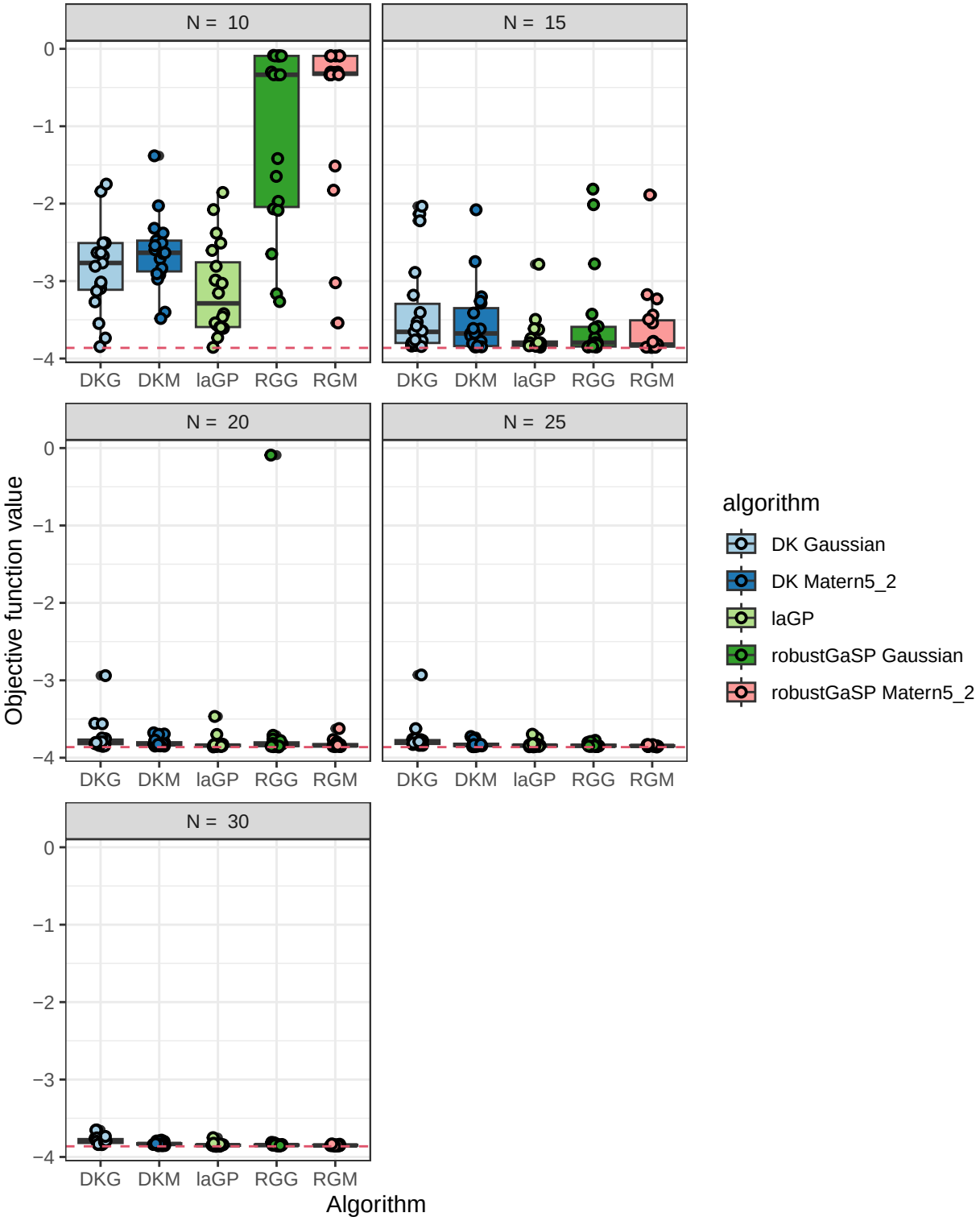

FIGURE C.3: Results for the Hartmann3 function

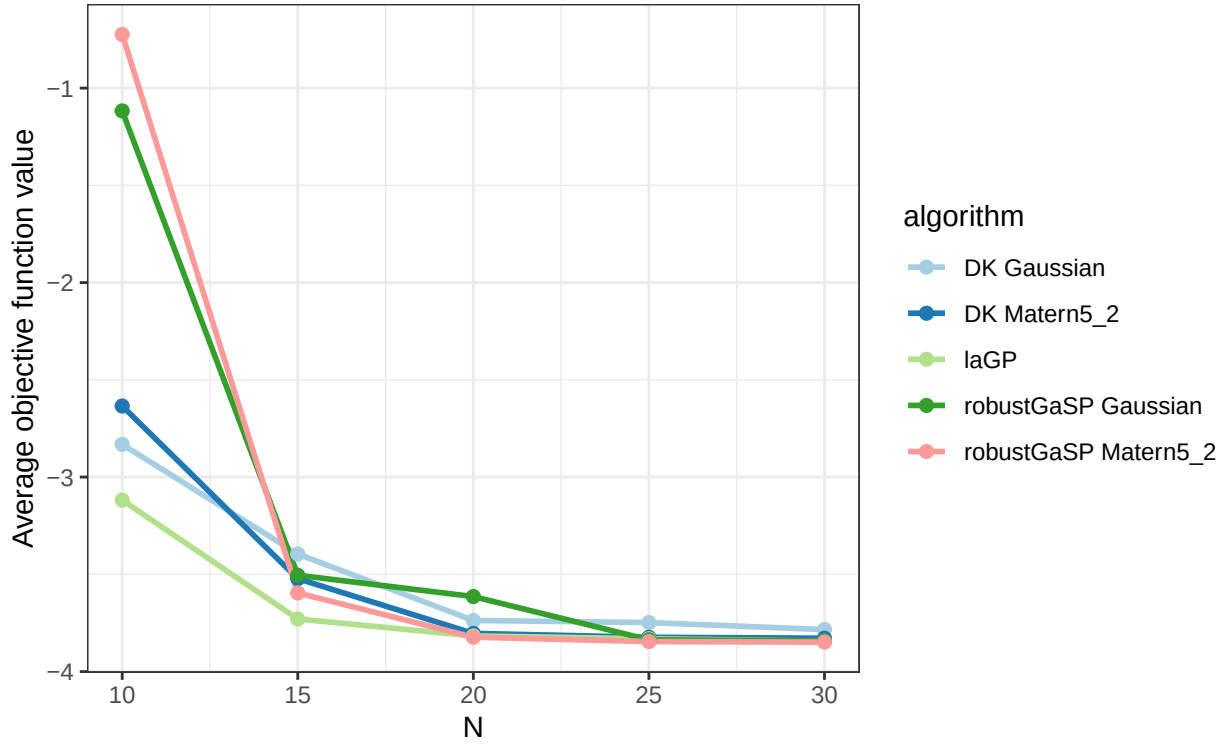

FIGURE C.4: Results for the Hartmann3 function

For this function, all of the packages are very close to each other at the end of the algorithm. The only differences are at the start of the algorithm during the first few iterations, where RobustGaSP performs much worse, and in the speed of the packages.

Shekel function: Shekel function (Al-Roomi, 2015b) is a 4-dimensional function, which is defined as

$$f_7(\mathbf{x}) = -\sum_{i=1}^{10}\left(\sum_{j=1}^{4}(x_j - C_{ji})^2 + \beta_i\right),$$

where

$$\beta = \frac{1}{10}(1,2,2,4,4,6,3,7,5,5)^T$$

$$C = \begin{pmatrix} 4.0 & 1.0 & 8.0 & 6.0 & 3.0 & 2.0 & 5.0 & 8.0 & 6.0 & 7.0 \\ 4.0 & 1.0 & 8.0 & 6.0 & 7.0 & 9.0 & 3.0 & 1.0 & 2.0 & 3.6 \\ 4.0 & 1.0 & 8.0 & 6.0 & 3.0 & 2.0 & 5.0 & 8.0 & 6.0 & 7.0 \\ 4.0 & 1.0 & 8.0 & 6.0 & 7.0 & 9.0 & 3.0 & 1.0 & 2.0 & 3.6 \end{pmatrix}$$

We choose the following parameters for our computer simulations:

- Initial design: maximin Latin Hypercube sample of size 20.
- Computational budget $N \in [30, 40, 50, 60]$
- Computer model noise: $\tau^2 = 0.1$

We then run each algorithm 20 times and get the results that are seen in Figures C.5 and C.6 as well as Table C.3.

Table C.3: Simulation results for the Shekel function

Algorithm	N	Mean Optimal Value	Median Optimal Value	Time
DK Matern5_2	60	-2.70	-2.58	191.49 s
DK Gaussian	60	-2.63	-2.53	181.56 s
laGP	60	-2.53	-2.65	88.20 s
robustGaSP Gaussian	60	-0.65	-0.42	119.43 s
robustGaSP Matern5_2	60	-0.22	-0.18	159.45 s

For this function, DiceKriging starts reaching the optimum quicker than the other packages. By $N = 60$, there are still a few instances where laGP fails to get close to the optimum, which leads to a higher mean optimum value. However, if you discard these few runs, laGP does get closer to the optimum, which leads to a lower median optimum value. RobustGaSP does not perform as well and does not reach the optimum a single time by $N = 60$, with Matern5_2 correlation function performing particularly poorly.

Hartmann6 function: We choose the following parameters for our computer simulations:

- Initial design: maximin Latin Hypercube sample of size 20.
- Computational budget $N \in [30, 35, 40, 45, 50, 55, 60]$
- Computer model noise: $\tau^2 = 0.1$

We then run each algorithm 20 times and get the results that are seen in Figures C.7 and C.8 as well as Table C.4.

Table C.4: Simulation results for the Hartmann6 function

Algorithm	N	Mean Optimal Value	Median Optimal Value	Time
laGP	60	-2.94	-2.96	92.37 s
DK Matern5_2	60	-2.91	-2.95	182.02 s
DK Gaussian	60	-2.89	-2.93	169.85 s
robustGaSP Matern5_2	60	-1.62	-1.44	222.29 s
robustGaSP Gaussian	60	-1.49	-1.42	133.58 s

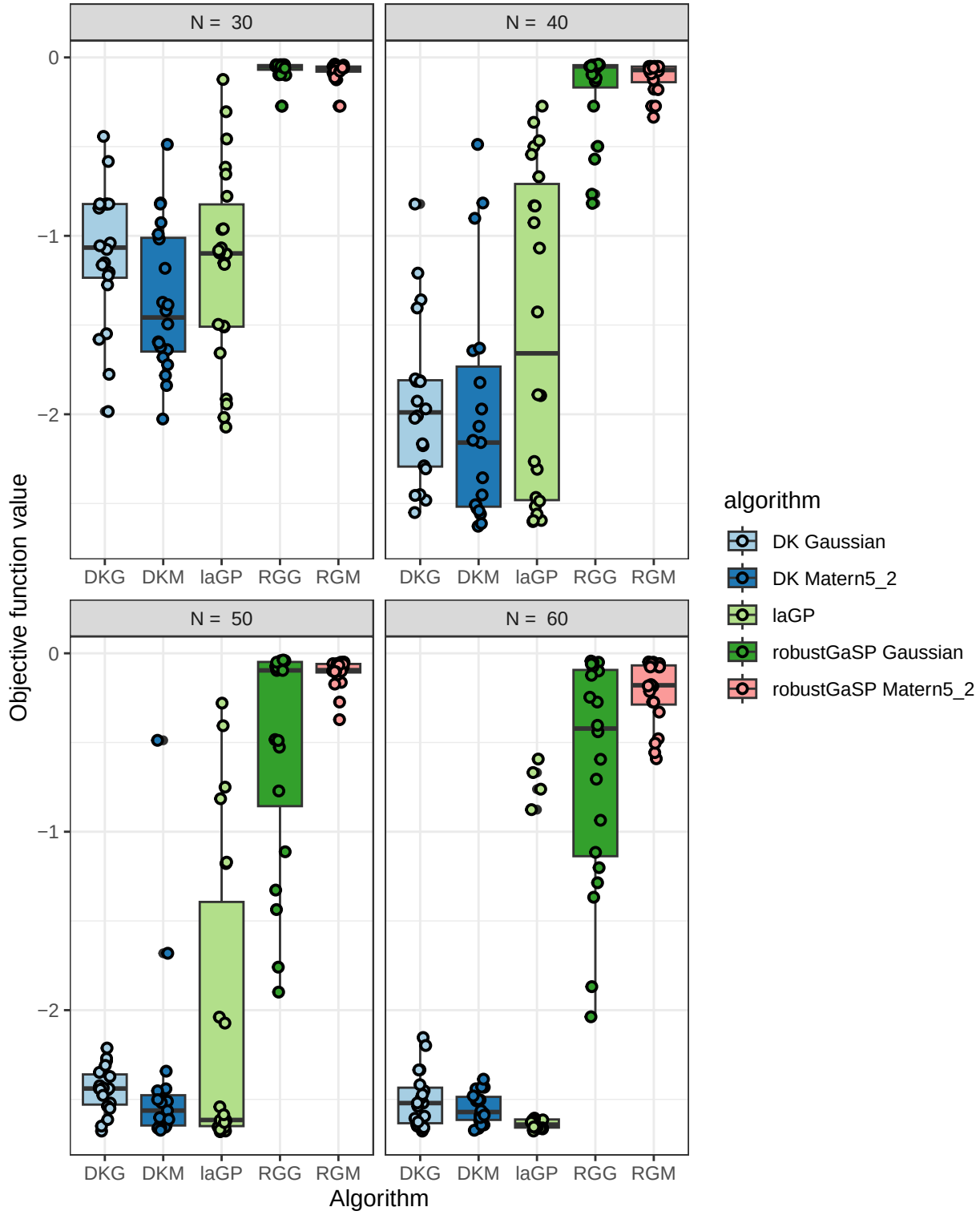

FIGURE C.5: Results for the Shekel function

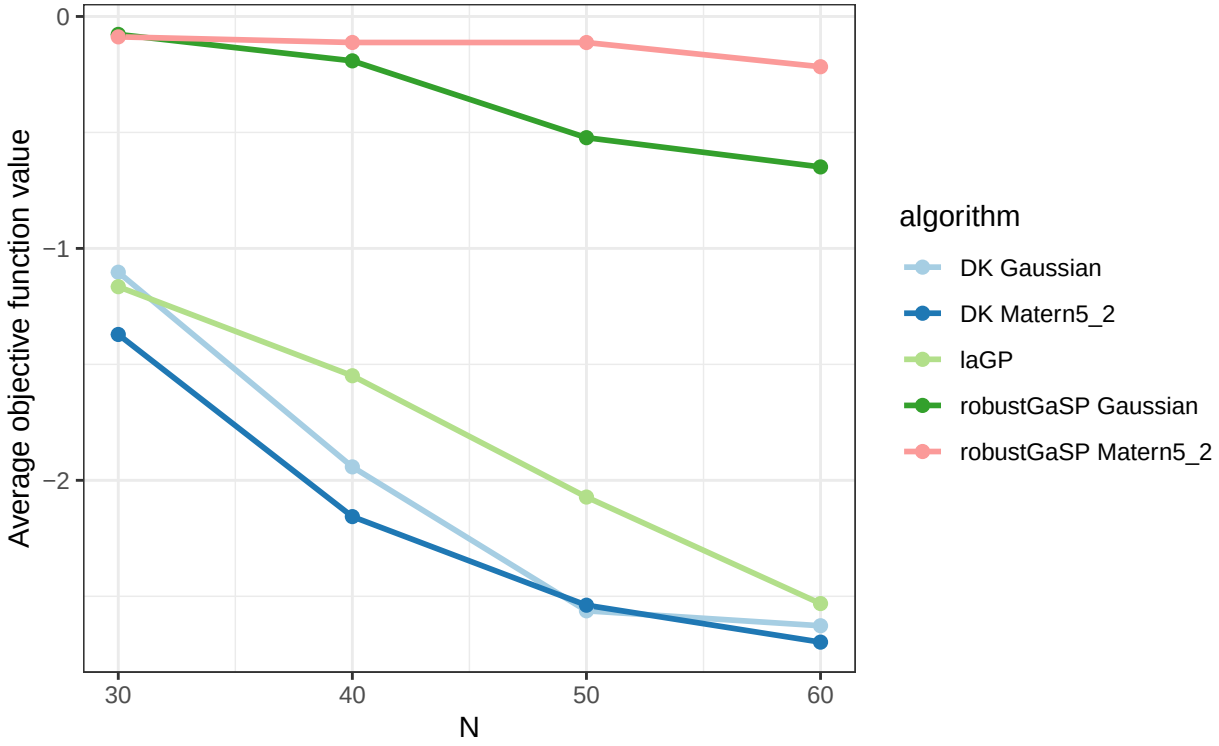

FIGURE C.6: Results for the Shekel function

For this function, laGP and DiceKriging are very close to each other and there are no significant differences. DiceKriging with the Matern5_2 does start reaching the optimum slightly quicker, but the other 2 catch up by $N = 50$. RobustGaSP again, fails to reach the optimum a single time by $N = 60$, which could be an indication that RobustGaSP does not perform as well in higher dimensions, where our design matrices are sparse.

Overall, DiceKriging and laGP are both comparable in performance, with DiceKriging having a slight edge over laGP in most cases. DiceKriging has the added benefit of being able to specify the correlation function, while laGP has the added benefit of being much quicker. Based on our simulations, we would not recommend using RobustGaSP, when having to model a function in higher than 3 dimensions.

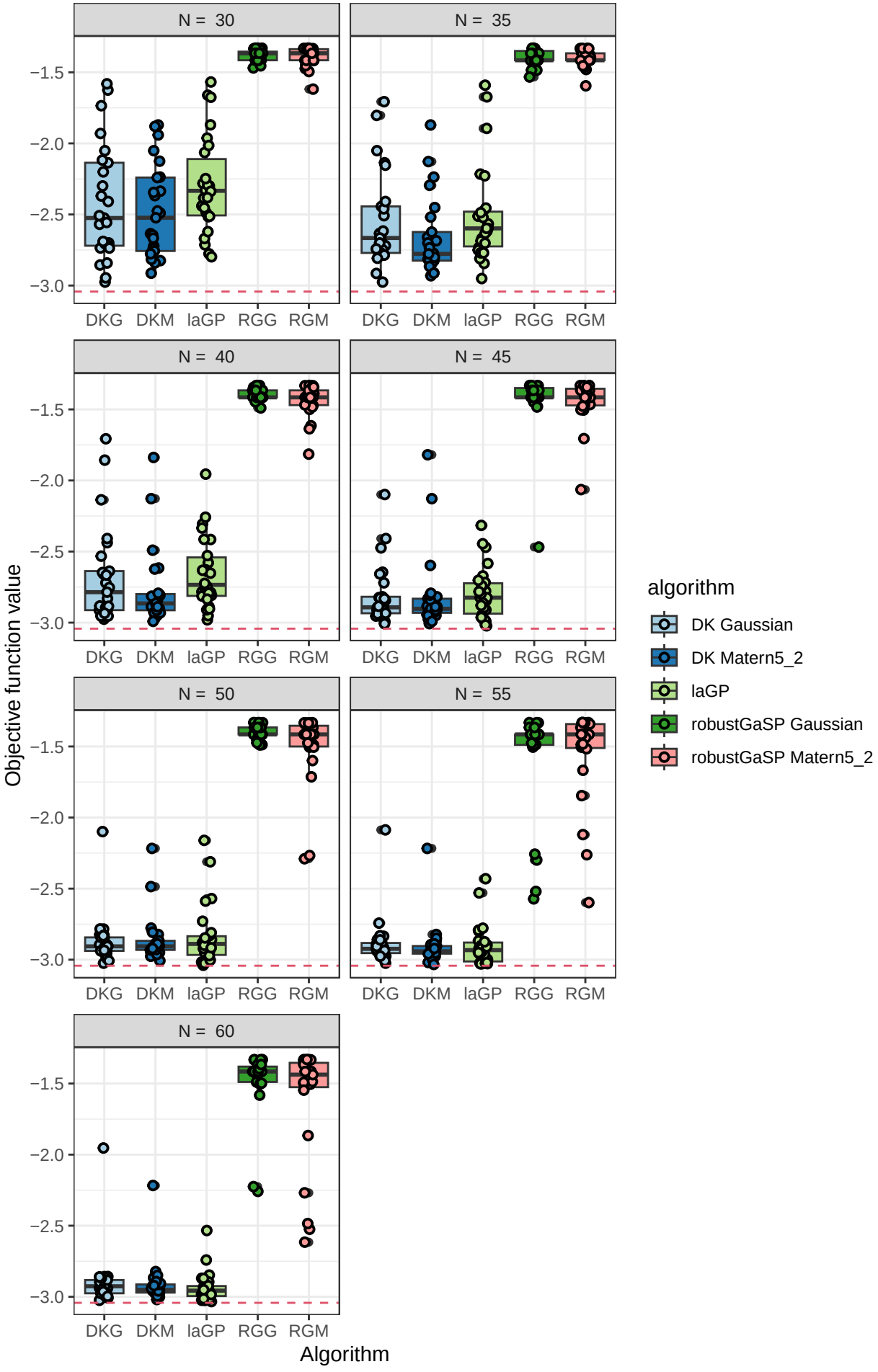

FIGURE C.7: Results for the Hartmann6 function

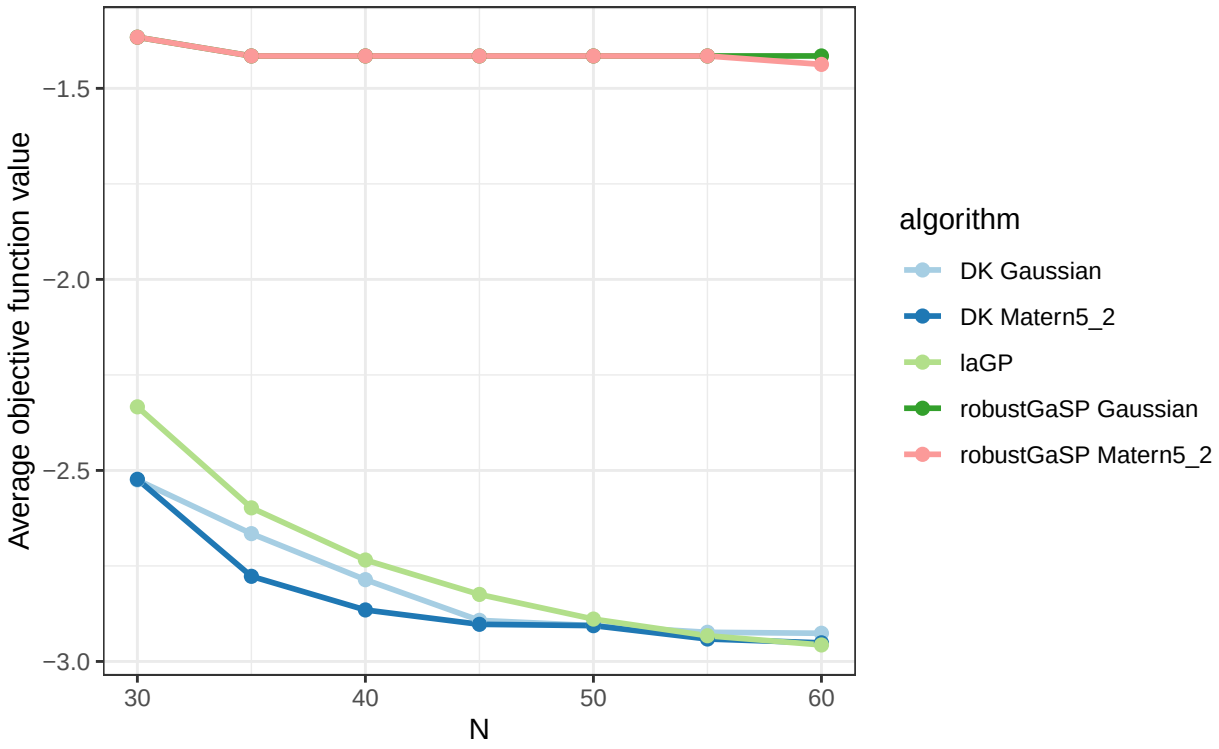

FIGURE C.8: Results for the Hartmann6 function

References

Robert J. Adler and Jonathan E. Taylor. *Gaussian Fields*, pages 7–48. Springer New York, New York, NY, 2007. ISBN 978-0-387-48116-6. . URL https://doi.org/10.1007/978-0-387-48116-6_1.

Ali R. Al-Roomi. Unconstrained Single-Objective Benchmark Functions Repository, 2015a. URL https://www.al-roomi.org/benchmarks/unconstrained.

Ali R. Al-Roomi. Unconstrained Single-Objective Benchmark Functions Repository, 2015b. URL https://www.al-roomi.org/benchmarks/unconstrained.

John-Alexander M. Assael, Ziyu Wang, Bobak Shahriari, and Nando de Freitas. Heteroscedastic treed bayesian optimisation, 2015.

Shan Ba and V. Roshan Joseph. *MaxPro: Maximum Projection Designs*, 2018. URL https://CRAN.R-project.org/package=MaxPro. R package version 4.1-2.

B.W. Bader. 1.16 - constrained and unconstrained optimization. In Steven D. Brown, Romá Tauler, and Beata Walczak, editors, *Comprehensive Chemometrics*, pages 507–545. Elsevier, Oxford, 2009. ISBN 978-0-444-52701-1. . URL https://www.sciencedirect.com/science/article/pii/B9780444527011000351.

Merve Bodur, Timothy C. Y. Chan, and Ian Yihang Zhu. Inverse mixed integer optimization: Polyhedral insights and trust region methods, 2021. URL https://arxiv.org/abs/2008.00301.

Luke Bornn, Gavin Shaddick, and James V Zidek. Modeling non-stationary processes through dimension expansion, 2011.

Thang D. Bui, Daniel Hernandez-Lobato, Yingzhen Li, Jos?? Miguel Hernandez-Lobato, and Richard E. Turner. Deep gaussian processes for regression using approximate expectation propagation, 2016.

Bo Chen, Rui Castro, and Andreas Krause. Joint optimization and variable selection of high-dimensional gaussian processes. *Proceedings of the 29th International Conference on Machine Learning, ICML 2012*, 2, 06 2012.

Chun-Hung Chen, Jianwu Lin, Enver Yücesan, and Stephen E. Chick. Simulation budget allocation for further enhancing the efficiency of ordinal optimization. *Discrete Event Dynamic Systems*, 10(3):251–270, jul 2000. ISSN 0924-6703. . URL https://doi.org/10.1023/A:1008349927281.

Clément Chevalier and David Ginsbourger. Fast computation of the multi-points expected improvement with applications in batch selection. In Giuseppe Nicosia and Panos Pardalos, editors, *Learning and Intelligent Optimization*, pages 59–69, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-44973-4.

David A. Cohn. Neural network exploration using optimal experiment design. *Neural Networks*, 9(6):1071–1083, 1996. ISSN 0893-6080. . URL https://www.sciencedirect.com/science/article/pii/0893608095001379.

T. M. Cover and Joy A. Thomas. *Elements of information theory*. Wiley, 1991.

Karel Crombecq, Dirk Gorissen, Dirk Deschrijver, and Tom Dhaene. A novel hybrid sequential design strategy for global surrogate modeling of computer experiments. *SIAM Journal on Scientific Computing*, 33(4):1948–1974, 2011. . URL https://doi.org/10.1137/090761811.

Andreas Damianou and Neil D. Lawrence. Deep Gaussian processes. In Carlos M. Carvalho and Pradeep Ravikumar, editors, *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics*, volume 31 of *Proceedings of Machine Learning Research*, pages 207–215, Scottsdale, Arizona, USA, 29 Apr–01 May 2013. PMLR. URL https://proceedings.mlr.press/v31/damianou13a.html.

Samuel Daulton, David Eriksson, Maximilian Balandat, and Eytan Bakshy. Multi-objective bayesian optimization over high-dimensional search spaces, 2022.

Pierre Del Moral. Non linear filtering: Interacting particle solution. *Markov Processes and Related Fields*, 2:555–580, 03 1996.

David J. Earl and Michael W. Deem. Parallel tempering: Theory, applications, and new perspectives. *Physical Chemistry Chemical Physics (Incorporating Faraday Transactions)*, 7(23):3910, January 2005. .

Collin B. Erickson, Bruce E. Ankenman, and Susan M. Sanchez. Comparison of Gaussian process modeling software. *European Journal of Operational Research*, 266(1):179–192, 2018. ISSN 0377-2217. . URL https://www.sciencedirect.com/science/article/pii/S0377221717308962.

David Eriksson, Michael Pearce, Jacob Gardner, Ryan D Turner, and Matthias Poloczek. Scalable global optimization via local bayesian optimization. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/6c990b7aca7bc7058f5e98ea909e924b-Paper.pdf.

Marco Falcioni and Michael W. Deem. A biased monte carlo scheme for zeolite structure solution. *The Journal of Chemical Physics*, 110(3):1754–1766, jan 1999. . URL https://doi.org/10.10632F1.477812.

Kai-Tai Fang, Runze Li, and Agus Sudjianto. *Design and Modeling for Computer Experiments (Computer Science & Data Analysis)*. Chapman & Hall/CRC, 2005. ISBN 1584885467.

Alexander I. J. Forrester. Comment: Properties and practicalities of the expected quantile improvement. *Technometrics*, 55(1):13–18, 2013. . URL https://doi.org/10.1080/00401706.2012.733322.

Alexander I.J Forrester, András Sóbester, and Andy J Keane. Multi-fidelity optimization via surrogate modelling. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 463(2088):3251–3269, 2007. . URL https://royalsocietypublishing.org/doi/abs/10.1098/rspa.2007.1900.

Peter Frazier, Warren Powell, and Savas Dayanik. The Knowledge-Gradient Policy for Correlated Normal Beliefs. *INFORMS Journal on Computing*, 21(4):599–613, November 2009. . URL https://ideas.repec.org/a/inm/orijoc/v21y2009i4p599-613.html.

N. Friel and A. N. Pettitt. Marginal likelihood estimation via power posteriors. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, 70(3):589–607, 2008. ISSN 13697412, 14679868. URL http://www.jstor.org/stable/20203843.

Jacob R. Gardner, Chuan Guo, Kilian Q. Weinberger, R. Garnett, and Roger Baker Grosse. Discovering and exploiting additive structure for bayesian optimization. In *International Conference on Artificial Intelligence and Statistics*, 2017.

Sushant S. Garud, Iftekhar A. Karimi, and Markus Kraft. Design of computer experiments: A review. *Computers & Chemical Engineering*, 106:71–95, 2017a. ISSN 0098-1354. . URL https://www.sciencedirect.com/science/article/pii/S0098135417302090. ESCAPE-26.

Sushant Suhas Garud, I.A. Karimi, and Markus Kraft. Smart sampling algorithm for surrogate model development. *Computers & Chemical Engineering*, 96:103–114, 2017b. ISSN 0098-1354. . URL https://www.sciencedirect.com/science/article/pii/S0098135416303210.

A. Gelman, W. R. Gilks, and G. O. Roberts. Weak convergence and optimal scaling of random walk Metropolis algorithms. *The Annals of Applied Probability*, 7(1):110 – 120, 1997. . URL https://doi.org/10.1214/aoap/1034625254.

CJ Geyer, MK Elaine, and MK Selma. Proc. 23rd symp. interface, computing science and statistics, 1991.

David Ginsbourger, Rodolphe Le Riche, and Laurent Carraro. A multi-points criterion for deterministic parallel global optimization based on kriging. 03 2008.

Javier González, Zhenwen Dai, Philipp Hennig, and Neil D. Lawrence. Batch bayesian optimization via local penalization, 2015.

Robert B. Gramacy. laGP: Large-scale spatial modeling via local approximate gaussian processes in R. *Journal of Statistical Software*, 72(1):1–46, 2016. .

Robert B. Gramacy and Herbert K. H. Lee. Bayesian treed gaussian process models with an application to computer modeling, 2009.

Robert B. Gramacy and Herbert K. H. Lee. Cases for the nugget in modeling computer experiments, 2010.

Ryan-Rhys Griffiths and Jose Miguel Hernandez-Lobato. Constrained bayesian optimization for automatic chemical design using variational autoencoders. *CHEMICAL SCIENCE*, 11(2):577–586, JAN 14 2020. ISSN 2041-6520. .

Mengyang Gu, Jesús Palomo, and James O. Berger. Robustgasp: Robust gaussian stochastic process emulation in r, 2018. URL https://arxiv.org/abs/1801.01874.

W. K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109, 04 1970. ISSN 0006-3444. . URL https://doi.org/10.1093/biomet/57.1.97.

Marton Havasi, Jos?? Miguel Hernandez-Lobato, and Juan Jos?? Murillo-Fuentes. Inference in deep gaussian processes using stochastic gradient hamiltonian monte carlo, 2018.

Philipp Hennig and Christian J. Schuler. Entropy search for information-efficient global optimization. *J. Mach. Learn. Res.*, 13(null):1809–1837, jun 2012. ISSN 1532-4435.

José Miguel Hernández-Lobato, Matthew W. Hoffman, and Zoubin Ghahramani. Predictive entropy search for efficient global optimization of black-box functions, 2014. URL https://arxiv.org/abs/1406.2541.

José Miguel Hernández-Lobato, James Requeima, Edward O. Pyzer-Knapp, and Alán Aspuru-Guzik. Parallel and distributed thompson sampling for large-scale accelerated exploration of chemical space, 2017.

Jeffrey W. Herrmann, Hongjie Liu, and Donald K. Milton. Modeling the spread of covid-19 in university communities, 2024. URL https://arxiv.org/abs/2403.10402.

D. Huang, T. T. Allen, W. I. Notz, and R. A. Miller. Sequential kriging optimization using multiple-fidelity evaluations. *Structural and Multidisciplinary Optimization*, 32 (5):369–382, 2006a. .

D. Huang, T. T. Allen, W. I. Notz, and N. Zeng. Global optimization of stochastic black-box systems via sequential kriging meta-models. *J. of Global Optimization*, 34(3):441–466, mar 2006b. ISSN 0925-5001. . URL https://doi.org/10.1007/s10898-005-2454-3.

Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proceedings of the 5th International Conference on Learning and Intelligent Optimization*, LION'05, page 507–523, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 9783642255656. . URL https://doi.org/10.1007/978-3-642-25566-3_40.

Rob J. Hyndman and Yeasmin Khandakar. Automatic time series forecasting: The forecast package for r. *Journal of Statistical Software*, 27(3):1–22, 2008. . URL https://www.jstatsoft.org/index.php/jss/article/view/v027i03.

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, December 1998. ISSN 1573-2916. . URL https://doi.org/10.1023/A:1008306431147.

V. Roshan Joseph, Evren Gul, and Shan Ba. Maximum projection designs for computer experiments. *Biometrika*, 102(2):371–380, 2015. ISSN 00063444. URL http://www.jstor.org/stable/43908541.

Kirthevasan Kandasamy, Jeff Schneider, and Barnabás Póczos. High dimensional bayesian optimisation and bandits via additive models. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, page 295–304. JMLR.org, 2015.

Kirthevasan Kandasamy, Gautam Dasarathy, Junier B Oliva, Jeff Schneider, and Barnabas Poczos. Gaussian process bandit optimisation with multi-fidelity evaluations. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper_files/paper/2016/file/605ff764c617d3cd28dbbdd72be8f9a2-Paper.pdf.

Matthias Katzfuss. Bayesian nonstationary spatial modeling for very large datasets. *Environmetrics*, 24(3):189???200, February 2013. ISSN 1099-095X. . URL http://dx.doi.org/10.1002/env.2200.

David A. Kofke. On the acceptance probability of replica-exchange monte carlo trials. *The Journal of Chemical Physics*, 117(15):6911–6914, 2002. . URL https://doi.org/10.1063/1.1507776.

Ron Kohavi and Mehran Sahami. Error-based and entropy-based discretization of continuous features. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, page 114–119. AAAI Press, 1996.

Takehisa Kohira, Hiromasa Kemmotsu, Oyama Akira, and Tomoaki Tatsukawa. Proposal of benchmark problem based on real-world car structure design optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '18, page 183–184, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450357647. . URL https://doi.org/10.1145/3205651.3205702.

Mario Köppen. The curse of dimensionality. 2000.

Shibo Li, Wei Xing, Mike Kirby, and Shandian Zhe. Multi-fidelity bayesian optimization via deep neural networks, 2020.

Jun S. Liu and Rong Chen. Sequential monte carlo methods for dynamic systems. *Journal of the American Statistical Association*, 93(443):1032–1044, 1998. ISSN 01621459. URL http://www.jstor.org/stable/2669847.

Z. Q. Luo and P. Tseng. On the convergence of the coordinate descent method for convex differentiable minimization. *J. Optim. Theory Appl.*, 72(1):7–35, January 1992. ISSN 0022-3239. . URL https://doi.org/10.1007/BF00939948.

David J. C. MacKay. Information-based objective functions for active data selection. *Neural Computation*, 4(4):590–604, 1992. .

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.

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, 06 1953. ISSN 0021-9606. . URL https://doi.org/10.1063/1.1699114.

E. Myers Donald. Matrix formulation of co-kriging. *Journal of the International Association for Mathematical Geology*, 14(3):249–257, 1982. . URL https://eurekamag.com/research/019/383/019383781.php.

Radford M. Neal. Probabilistic inference using markov chain monte carlo methods. Technical report, 1993.

Antony M. Overstall and David C. Woods. Bayesian design of experiments using approximate coordinate exchange. *Technometrics*, 59(4):458–470, 2017. . URL https://doi.org/10.1080/00401706.2016.1251495.

Antony M. Overstall, David C. Woods, and Maria Adamou. acebayes: An R package for Bayesian optimal design of experiments via approximate coordinate exchange. *Journal of Statistical Software*, 95(13):1–33, 2020. .

Christopher Paciorek and Mark Schervish. Nonstationary covariance functions for gaussian process regression. In S. Thrun, L. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems*, volume 16. MIT Press, 2003. URL https://proceedings.neurips.cc/paper_files/paper/2003/file/326a8c055c0d04f5b06544665d8bb3ea-Paper.pdf.

Alessandro De Palma, Celestine Mendler-Dünner, Thomas Parnell, Andreea Anghel, and Haralampos Pozidis. Sampling acquisition functions for batch bayesian optimization, 2019.

Bandu Pamadi, Peter Covell, Paul Tartabini, and Kelly Murphy. Aerodynamic characteristics and glide-back performance of langley glide-back booster. *Collection of Technical Papers - AIAA Applied Aerodynamics Conference*, 2, 08 2004a. .

Bandu Pamadi, Paul Tartabini, and Brett Starr. *Ascent, Stage Separation and Glideback Performance of a Partially Reusable Small Launch Vehicle*. 2004b. . URL https://arc.aiaa.org/doi/abs/10.2514/6.2004-876.

Chiwoo Park and Jianhua Z. Huang. Efficient computation of gaussian process regression for large spatial data sets by patching local gaussian processes. 17(1):6071–6099, January 2016. ISSN 1532-4435.

Giulia Pedrielli, Songhao Wang, and Szu Hui Ng. An extended two-stage sequential optimization approach: Properties and performance. *European Journal of Operational Research*, 287(3):929–945, 2020. ISSN 0377-2217. . URL https://www.sciencedirect.com/science/article/pii/S0377221720303982.

V. Picheny, T. Wagner, and D. Ginsbourger. A benchmark of kriging-based infill criteria for noisy optimization. *Structural and Multidisciplinary Optimization*, 48, 09 2013. .

Christian Plagemann, Kristian Kersting, and Wolfram Burgard. Nonstationary gaussian process regression using point estimates of local smoothness. In Walter Daelemans, Bart Goethals, and Katharina Morik, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 204–219, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-87481-2.

Foster Provost, David Jensen, and Tim Oates. Efficient progressive sampling. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '99, page 23–32, New York, NY, USA, 1999. Association

for Computing Machinery. ISBN 1581131437. . URL https://doi.org/10.1145/312129.312188.

Ning Quan, Jun Yin, Szu Hui Ng, and Loo Hay Lee. Simulation optimization via kriging: a sequential search using expected improvement with computing budget constraints. *IIE Transactions*, 45(7):763–780, 2013. . URL https://doi.org/10.1080/0740817X.2012.706377.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2023. URL https://www.R-project.org/.

Dushhyanth Rajaram, Tejas G. Puranik, S. Ashwin Renganathan, WoongJe Sung, Olivia Pinon Fischer, Dimitri N. Mavris, and Arun Ramamurthy. Empirical assessment of deep gaussian process surrogate models for engineering problems. *Journal of Aircraft*, 58(1):182–196, 2021. . URL https://doi.org/10.2514/1.C036026.

Carl Rasmussen and Zoubin Ghahramani. Infinite mixtures of gaussian process experts. In T. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems*, volume 14. MIT Press, 2001. URL https://proceedings.neurips.cc/paper_files/paper/2001/file/9afefc52942cb83c7c1f14b2139b09ba-Paper.pdf.

Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian processes for machine learning*. Adaptive computation and machine learning. MIT Press, 2006. ISBN 026218253X.

Gareth O. Roberts and Jeffrey S. Rosenthal. Coupling and ergodicity of adaptive markov chain monte carlo algorithms. *Journal of Applied Probability*, 44(2):458–475, 2007. ISSN 00219002. URL http://www.jstor.org/stable/27595854.

Olivier Roustant, David Ginsbourger, and Yves Deville. DiceKriging, DiceOptim: Two R packages for the analysis of computer experiments by kriging-based metamodeling and optimization. *Journal of Statistical Software*, 51(1):1–55, 2012. URL https://www.jstatsoft.org/v51/i01/.

Kenneth J. Ryan. Estimating expected information gains for experimental designs with application to the random fatigue-limit model. *Journal of Computational and Graphical Statistics*, 12(3):585–603, 2003. ISSN 10618600. URL http://www.jstor.org/stable/1391040.

Hugh Salimbeni and Marc Deisenroth. Doubly stochastic variational inference for deep gaussian processes, 2017.

Thomas J. Santner, Brian J. Williams, and William I. Notz. *Physical Experiments and Computer Experiments*, pages 1–13. Springer New York, New York, NY, 2003. ISBN 978-1-4757-3799-8. . URL https://doi.org/10.1007/978-1-4757-3799-8_1.

Annie Sauer. *deepgp: Deep Gaussian Processes using MCMC*, 2022. URL https://CRAN.R-project.org/package=deepgp. R package version 1.1.0.

Annie Sauer, Robert B. Gramacy, and David Higdon. Active learning for deep gaussian process surrogates, 2021.

Annie Sauer, Andrew Cooper, and Robert B. Gramacy. Non-stationary gaussian process surrogates, 2023.

Alexandra M. Schmidt and Anthony O'Hagan. Bayesian Inference for Non-Stationary Spatial Covariance Structure via Spatial Deformations. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 65(3):743–758, 07 2003. ISSN 1369-7412. . URL https://doi.org/10.1111/1467-9868.00413.

Matthias Schonlau. *Computer Experiments and Global Optimization*. PhD thesis, CAN, 1997. AAINQ22234.

Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Md. Mostofa Ali Patwary, Prabhat Prabhat, and Ryan P. Adams. Scalable bayesian optimization using deep neural networks. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, page 2171–2180. JMLR.org, 2015.

A. Sóbester, S.J. Leary, and A.J. Keane. On the design of optimization strategies based on global response surface approximation models. *Journal of Global Optimisation*, 33:31–59, 2005. URL https://link.springer.com/article/10.1007/s10898-004-6733-1.

Niranjan Srinivas, Andreas Krause, Sham Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 1015–1022, Madison, WI, USA, 2010. Omnipress. ISBN 9781605589077.

Michael Stein. Large sample properties of simulations using latin hypercube sampling. *Technometrics*, 29(2):143–151, 1987. . URL https://www.tandfonline.com/doi/abs/10.1080/00401706.1987.10488205.

Yuji Sugita and Yuko Okamoto. Replica-exchange molecular dynamics method for protein folding. *Chemical Physics Letters*, 314(1):141–151, 1999. ISSN 0009-2614. . URL https://www.sciencedirect.com/science/article/pii/S0009261499011239.

Furong Sun, Robert B. Gramacy, Benjamin Haaland, Earl Lawrence, and Andrew Walker. Emulating satellite drag from large simulation experiments, 2019.

Robert Swendsen and Jian-Sheng Wang. Replica monte carlo simulation of spin-glasses. *Physical review letters*, 57:2607–2609, 12 1986. .

Shion Takeno, Hitoshi Fukuoka, Yuhki Tsukada, Toshiyuki Koyama, Motoki Shiga, Ichiro Takeuchi, and Masayuki Karasuyama. Multi-fidelity bayesian optimization with max-value entropy search and its parallelization, 2020.

William R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933. ISSN 00063444. URL http://www.jstor.org/stable/2332286.

Jie Tian, Ying Tan, Jianchao Zeng, Chaoli Sun, and Yaochu Jin. Multiobjective infill criterion driven gaussian process-assisted particle swarm optimization of high-dimensional expensive problems. *IEEE Transactions on Evolutionary Computation*, 23(3):459–472, 2019. .

Ryan Turner, David Eriksson, Michael McCourt, Juha Kiili, Eero Laaksonen, Zhen Xu, and Isabelle Guyon. Bayesian optimization is superior to random search for machine learning hyperparameter tuning: Analysis of the black-box optimization challenge 2020, 2021.

Mark Vogelsberger, Federico Marinacci, Paul Torrey, and Ewald Puchwein. Cosmological simulations of galaxy formation, 2019. URL https://arxiv.org/abs/1909.07976.

Konstantinos Vogklis and Isaac Lagaris. *A Rectangular Trust-Region Approach for Unconstrained and Box-Constrained Optimization Problems*, pages 562–565. 04 2019. ISBN 9780429081385. .

W. D. Vousden, W. M. Farr, and I. Mandel. Dynamic temperature selection for parallel tempering in Markov chain Monte Carlo simulations. *Monthly Notices of the Royal Astronomical Society*, 455(2):1919–1937, 1 2016. ISSN 0035-8711. . URL https://doi.org/10.1093/mnras/stv2422.

Jialei Wang, Scott C. Clark, Eric Liu, and Peter I. Frazier. Parallel bayesian global optimization of expensive functions, 2019.

Zi Wang and Stefanie Jegelka. Max-value entropy search for efficient bayesian optimization, 2017. URL https://arxiv.org/abs/1703.01968.

Ziyu Wang, Masrour Zoghi, Frank Hutter, David Matheson, and Nando De Freitas. Bayesian optimization in high dimensions via random embeddings. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI '13, page 1778–1784. AAAI Press, 2013. ISBN 9781577356332.

Ya-xiang Yuan. A review of trust region algorithms for optimization. *ICM99: Proceedings of the Fourth International Congress on Industrial and Applied Mathematics*, 09 1999.