University of Cape Town

Department of Statistical Sciences

Honours Thesis

Gaussian Processes with Applications in Bayesian Optimization

---

Author:
Benjamin Chiddy

Student Number:
CHDBEN002

Supervised by: Emeritus Professor Linda Haines & Jake Stangroom

October 24, 2023

Abstract

This project aims to understand the theoretical aspects of the Gaussian Process as a non-parametric statistical learning tool as well as its role as a surrogate function in Bayesian optimization. This is achieved through a comprehensive review of the literature associated with both the Gaussian Process and Bayesian optimization along with introductory examples on how they are used to conduct inference over functions. We then motivate the use of Bayesian optimization in real world applications where the collection of data is either computationally or commercially expensive through a series of diverse examples. This includes a discussion around the trade-off between exploitation and exploration of the solution space in search of the global optimum. After this we assess the performance of the Bayesian optimization algorithm across three analytical test functions and against several popular optimization routines using a Monte Carlo simulation. This results in the conclusion that Bayesian optimization outperforms most methods on function surfaces which are non-convex and multi-modal but is largely unnecessary on surfaces which are not. We end with a discussion around the benefits of Bayesian optimization in situations where we seek to optimize a function which has no analytical form or is either expensive or impossible to evaluate.

Acknowledgements

First and foremost I would like to thank both my supervisors, Emeritus Professor Linda Haines and Jake Stangroom for their support and guidance during my research. I would specifically like to thank Jake for introducing me to the idea of Bayesian optimization and inspiring me to pursue it as a research topic. Likewise, I want to thank Linda for without fail greeting my many knocks on her door with a smile and for the the insightful conversations which I learnt so much from. Secondly I would like to thank both Mark and Nori for their financial support during these past four years. Every day, I am grateful for the opportunity you have given me to pursue something that fills me with such immense passion. Thirdly, I would like to thank the lecturers and staff in the University of Cape Town Statistical Sciences Department who have all in someway contributed to this project and from whom I am constantly challenged to improve.

Lastly I would like to dedicate this project to my soon-to-be wife, Olivia. Nearly five years ago I was humbled by the enthusiasm you showed towards me returning to university as well as the many sacrifices you have made since. You are my constant source of motivation and without the love and support you have shown, none of this will have been possible.

- Ben

Contents

1 Introduction

In the ubiquitous *An Introduction to Statistical Learning* (James et al., 2013) the idea of statistical learning is described as the situation where one has observed a quantitative response $Y \subset \mathbb{R}$ and p different predictors $\mathbf{X} = (X_1, X_2, \ldots, X_p) \subset \mathcal{X}$. We then assume that there is some relationship between Y and X such that

$$Y = f(X) + \epsilon \tag{1.1}$$

where f is some fixed but unknown function of X and ϵ is a zero-mean error term that is independent of X. In this setting the function $f : \mathcal{X} \to \mathbb{R}$ describes an underlying relationship between X and Y through the mapping of some input space $\mathcal{X}$ to the reals. Statistical learning can then be described as *"the set of approaches for estimating f"* or in other words, modelling a function $\hat{f}$ such that $Y \approx \hat{f}(X)$ given some dataset $D_n = (X, Y)$. James et al. (2013) go on to categorized these approaches as being either *parametric* or *non-parametric*.

Parametric methods of statistical learning involve making an assumption about the functional form of f. For example, that it is linear in X such that

$$f(X) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p. \tag{1.2}$$

This greatly simplifies the modelling process as it does not require us to directly estimate the function f but rather the $p + 1$ coefficients $\beta_0, \beta_1, ..., \beta_p$, a far easier task according to James et al. (2013). Although the assumption of linearity lacks the expressive power to describe complex relationships between X and Y it is able to generalize in a way which is easily interpretable. Other parametric methods such as polynomial regression or neural-networks allow us to describe complex relationships more accurately but are difficult to work with in practice and lose in interpretability what they gain in complexity (Rasmussen, 2004).

In contrast to their parametric counterparts, non-parametric methods make no explicit assumptions about the functional form of f and instead attempt to *learn* it directly from the data. This approach offers the advantage of flexibility, as it permits the modeling of a far wider range of potential shapes for $\hat{f}$. This flexibility is particularly valuable in scenarios where we suspect the existence of complex and possibly non-linear relationships between Y and X. Popular techniques such as regression splines allow us to fit datasets to a high degree of accuracy but require careful tuning to mitigate the risk of *overfitting*. An alternative approach is to model the function f probabilistically as a stochastic process, specifically a Gaussian Process. As highlighted in the widely cited tutorial *Gaussian Processes in Machine Learning* (Rasmussen, 2004) this approach enables the formulation of a Bayesian framework for regression and is often preferred due to its ability to quantify uncertainty in our estimate for f.

As the title of his book (*Surrogates: Gaussian Process Modeling, Design, and Optimization for the Applied Sciences*) suggests, Gramacy (2020) refers to the probabilistic model of f as a surrogate. This concept is popular within materials science literature where a surrogate is thought to be a mathematical model which mimics the behaviour of a complex system

to a reasonable degree of accuracy while being both computationally and commercially less expensive to evaluate (Thombre et al., 2015). Examples of this include modelling the dynamics of a rocket booster (Gramacy, 2020), results of a marketing campaign or even in the hyperparameter tuning of a large machine learning model (Shahriari et al., 2016). In each of these cases we have a quantitative response Y and a set of predictors X which are in some way expensive to obtain. It is therefore within our interest to model the relationship between these quantities with as little evaluations of the function f as possible (eg. running a rocket booster, implementing a marketing campaign or training a large machine learning model). As previously mentioned, surrogate models allow us to probabilistically model f such that we can can run computer experiments or simulations on the far cheaper to evaluate surrogate $\hat{f}$. The Gaussian Process is one of the most popular such surrogate models (Gramacy, 2020) and is the topic of this project.

In what follows we provide an in-depth review regarding the theory and literature associated with the Gaussian Process. Within this we describe how it was first proposed by the South African born Krige (1951) in the context of spatial statistics but later shown to enable Bayesian inference over functions by Rasmussen (2004). After this we investigate one of the most popular uses of the Gaussian Process which is as a surrogate function in Bayesian optimization. Within this section we see how balancing the competing objectives of exploration and exploitation can lead to efficient optimization of non-convex and multi-modal function surfaces in situations where function evaluations are either commercially or computationally expensive. We then benchmark the Bayesian optimization algorithm against several popular numerical optimization routines to gauge its performance on a variety of analytical test functions. Finally, we end with a summary of our results and a discussion on the strengths and weaknesses of the Bayesian optimization algorithm.

2 A Theoretical Review on the Gaussian Process

2.1 Definition of a Gaussian Process

Although a somewhat recent development in the context of machine learning (Williams and Rasmussen, 1995) the idea of probabilistically modelling f has its origins in mid-century spatial statistics (Krige, 1951) where f was considered to be a physical process. In this context we are interested in predicting the value of f at an *unobserved* location $\mathbf{x}_*$ using n observations $\boldsymbol{f} = [f(\mathbf{x_1}), f(\mathbf{x_2}), \ldots, f(\mathbf{x_n})]^T$ obtained at n locations $\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_n}$ within a study area (or input space $\mathcal{X}$). In his masters thesis Ngwenya (2011) describes the spatial process as

$$F(\cdot) = \{F(\mathbf{x}) : \mathbf{x} \in \mathcal{X} \subset \mathbb{R}^p\} \tag{2.1}$$

operating in p dimensions over the study area. Ngwenya then outlines the practical problem of predicting $F(\cdot)$ at $\mathbf{x}_* \in \mathcal{X}$ using $\boldsymbol{f} = [f(\mathbf{x_1}), f(\mathbf{x_2}), \ldots, f(\mathbf{x_n})]^T$ where we assume $\boldsymbol{f}$ is a realization of the n random variables $\mathbf{F} = [F(\mathbf{x_1}), F(\mathbf{x_2}), \ldots, F(\mathbf{x_n})]^T$. This problem is colloquially known as *Kriging* after its originator Danie Krige who in his 1951 masters thesis used it to predict the location of gold reserves in the Witwatersrand region of South Africa. While a physical process is typically modeled in 2 or 3 dimensions, there are no practical limitations when considering more than 2 inputs ($p > 2$).

When we consider the random variables $\mathbf{F}$ to be jointly Gaussian, the process is known as a Gaussian Process - formally defined as follows:

> *"A Gaussian Process is a collection of random variables, where any finite subset has (consistent) joint Gaussian distributions."* (Rasmussen, 2004)

2.2 Generalization of the Gaussian Distribution

A helpful way to think about the Gaussian Process is as a generalization of the Gaussian distribution. While the Gaussian distribution is defined by its finite-dimensional mean vector and covariance matrix, the Gaussian Process extends this concept into an infinite-dimensional space (specifically Hilbert space) where it is defined over functions (Rasmussen and Williams, 2006). In this infinite-dimensional space the Gaussian Process is characterized by its first two moments, both of which are themselves now functions. We describe a Gaussian Process as

$$f \sim \mathcal{GP}(m, K) \tag{2.2}$$

meaning the random **function** f is distributed as a Gaussian Process with mean function m and covariance function (also known as a *kernel*) K such that

$$\begin{aligned} m : \mathcal{X} \to \mathbb{R} &= \mathbb{E}[f(\mathbf{x})] \\ K : \mathcal{X} \times \mathcal{X} \to \mathbb{R} &= \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))] \end{aligned} \tag{2.3}$$

The mean function m encodes the central tendency of the function and is often assumed to stationary. In spatial statistics literature this assumption is known as *Ordinary Kriging*

when the value of $m(\mathbf{x})$ is unknown and *Simple Kriging* when $m(\mathbf{x})$ is known (usually taken to be 0). The covariance function K encodes information about the shape and structure we expect the function to have, or in other words how every location $\mathbf{x}$ is related to every other location $\mathbf{x}'$ (Garnett, 2023). A common choice is the squared exponential kernel, defined as

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\|\mathbf{x} - \mathbf{x}'\|^2\right). \tag{2.4}$$

What this tells us is that the covariance between $\mathbf{x}$ and $\mathbf{x}'$ (defined as Euclidean distance in 2.4) decays exponentially fast as $\mathbf{x}$ and $\mathbf{x}'$ become further apart (Gramacy, 2020). Note that $K(\mathbf{x}, \mathbf{x}) = 1$ and $K(\mathbf{x}, \mathbf{x}') < 1$ for $\mathbf{x} \neq \mathbf{x}'$. In addition, just as with the covariance matrix of a Gaussian distribution we require that $K(\mathbf{x}, \mathbf{x}')$ be *positive definite* and *symmetric* such that

$$\mathbf{x}^T K \mathbf{x} > 0 \ \forall \ \mathbf{x} \neq 0 \text{ and } K(\mathbf{x}, \mathbf{x}') = K(\mathbf{x}', \mathbf{x}) \text{ respectively.} \tag{2.5}$$

Besides the squared exponential there are many other types of kernels which share the above properties, most notably the Matérn which is used extensively in the context of machine learning (Gramacy, 2020). In what follows our focus will be on the squared exponential kernel due to its relative ease-of-use and flexibility. We encourage readers to consult page 2-12 of Ngwenya (2011) for specifications of several other popular kernels.

2.3 Distribution over Functions

Although Gaussian Processes are infinite-dimensional objects, by definition any finite subset has a joint Gaussian distribution. This is of great practical importance as it allows us to draw samples from f by simply computing the related distribution at n locations within $\mathcal{X}$. Consider the following example which is an adaption on one which is seen in *Gaussian Processes in Machine Learning* (Rasmussen, 2004). Suppose we have the simple one dimensional process defined as

$$f \sim \mathcal{GP}(m, K) \text{ where } m = 2\mathbf{x} \cdot \cos(\mathbf{x}) \text{ and } K(\mathbf{x}, \mathbf{x}') = \exp\left(-\|\mathbf{x} - \mathbf{x}'\|^2\right) \tag{2.6}$$

Given a set of $\mathbf{x} \in \mathbb{R}$ we can compute a vector of means and covariance matrix using m and K, the result of which will be a Gaussian distribution with

$$\begin{aligned} \mu_i &= m(x_i) = 2x_i \cdot \cos(x_i) \ \ i = 1, 2, ..., n \\ \Sigma_{ij} &= K(x_i, x_j) = \exp\left(-\|x_i - x_j\|^2\right) \ \ i, j = 1, 2, ..., n \end{aligned} \tag{2.7}$$

If we now draw a random sample from $f|\mathbf{x} \sim \mathcal{N}_n(\boldsymbol{\mu}, \Sigma)$ the corresponding vector will have as its coordinates the function values $f(x)$ for each corresponding $x_1, x_2, ..., x_n$.

In Figure 1 we can see 100 random realizations of the Gaussian Process defined in Equation 2.6 and shows why Gaussian Processes are often referred to as *distributions over functions*. This feature makes them an ideal choice of prior for Bayesian inference on f which is the topic of the next section.

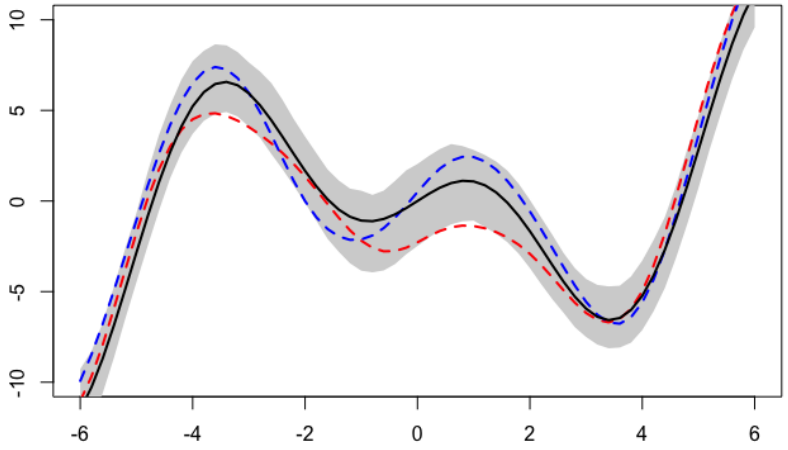


Figure 1: Two random realizations of f shown as dotted red and blue lines drawn from the Gaussian Process defined in Equation 2.6 whose true form is shown as a solid black line. The shaded gray region represents 95% confidence intervals from 100 similar realizations of f.

2.4 Bayesian Inference on f using Gaussian Processes

We may now ask ourselves: given a *training* set $D_n = (\mathbf{x}_n, f(\mathbf{x}_n))$ and a collection of unseen locations $\mathbf{x}_* \in \mathcal{X}$, what are the plausible realizations of $\boldsymbol{f}$ that could explain the observed values? In essence, our focus lies in understanding the conditional distribution of $f(\mathbf{x}_*)|D_n$. If we regard $f(\mathbf{x})$ to be the prior, as discussed in Section 2.3 then $f(\mathbf{x}_*)|D_n$ naturally becomes the posterior as emphasized by Gramacy (2020). The computation of this posterior distribution allows us to refine our initial beliefs on the form of f, represented by the prior in light of the training data. Consequently, we gain the ability to make inference on the underlying functional form of f across a set of unseen test cases. This objective aligns closely with the core principles of statistical learning, as was outlined in our introduction.

2.4.1 Gaussian Process Posterior

For notational convenience, let $f(\mathbf{x}) \equiv \boldsymbol{f}$ be the *known* function values of the training cases $\mathbf{x} \in \mathcal{X}$ and $f(\mathbf{x}_*) \equiv \boldsymbol{f_*}$ be the function values corresponding to the *unseen* test cases $\mathbf{x}_* \in \mathcal{X}$. By definition, each of these quantities have a joint Gaussian distribution such that $\boldsymbol{f} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$ and $\boldsymbol{f}_* \sim \mathcal{N}(\boldsymbol{\mu}_*, \Sigma_*)$ where $\boldsymbol{\mu} = m(x_i),\ i = 1, 2, ..., n$ is the vector of training means and likewise $\boldsymbol{\mu}_*$ is the vector of test means. Equivalently $\Sigma = K(x_i, x_j),\ i, j = 1, 2, ..., n$ is the training set covariance matrix while Σ_* is the test set covariance matrix. Recall that by Equation 2.5 these are both valid covariance matrices. We can now write out the joint distribution between $\boldsymbol{f}$ and $\boldsymbol{f}_*$ as

$$\begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{f}_* \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu} \\ \boldsymbol{\mu}_* \end{bmatrix}, \begin{bmatrix} \Sigma & \Sigma_* \\ \Sigma_*^T & \Sigma_{**} \end{bmatrix} \right). \tag{2.8}$$

Since we are interested in the function values at the unseen test locations conditional on the training data we can derive the following expression using the laws of the multivariate-normal distribution,

$$\boldsymbol{f}_*|\boldsymbol{f} \sim \mathbf{N}(\mu_* + \Sigma_*^T\Sigma^{-1}(\boldsymbol{f} - \boldsymbol{\mu}),\ \Sigma_{**} - \Sigma_*^T\Sigma^{-1}\Sigma_*). \tag{2.9}$$

This is the posterior distribution for any set of test cases $\mathbf{x}_* \in \mathcal{X}$ (Rasmussen, 2004). The corresponding **posterior** Gaussian Process is therefore

$$\begin{aligned} f|D &\sim \mathcal{GP}(m_D, K_D) \\ m_D(x) &= m(x) + \Sigma(\mathbf{x}, x)^T\Sigma^{-1}(\boldsymbol{f} - \boldsymbol{m}) \\ K_D &= K(x, x') - \Sigma(\mathbf{x}, x)^T\Sigma^{-1}\Sigma(\mathbf{x}, x') \end{aligned} \tag{2.10}$$

where $\Sigma(\mathbf{x}, x)$ is the vector of covariances between every training case and $x_1, x_2, ..., x_n$ (Rasmussen, 2004). Notice that $m_D(x)$ is a linear predictor, in fact it is the Best Linear Unbiased Predictor (BLUP) for $\boldsymbol{f}_*$ as proven by Ngwenya (2011). Also note that the posterior variance K_D is equal to the prior variance minus a positive term meaning we have indeed *learnt* something from the training data (Gramacy, 2020).

Let's pause for a moment and appreciate the elegant simplicity of this line of thinking. As demonstrated by Equation 2.9 our path to Bayesian inference on an infinite-dimensional stochastic process requires merely the computation of a conditional Gaussian distribution. This insight allows us to make predictions based solely on data, without the need to *fit* any parameters or minimize some loss criterion. This is why Gaussian Processes are so highly regraded as a non-parametric regression tool (Gramacy, 2020). The equations in 2.10 are what's known as the *Kriging* equations. They were first derived by Danie Krige in 1951, some 40 years before Williams and Rasmussen (1995) would reinterpret them through a Bayesian perspective in the context of machine learning.

2.4.2 1D Prediction Example

We now return to our earlier example described in Equation 2.6. Instead of generating random samples from f, we now obtain $n = 9$ training cases from the area of interest within our input space $(-6, 6) \subset \mathcal{X} = \mathbb{R}$. We then use the *Kriging* equations given in 2.10 to obtain the posterior distribution for all *unseen* test cases within the input space. Figure 2 below shows 100 sample draws from this posterior distribution superimposed over the true response from f for these test locations (shown in black). Notice the sausage shape that the distribution of these samples take the further they are from the sampled locations. This is due to the fact that $K(x, x) = 1$ and $K(x, x') \to 1$ as $x' \to x$ which tells us we are interpolating the space between each of these sampled locations (Gramacy, 2020). This is known as *honouring* the training data in the context of geostatistics and is a reason why Gaussian Processes are also popular tools in the modelling of large scale computer experiments where a quantification of uncertainty is often required (Santner et al., 2003). Finally note that towards the edge of the study area, where we have less training examples, our posterior samples start to diverge from the true value of f. This warns us against making predictions at locations too far outside of the area of focus.

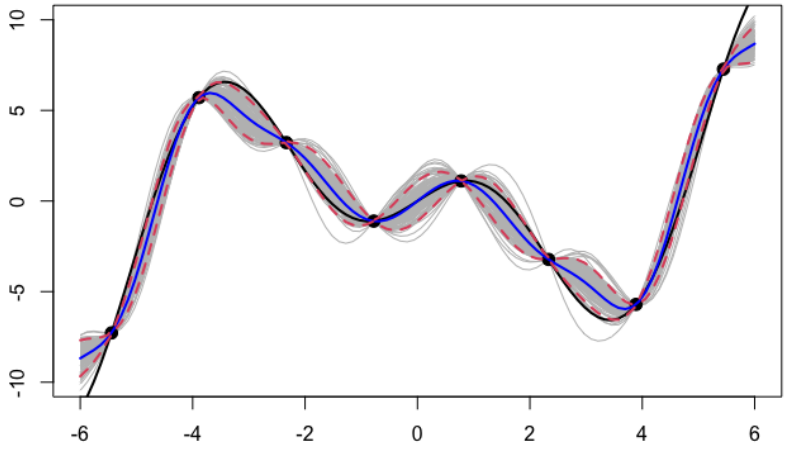


Figure 2: 100 samples from the posterior distribution of Equation 2.6 shown in grey with the posterior mean m_D shown in blue and the true response in black. 95% credibility intervals for m_D are shown as dotted red lines and the $n = 9$ training cases are shown as black dots.

2.5 Training a Gaussian Process

In the previous section we saw how a Gaussian Process can be used within a Bayesian framework to make inference on an unknown function f. We did this by specifying a prior mean and covariance function which, when combined with training data, allowed us to compute a posterior distribution. However, more often than not we cannot specify an *informative* prior as was done in Equation 2.6. Instead we want to be able to choose prior mean and covariance functions in light of the data, this is referred to as model *training* by Rasmussen (2004). To achieve this we typically specify a *hierarchical* prior under an empirical Bayes scheme where the mean and covariance functions are parameterised by a set of hyperparameters $\boldsymbol{\theta}$.

Now, one might wonder why we are introducing parameters despite the earlier emphasis on non-parametric techniques. What this ultimately comes down to is a matter of perspective. In spatial statistics literature, Kriging is considered a parametric model. In this context one would specify a covariance structure (known as a semivariogram) and fit its parameters either through ordinary least squares or maximum likelihood estimation. Surprisingly (or perhaps confusingly) this leads to the posterior Gaussian Process equations seen in 2.10 without any Bayesian reasoning. Our focus will remain on the Bayesian interpretation given by Rasmussen (2004) but again we encourage the reader to consult Ngwenya (2011) for a detailed breakdown on the parametric approach in the context of spatial statistics.

2.5.1 Hyperparameters of the Squared Exponential Prior

Recall the squared exponential kernel we introduced in Equation 2.4. What this prior encodes is our belief that "covariance decays exponentially fast as $\mathbf{x}$ and $\mathbf{x}'$ become farther

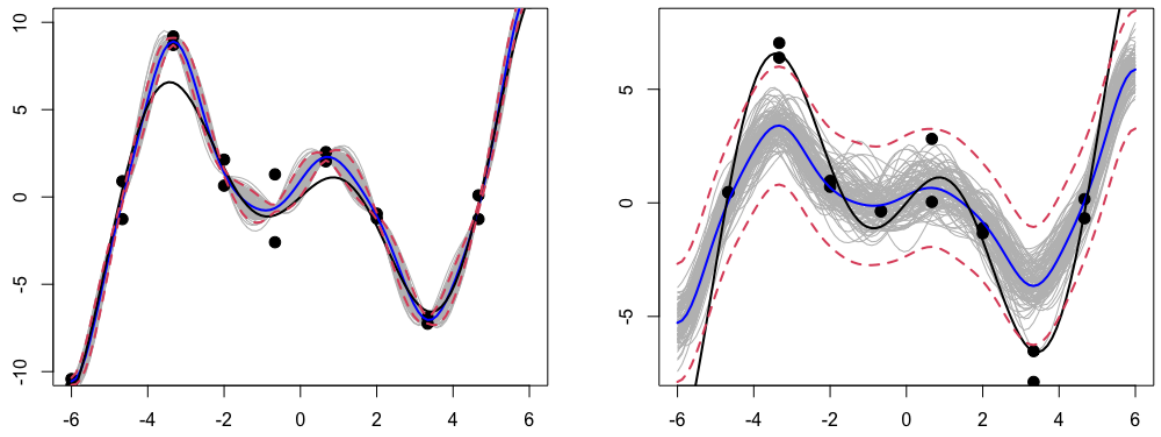
apart" (Gramacy, 2020). This kernel is parameterised it in the following way

$$K(x, x') + \sigma^2 = \tau^2 \cdot \exp\left\{-\frac{||x - x'||^2}{2\lambda}\right\} + \sigma^2\delta_{ii} \tag{2.11}$$

therefore giving $\boldsymbol{\theta} = \{\tau^2, \lambda, \sigma^2\delta_{ii}\}$. Note that it is common in most applications to assume that $m(x) = 0$ *a priori* (also known as *Simple Kriging*) which is what we will do here. Below we provide some intuitions for each of the hyperparameters contained within $\boldsymbol{\theta}$:

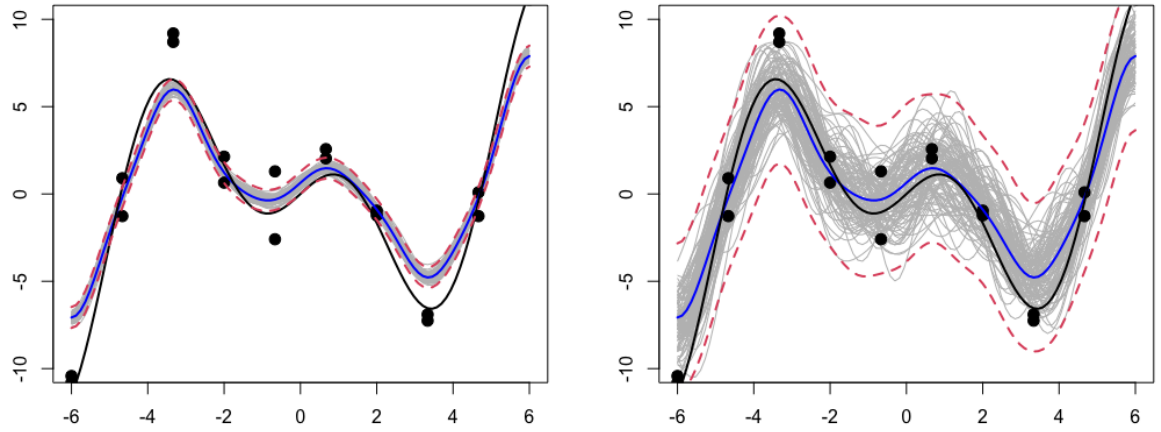
- **Scale** τ^2 - Also known as the *sill* in the context of *Kriging*. Controls the maximum covariance between any two observations x_i and x_j where $i, j = 1, 2, ...n$.
- **Length-scale** λ - Also known as the *range* in the context of *Kriging*. Controls the rate-of-decay or how far reaching the effect each observation x_i has on every other observation x_j where $i, j = 1, 2, ...n$.
- **Noise-variance** σ^2 - Also known as the *nugget* in the context of *Kriging*. An extra source of variation applied individually to each observation where δ_{ii} is the Kronecker delta. This accounts for possible noise in our training data.

Recall our running example from Equation 2.6 which we now augment with Gaussian noise such that $\mathbf{y} = f(\mathbf{x}) + \epsilon$ where $\epsilon \sim \mathcal{N}(0, 1)$. To illustrate the effect each hyperparameter has on the posterior distribution, consider Figures 3, 4 and 5 below. As before, the true value for f is shown in black, with 100 realizations drawn from the posterior distribution are shown in grey. 95% credibility intervals shown as dotted red lines while the posterior mean is shown in blue.



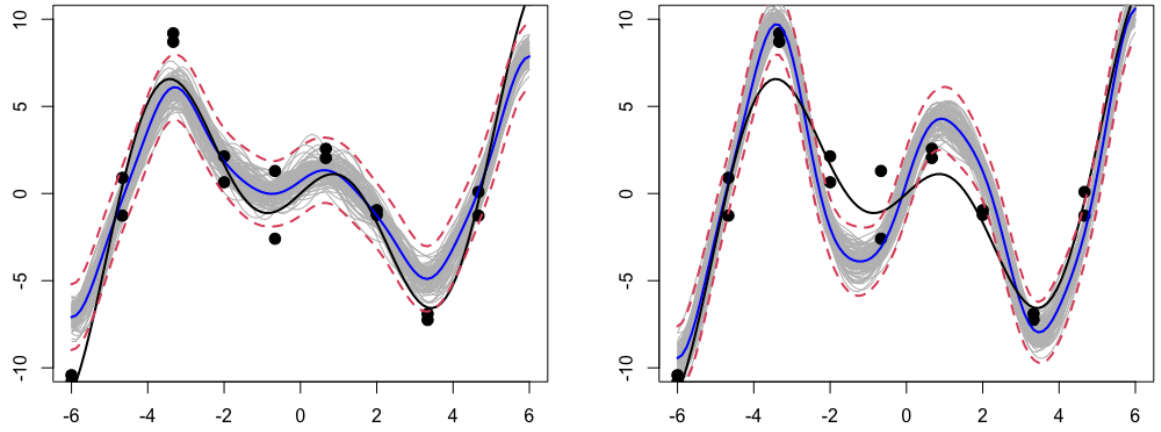
(a) $\boldsymbol{\theta} = \{\tau^2 = 1, \lambda = 1, \sigma^2 = 0.01\}$ **(b)** $\boldsymbol{\theta} = \{\tau^2 = 1, \lambda = 1, \sigma^2 = 1\}$

Figure 3: Effect of changing σ^2 on the posterior distribution. Notice that small values for σ^2 do not account for noisy observations and try to fit the noisy data exactly. Appropriate values for σ^2 correctly account for the uncertainty in our observations.



(a) $\boldsymbol{\theta} = \{\tau^2 = 0.1, \lambda = 1, \sigma^2 = 1\}$ **(b)** $\boldsymbol{\theta} = \{\tau^2 = 6, \lambda = 1, \sigma^2 = 1\}$

Figure 4: Effect of changing τ^2 on the posterior distribution. Small values for τ^2 limit the maximum effect each observation can have an other observations leading to smaller covariances. This creates posterior samples which have low amplitudes (are less wiggly) while larger values for τ^2 lead to posterior samples which have higher amplitudes (are more wiggly).



(a) $\boldsymbol{\theta} = \{\tau^2 = 1, \lambda = 0.01, \sigma^2 = 1\}$ **(b)** $\boldsymbol{\theta} = \{\tau^2 = 1, \lambda = 7, \sigma^2 = 1\}$

Figure 5: Effect of changing λ on the posterior distribution. Small values for λ slow the rate-of-decay and means that observations which are further apart will have larger effects on each other leading to smoother, more slowly changing surfaces. Large values for λ increase the rate-of-decay meaning only close-by observations can effect each other leading to less smooth/ rapidly changing surfaces.

These figures show that choosing appropriate hyperparameters has a large impact on the accuracy of the posterior distribution. Next we investigate how to *optimally* select these values in a fairly simple manner. Another reason why Gaussian Processes are such a popular modelling tool.

2.5.2 Fitting the Hyperparameters

Using an empirical Bayes approach allows us to specify vague prior information in a simple way by computing the probability of the data given the hyperparameters $\boldsymbol{\theta}$. This is known as the *marginal likelihood* and thankfully can be achieved analytically under the assumption of the data being Gaussian. This is shown by Garnett (2023) by assuming the hierarchical prior specification $\pi(f|\boldsymbol{\theta}) = \mathcal{GP}(0, K)$. We now want to measure the quality of fit to our training data $D = (X, \mathbf{y})$. Therefore through an application of Bayes theorem we have that

$$\begin{aligned} \pi(\mathbf{y}|X, \boldsymbol{\theta}) &= \int \pi(\mathbf{y}|f) \cdot \pi(f|X, \boldsymbol{\theta})\, df \\ &= \int \mathcal{N}(f, \sigma^2 \mathbb{I}_n) \cdot \mathcal{N}(0, K(X, X; \boldsymbol{\theta}))\, df \\ &= \mathcal{N}(0,\ K(X, X; \boldsymbol{\theta}) + \sigma^2 \mathbb{I}_n) \end{aligned} \tag{2.12}$$

where $\mathbb{I}_n$ is the $n \times n$ identity matrix and σ^2 is the nugget described in Section 2.5.1. Now if we let $\Sigma = K(X, X; \boldsymbol{\theta}) + \sigma^2 \mathbb{I}_n$ we obtain the *marginal* log-likelihood of our data under the chosen prior as being

$$\log(\pi(\mathbf{y}|X, \boldsymbol{\theta})) = -\frac{1}{2} \log |\Sigma| - \frac{1}{2} \mathbf{y}^T \Sigma^{-1} \mathbf{y} - \frac{n}{2} \log(2\pi). \tag{2.13}$$

Appropriate hyperparameter settings can then be found through maximisation of Equation 2.13 using a numerical optimization routine available in most statistical software packages.

2.5.3 Automatic Regularization

Equation 2.13 comprises of three terms which Rasmussen (2004) describes in the following way: the first is a *complexity penalty term* that penalizes overly complex model specifications. The second term, which is the only term dependent on the training set output values $\mathbf{y}$ is considered to be a data fit measure. The third term is a log normalization term which is independent of the data and is humorously described as "not interesting" by Rasmussen. The data fit term is large when the data fit the model well and the complexity term is large when the "volume of the prior covariance is small", or in other words the model is simpler (Garnett, 2023). This means that when we are fitting the hyperparameters of a Gaussian Process model there is an "automatic" trade-off between complexity and data-fit. Regularisation is automatic as there is no weighting parameter which needs to be set by some form of cross validation, another feature of great practical importance as it greatly simplifies training (Rasmussen, 2004). This trade-off is also known as *Occam's Razor* and ensures that more parsimonious models are preferred in a process similar to Bayesian Model selection (de Benito Delgado and Wacker, 2015).

2.6 Challenges of working with Gaussian Processes

The primary challenge when working with Gaussian Processes is its computational complexity. We can see that in Equation 2.10 that the computation of the posterior Gaussian Process involves the inversion of an $n \times n$ covariance matrix. Therefore, standard implementations have a time complexity of $\mathcal{O}(n^3)$ and thus when N is sufficiently large become infeasible. Gramacy (2020) suggests that the Gaussian Process can be ruled out at $N > 2000$ meaning that their use is large scale systems is severely limited. Early solutions to this scalability issue include using tapered covariance functions, low rank approximations, model surrogates and many more (Gramacy, 2020). However, these methods often provide poor approximations to the actual structure of the data and because of this modern solutions instead focus on parallelizing the computation. In addition to this, Gaussian Processes perform poorly when data is non-stationary. This limits their use when predicting real time datasets or capturing evolving dynamics. Being a non-parametric method, it is often robust enough to be used as is - but will mostly lead to poor results. This problem is solved through partitioning a data set into smaller non-stationary regions and then fitting each partition with its own model. By the assumption that each partition is independent, it can be computed in parallel with a reduced $\mathcal{O}(m^3)$ where $m < n$.

2.7 Applications of Gaussian Processes

Gaussian process are an extremely flexible class of supervised learning model. Their ability to make predictions while also providing a quantification of uncertainty has made them extremely popular in many scientific fields. Apart from its origins in spatial statistics, other applications include most classical regression and classification problems, conducting numerical analysis such as probabilistically solving differential equations (Hennig et al., 2015), and modelling a diverse set of topics ranging from financial markets (Gonzalvez et al., 2019) to black holes (Griffiths, 2023). An application that has gained traction in recent years is in the field of optimization, where Gaussian Processes are used as the *"backbone"* of modern Bayesian optimization (Gramacy, 2020), the theory of which is the focus of the following section.

3 Gaussian Processes in Bayesian Optimization

3.1 Optimization of Expensive Functions

Most industries and scientific disciples are interested in making an optimal set of decisions in regards to some problem. Carrying on from the examples introduced in Section 1; engineers might want to find the optimal mixture of fuels which produce maximum thrust inside a rocket engine, a company may be interested an advertising campaign that maximizes return on investment or a data scientist may wish to uncover an optimal hyper-parameter setting to optimize the performance of a machine learning model. Answering questions such as these is a complex task which requires making multiple high-dimensional decisions, many of which interact. In addition to this, these problems are challenging in the sense that gathering experimental observations are costly, either in terms of commercial or computational cost meaning that experimental datasets are often small. Furthermore, these problems are highly noise-corrupted making modelling, and subsequent optimization difficult. This can be expressed mathematically as finding a parameter setting $\mathbf{x}$ such that an unknown objective function f is at its global maximum (or conversely minimum). That is, we wish to find $\mathbf{x}^*$ such that

$$\mathbf{x}^* = \underset{\mathbf{x} \in \mathcal{X}^d}{\operatorname{argmax}} f(\mathbf{x}) \tag{3.1}$$

where $\mathcal{X}^d$ is some parameter (or feature) space of interest having dimension d, representing the amount of decisions that must be made. In regards to Bayesian optimization Shahriari et al. (2016) make the further distinction that f is a *black-box* function for which we have no simple analytical expression, but can be evaluated at any $\mathbf{x} \in \mathcal{X}^d$. However, these evaluations are noise-corrupted and produce outputs $y \in \mathbb{R}$ such that $\mathbb{E}[y|f(\mathbf{x}] = f(\mathbf{x})$. Furthermore, evaluations of f are considered *expensive* in some sense and therefore we are interested in solving Equation 3.1 using as little evaluations of f as possible. With this in mind, we may want to formulate an alternative problem

$$\mathbf{x}^* = \underset{\mathbf{x} \in \mathcal{X}^d}{\operatorname{argmax}} \hat{f}(\mathbf{x}) \tag{3.2}$$

where $\hat{f}$ is surrogate model that is cheaper to evaluate and can be numerically optimized. An ideal choice of surrogate is the Gaussian Process as it allows us to closely approximate f in a data efficient manner with the added benefit of providing a quantification of uncertainty in our approximation (Shahriari et al., 2016). This idea was popularized by Booker et al. (1999) under the name *surrogate assisted optimization* but its origins can be dated back to Močkus (1975) in his paper *"The application of Bayesian methods for seeking the extremum"*. Although, instead of the Gaussian Process Močkus makes use of linear models to approximate the function f.

3.2 Leveraging Active Learning

By solving Equation 3.2 we are merely approximating or proposing a potential solution to Equation 3.1. It therefore makes sense to leverage Equation 3.2 to *sequentially* solve Equation 3.1 - at each step obtaining a slightly improved approximate answer.

For situations where the evaluation of f is expensive, this approach is far more considered as it enables us to use each evaluation to inform the next thus managing the evaluation budget more efficiently. In classical statistics literature this process is know as sequential design, but has more recently come to be known as *Active Learning* in the machine learning community (Gramacy, 2020). This is in contrast to *one-shot* or *single-batch* design in which we query f at n locations within the feature space simultaneously. Instead, we can use the information from each evaluation of f to inform the next. Shahriari et al. (2016) describe this process as a "sequential search algorithm" where at each step n, we fit a surrogate model $\hat{f}$ and use it to select $\mathbf{x}_{n+1}$ at which to evaluate f and obtain y_{n+1}. This search continues for N steps or until the search budget (eg. compute cycles or capital) has been exhausted at which point the algorithm returns the final recommendation $\mathbf{x}^*$ which represents its best known solution to Equation 3.1.

Fundamentally, at each step of this algorithm we propose a prior belief over possible function configurations for f and then sequentially update this model via Bayesian posterior updating (Shahriari et al., 2016). Having read Section 2.3 we can see why the ideal choice of prior, and therefore the ideal surrogate model is the Gaussian Process. Below we summarise the process of Active Learning in a graphic inspired by a similar one created by Gramacy (2020).

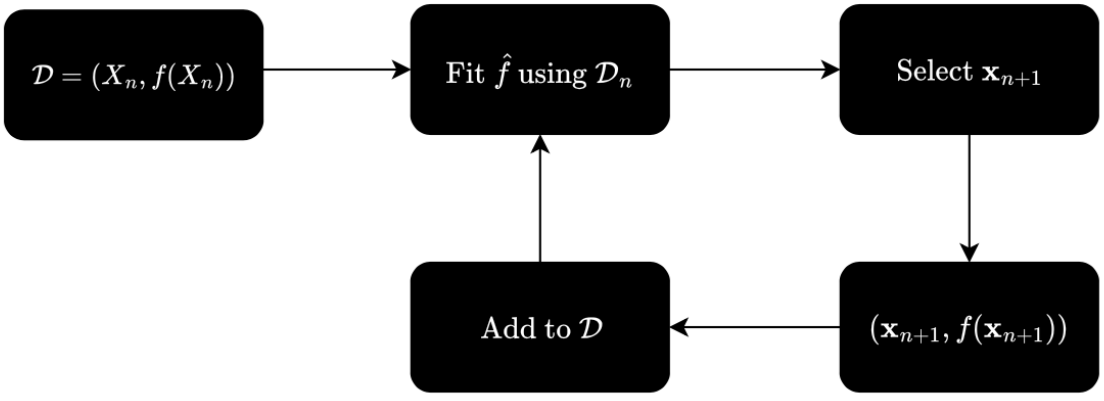


Figure 6: Flowchart describing a general Active Learning procedure.

This shows how active learning can be leveraged to sequentially propose increasingly improved solutions to Equation 3.1. In the following section we discuss methods for the selection of $\mathbf{x}_{n+1}$ in a way which maximises the utility of this process.

3.3 Trade-off between Exploitation and Exploration

In the previous section we saw how the surrogate model $\hat{f}$ can be used to sequentially provide an approximate solution to the optimization problem defined in Equation 3.1. However, since $\hat{f}$ represents a probabilistic model for f limited to the previously evaluated locations it would be naive to simply maximise $\hat{f}$ at each step of the algorithm as was defined in Equation 3.2. Since most real world problems (like those described in Section 2.2.2) are non-convex and multimodal, this strategy might lead to the algorithm getting stuck in local extrema. This is known as over-exploitation and limits the algorithm from exploring areas of the search space which may provide better solutions to Equation 3.1. Instead we might to to leverage the uncertainty in our surrogate model to guide this exploration in the hope of discovering a better, possibly global extrema. The trade-off between exploitation and exploration is balanced through what is known as an **acquisition** or **infill** function which encodes a policy for how we want our algorithm to progress through the search space. This can be defined mathematically as,

$$\mathbf{x}^*{}_{n+1} = \underset{\mathbf{x}\in\mathcal{X}^d}{\operatorname{argmax}}\, \mathcal{J}(\mathbf{x}) \tag{3.3}$$

where $\mathcal{J}$ defines our acquisition function otherwise known as a search policy. There exists a rich literature on acquisition functions in both experimental design, where it is known as expected utility and Bayesian optimization. Shahriari et al. (2016) define 4 broad classes into with each strategy can be categorised. The characteristics of each of these acquisition functions are as follows:

1. **Optimistic Policies**: Being optimistic in the face of uncertainty or favouring exploration over exploitation of the search space.
2. **Improvement-based Policies**: Favour points that are predicted to improve upon the current best solution or striking a balance between exploration and exploitation.
3. **Information-based Policies**: Considers the posterior distribution over the choices of $\mathbf{x}_{n+1}$ to maximise information gained or minimise entropy.
4. **Portfolio Policies**: Considers a range of policies and uses a meta-criterion (acquisition function at a higher level) to select the best strategy at each search step.

In what follows we explore examples of the first two categories but encourage the reader to consult Shahriari et al. (2016) for examples of the other two.

3.3.1 Mean Criterion

We firstly consider the naive policy of simply maximizing (or minimizing) the posterior mean (defined in Equation 2.10) at each step of the algorithm. This amounts to aggressive exploitation of the search space and although can work well in many cases, often leads to the search getting stuck in some local extrema. Gramacy (2020) introduces this criterion as an illustrative example of something that should be improved upon which is the reason for its inclusion here. Mathematically the mean criterion is described as,

$$\mathbf{x}^* = \underset{\mathbf{x}\in\mathcal{X}^d}{\operatorname{argmax}}\{\mu(\mathbf{x})\} \tag{3.4}$$

where $\mu(\mathbf{x})$ refers to the posterior mean of f as defined in Equation 2.10. To avoid confusion with Monte Carlo, abbreviated to MC, we will denote this criterion as EF moving forward - referring to the Expected value of f.

3.3.2 Upper Confidence Bound

A slightly more considered approach is the Upper Confidence Bound or UCB acquisition policy which was proposed by Srinivas et al. (2012) and falls under the category of optimistic acquisition policies. In contrast to EF which focuses solely on exploitation of search space, UCB seeks to strike a balance between exploration and exploitation through a weighted sum of the posterior mean and variance. Thus our acquisition function becomes,

$$\mathbf{x}^* = \underset{\mathbf{x}\in\mathcal{X}^d}{\operatorname{argmax}}\{\mu(\mathbf{x}) + \beta \cdot \sigma^2(\mathbf{x})\} \tag{3.5}$$

where $\mu(\mathbf{x})$ and $\sigma^2(\mathbf{x})$ refer to the posterior mean and variance of f as defined in Equation 2.10. In Equation 3.5 we can see the level of exploration is determined by "*tuning*" parameter β. Higher values for β lead to more exploration of the search space, while lower values lead to more exploitation, with $\beta \to 0$ tending towards EF. Introducing a parameter which requires tuning may seem wasteful in situations where we are already trying to minimize the amount of times we evaluate our expensive function f, but Srinivas et al. (2012) provide theoretically motivated guidelines for choosing its value. In practice this acquisition policy is used in situations when we are interested in exploring the search space in hope of finding improvements on solutions already found. An example of this being the hyperparameter tuning of small scale machine learning algorithms where UCB is the acquisition policy implemented in most software packages, a popular choice being the Keras Tuner (O'Malley et al., 2019).

3.3.3 Expected Improvement

Turning our attention to explicitly seeking a global extrema, we now consider the *Expected Improvement* or EI criterion. EI serves as the basis of the *Efficient Global Optimization* algorithm first proposed by Jones et al. (1998) in their paper which revisited Močkus' ideas from a Gaussian Process perspective. The idea is to quantify the potential for improvement through the random variable $I(\mathbf{x}) = \max\{0, f^{\text{best}} - f(\mathbf{x})\}$ which measures the amount by which the proposed response $f(\mathbf{x})$ could be better than the best response observed so far, f^{best} (how much more for maximization and conversely less for minimization). This allows us to calculate and subsequently maximise $\mathbb{E}[I(\mathbf{x})|\mathcal{D}]$ known as the *Expected Improvement* of evaluating $f(\mathbf{x})$ given the evaluations already made. Gramacy (2020) shows how this can be done without any functional form for $f(\mathbf{x})$ using a Monte Carlo approximation,

$$\mathbb{E}[I(\mathbf{x})|\mathcal{D}] \approx \frac{1}{T}\sum_{t=1}^{T} \max\{0, f^{\text{best}} - f_t(\mathbf{x})\} \tag{3.6}$$

which is not particularly attractive given evaluations of f are assumed expensive. However if $f(\mathbf{x})$ is assumed to be Gaussian, as is the case with a Gaussian Process surrogate,

$\mathbb{E}[I(\mathbf{x})|\mathcal{D}]$ has a closed analytical form. This was first shown by Schonlau (1997) in his dissertation entitled *Computer experiments and global optimization* as being,

$$\mathbb{E}[I(\mathbf{x})|\mathcal{D}] = (f^{\text{best}} - \mu(\mathbf{x})) \cdot \Phi\left(\frac{f^{\text{best}} - \mu(\mathbf{x})}{\sigma(\mathbf{x})}\right) + \sigma^2(\mathbf{x}) \cdot \phi\left(\frac{f^{\text{best}} - \mu(\mathbf{x})}{\sigma(\mathbf{x})}\right) \tag{3.7}$$

where Φ and ϕ are the standard normal CDF and PDF respectively. $\mu(\mathbf{x})$ and $\sigma^2(\mathbf{x})$ are again the posterior mean and variance of f defined in Equation 2.10. The proof uses both substitution and integration by parts and is best left to Schonlau. The first term in Equation 3.7 is the amount by which the predictive mean differs from the current best observation f^{best}, multiplied by what is known as the *probability of improvement*

$$\begin{aligned}\mathbb{P}[I(\mathbf{x}) > 0|\mathcal{D}] &= \mathbb{P}[f^{\text{best}} - f(\mathbf{x}) > 0|\mathcal{D}] \\ &= \mathbb{P}[f(\mathbf{x}) < f^{\text{best}}|\mathcal{D}].\end{aligned} \tag{3.8}$$

The second term is the posterior variance weighted by a Gaussian density evaluation. Equation 3.7 provides a organic balance between *exploitation* (how much better $f(\mathbf{x})$ is than f^{best}) and *exploration* (seeking areas with large posterior variance or uncertainty). With this all in mind, we finally define the EI acquisition policy as,

$$\mathbf{x}^* = \underset{\mathbf{x}\in\mathcal{X}^d}{\text{argmax}}\{\mathbb{E}[I(\mathbf{x})|\mathcal{D}]\} \tag{3.9}$$

EI is guaranteed to converge towards the global maximum (or minimum) but only because in the limit it will have *explored* the entire search space - although unsatisfying this is a property it shares with a simple random search. The impressive part is that it can be shown to provide the best *next* sample if the *next* sample is the last one to be taken. That is when using EI ,$\mathbf{x}_{n+1}$ will be optimal if $n + 1 = N$ where N denotes our total available budget (Gramacy, 2020). For obvious reasons EI is one of the most popular choices of acquisition strategy in limited budget settings.

3.4 Bayesian Optimization Algorithm

In summary Bayesian optimization requires two fundamental pieces. Firstly a probabilistic surrogate which captures our belief in the unknown function f and can be updated in light of new training data. Secondly, an acquisition function or search policy which determines how we sequentially search for a solution to Equation 3.1 in a way which balances the competing objectives of exploitation and exploration. With all of this in mind, we can formally describe the Bayesian optimization algorithm using the pseudo-code seen in below in Algorithm 1. This specific implementation is an adaption on similar algorithms proposed by both Gramacy (2020) and Frazier (2018). In addition to this we provide a slightly more friendly flowchart which gives a high-level overview of the Bayesian optimization procedure below in Figure 7.

Algorithm 1: Bayesian Optimization

Choose surrogate $\hat{f}$ and acquisition function $\mathcal{J}$
Evaluate f at n_0 initial locations to create $\mathcal{D}_{n_0} = (\mathbf{X}_{n_0}, f(\mathbf{X}_{n_0}))$
Set maximum number of allowable iterations N and $n = n_0$
while $n < N$:
    Fit $\hat{f}$ using $\mathcal{D}_n$
    Obtain $\mathbf{x}_{n+1} = \underset{\mathbf{x} \in \mathcal{X}^d}{\operatorname{argmax}} \mathcal{J}(\mathbf{x})$ using $\hat{f}$
    Evaluate f at $\mathbf{x}_{n+1}$
    Update $\mathcal{D}_{n+1} = \mathcal{D}_n \cap (\mathbf{x}_{n+1}, f(\mathbf{x}_{n+1}))$
    Increment n
return $\mathcal{D}_n$ and best $(\mathbf{x}, f(\mathbf{x}))$

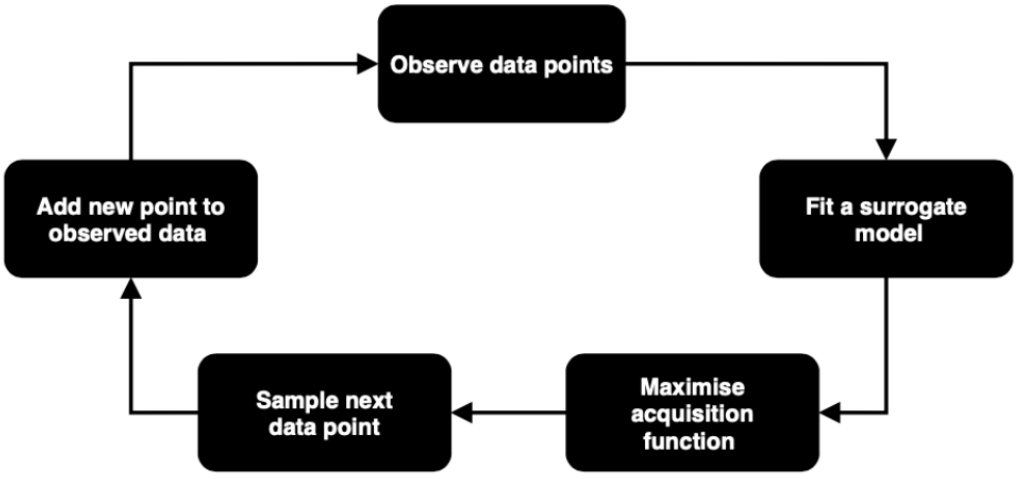


Figure 7: Flowchart describing the Bayesian optimization algorithm.

3.4.1 Implementation Details

Both Algorithm 1 and Figure 7 give us an idea of the sequential nature of Bayesian optimization. To reiterate, this is ideal in situations where we have a limited budget of function evaluations and therefore want wish to maximise the utility of the next evaluation. This is achieved within the inner-loop of Algorithm 1 by the maximisation of our acquisition function $\mathcal{J}$, typically through the use of a numerical optimization routine. We also note that the algorithm requires random initialization through the evaluation of f at n_0 initial locations. This number varies given specific contexts but Gramacy (2020) suggests $n_0 = 12$ as a good starting point.

For this project we have implemented our own version of Algorithm 1 from first-principles using the R programming language (R Core Team, 2023). This code can be found at `github.com/bchiddy` where you will find several functions which implement both Algorithm 1 as well as the acquisition functions described in Sections 3.3.1, 3.3.2 and 3.3.3. Our implementation is heavily influenced by Gramacy (2020) and makes use of his `laGP` package for fitting of the Gaussian Process surrogate in line 5 of Algorithm 1, both for its speed and ease of use. In addition to this, we make use of the L-BFGS-B optimization routine (Byrd et al., 1995) to perform the maximisation of $\mathcal{J}$ in line 6 of Algorithm 1 - another choice which is inspired by Gramacy (2020). Note that this is done by means of the `optim` function which is freely available inside the `stats` package which ships with base R. A final note on implementation is in regards to the setting of the tuning parameter β when using the UCB acquisition strategy. Here the decision is made not to explicitly tune its value but to rather set it arbitrarily to $10 \times \{\max(f(\mathbf{x}) - \min(f(\mathbf{x})\}$ where f is the function being evaluated. This will ensure it favours exploration over exploitation which is the reason for its inclusion in this study.

Lastly, we note that Bayesian optimization is extremely popular in the context of the hyperparamater tuning of machine learning models and is therefore also implemented in many open-source software packages. Notably, the already mentioned `kerastuner` (O'Malley et al., 2019) and `bayesian_optimization` (Nogueira, 2014) libraries are popular choices for python users.

In the following section we make use of our custom implementation of the Bayesian optimization algorithm to benchmark its performance against several popular optimization routines and across three popular analytical test functions in the field of numerical optimization. A concise version of this analysis can also be found at the above `url` where general purpose code such as that for plotting and data processing has been removed.

4 Benchmarking Algorithm Performance

4.1 Experimental Overview

Having reviewed the fundamental elements of the Bayesian optimization algorithm, we now investigate its performance across several popular test functions in the field of numerical optimization. These functions, taken from the *Virtual Library of Simulation Experiments* (Surjanovic and Bingham, 2013), have been selected as they present a diverse set of challenges which will allow us to understand both the strengths and limitations of Bayesian optimization.

The goal of this section will be to gauge the performance of various implementations of the Bayesian optimization algorithm, specifically those using the acquisition functions discussed in the previous sections. Naturally, we include several popular optimization routines as benchmarks to compare algorithm performance against. Our approach will be to first gain an understanding of each function surface and what makes it difficult to optimize. These function surfaces are typically presented as global minimization problems and thus our objective in each case will be to find $\mathbf{x}^*$ such that

$$\mathbf{x}^* = \underset{\mathbf{x} \in \mathcal{X}^d}{\operatorname{argmin}} f(\mathbf{x}) \tag{4.1}$$

where f is assumed to be in some sense expensive to evaluate. This means we seek solutions to Equation 4.1 in as little evaluations as possible while remaining within some specified budget. To gain an understanding for each algorithm (benchmarks included) we visualise how it sequentially searches for solutions across the function surface using series of plots which provide a snapshot of the optimization process. This will hopefully allow us to better understand why an algorithm might excel or conversely perform poorly on a particular function surface. Noting that the test functions selected are highly multimodal, the selection of starting values for each algorithm is of particular importance. For this reason we then present a simulation study which analyses algorithm performance across multiple random initializations. This will show us which algorithm provides the most robust performance and thus can be used with confidence.

4.2 Benchmark Algorithms

These optimization routines have been selected as they employ a variety of different approaches when solving Equation (4.1). In addition to this most form part of the popular optimization function `optim` (R Core Team, 2023) and are therefore used extensively when solving general purpose optimization problems as they come with little overhead. Note that we are dealing with surfaces which are multi-model and non-convex thus these routines are not guaranteed to find the global minimum of each function surface, in this regard they are closer to being heuristics than optimization techniques. In addition to this, because our focus is in regards to the optimization of expensive functions we will limit the amount of function evaluations each of these routines are allowed in solving Equation 4.1. Although this is not at all what most of these algorithms were designed for, it will allow us to assess just how much better the Bayesian alternative might be when

tasked with optimization in a low budget setting. In what follows we briefly describe the approach of each algorithm along with its specific reason for inclusion in this study.

Random Search

The Random Search (RS) algorithm is a classical benchmark in many optimization studies which simply evaluates random locations within the parameter space. Because of this it is known as a direct search technique. Apart from it being by far the easiest method to implement it sometimes can obtain impressive results in difficult, non-convex optimization problems. Because the algorithm has no sense as to the form of f it freely searches the parameter space without becoming stuck in local minima. This also means that it provides no sense of convergence and will continue until stopped by the user. It has become a popular choice in the field of machine learning where it is often used to conduct hyperparameter tuning (Bergstra and Bengio, 2012). Although being an extremely light-weight approach, it is a naive choice in situations where we are truly interested in finding the global minimum and is extremely wasteful in situations where evaluations of f are expensive.

Limited Memory Broyden–Fletcher–Goldfarb–Shannon

The Limited Memory Broyden–Fletcher–Goldfarb–Shannon (L-FBGS) algorithm (Byrd et al., 1995) is a gradient based optimization routine designed for smooth and differentiable functions. It is known as a quasi-Newton newton method as at each step it estimates the Hessian matrix of $f(\mathbf{x})$ to guide its movement throughout the parameter space. Notably, the L-BGFS algorithm stores only a few vectors instead of the entire $p \times p$ Hessian matrix at each iteration making it an ideal choice in high dimensional optimization problems. Gradient information allows it to find the direction of steepest descent which can lead to quick convergence when optimizing *well-behaved* functions. However, in situations where f is non-convex and multi-modal the it can become stuck in a local minima leading to a false sense of convergence to the global minimum. This routine is also a poor choice in situations where function evaluations are limited as it expends many calculating gradients. Note that we will make use of the L-BFGS-B variant of this algorithm which introduces box constraints on the L-BGFS variant.

Nelder-Mead

The Nelder-Mead (NLM) algorithm (Nelder and Mead, 1965) is a gradient free routine designed for robust optimization of non-differentiable, non-linear and non-convex functions. It is also known as the "downhill-simplex" method as the algorithm creates a geometric simplex within the parameter space which it then iteratively reflects, contracts and adapts the vertices of to *climb* downhill towards the function minimum. This algorithm serves as the default setting of the `optim` function due to its versatility and derivative free approach. Because the method only makes use of function evaluations it can lead to slower optimization than other methods. Although both Nelder-Mead and Bayesian optimization offer derivative free approaches, the former requires far more function evaluations and is therefore a poor choice when function evaluations are expensive.

Simulated Annealing

In contrast to both NLM and L-BFGS-B, the Simulated Annealing (SANN) algorithm (Bélisle, 1992) is a probabilistic optimization technique. At each iteration it evaluates the function surface at *neighbouring* locations and permits movement to possibly worse solutions based on a probability rule in a similar fashion to the Metropolis Hastings algorithm (Metropolis et al., 1953). This allows SANN to escape, or *jump-out* local minima and is therefore considered to be an approximate global optimization routine. This feature makes it an ideal choice in non-convex and combinatorial optimization problems. The process is said to closely resemble the cooling process in metallurgy which is where is derives its name from. While both Bayesian optimization and SANN are probabilistic techniques for approximating the global optimum, SANN requires many function evaluations leading to poor performance on small optimization budgets. In addition to this SANN requires careful tuning of a set of control parameters which presents added overhead for the user. Our implementation will make use the default settings for SANN as implemented in `optim` in an attempt to mimic its use by a non-expert and to match the process they would follow when implementing each of the other benchmark algorithms.

4.3 Methodology

As previously described, we are interested finding an optimal set of parameters $\mathbf{x}^*$ in regards to Equation (4.1) with as little evaluations of the function f as possible. This is because f is some sense (either commercially or computationally) expensive to evaluate. In addition to this, when solving problems such as those described in Section 3.1 we would not normally known the analytical form of the function we are trying to optimize. However, for the purposes of this study we have intentionally chosen a set of test function for which we have a known analytical form. This will aid both visualisation and in understanding the strengths and weaknesses of each algorithm. For all intents and purposes we will assume that our test functions are opaque and represent something which is expensive to evaluate. We provide in-depth descriptions of each function's characteristics in the following sections but provide a brief summary below in Table 1.

Table 1: Summary of analytical test functions taken from the *Virtual Library of Simulation Experiments* (Surjanovic and Bingham, 2013).

Function	**Modality**	**Dimensionality**	**Reference**
Scaled Goldstein-Price	Multi	2	Picheny et al. (2013)
Ackley	Multi	2	Ackley (1987)
Bukin Number 6	Multi	2	Silagadze (2007)

These functions provide a diverse set of challenges which make convergence to the global minimum difficult. Our goal will be to asses the performance of the three variants of the Bayesian optimization algorithm described in Sections 3.3.1, 3.3.2 and 3.3.3 across the test functions listed in Table 1. Their performance will be judged against that of the benchmark optimization routines described in Section 4.2. We provide a brief summary of each algorithm to be tested along with categorization of its approach to optimization below in Table 2 .

Table 2: Summary of algorithms to be compared.

Algorithm	**Approach**	**Abbreviation**
Random Search	Direct	RS
Limited Memory Broyden–Fletcher–Goldfarb–Shannon	Gradient	L-BFGS-B
Nelder-Mead	Simplex	NLM
Simulated Annealing	Stochastic	SANN
Mean Critereon	Bayesian	EF
Upper Confidence Bound	Bayesian	UCB
Expected Improvement	Bayesian	EI

4.3.1 Experimental Approach

In the pseudo-code for the Bayesian optimization algorithm shown in Algorithm 1 we saw that it required random initialization. In addition to this, it is customary to use each of the `optim` algorithms with a variety of different start points to try and avoid convergence to a local minima. For these reasons we have decided to conduct a Monte-Carlo simulation to obtain a robust view of algorithm performance across the chosen test functions. In each case we will run the algorithm $N = 1000$ times, each run using a different random initialization, and report the average results. Note that to ensure a fair comparison the, same starting values will be used for each algorithm. In regards to the `optim` routines this will be a single starting location. However, in the case of the Bayesian optimization algorithms this will be a random initialization of $n_0 = 12$ starting locations based on the recommendation of Gramacy (2020). Because the test functions are assumed to be expensive to evaluate we also impose a strict budget of $n = 50$ evaluations (including the initialization) on each function. Although our main focus will be the performance at the end of these n evaluations, we will also be interested in how the algorithm progresses at each of the $n = 1, 2, \ldots, 50$ steps. The reason for this is to assess which algorithm converges to an optimal solution the quickest. With this in mind our metrics of success across the $N = 1000$ algorithm runs will be the following:

1. **Best Objective Function Value (BOV):** The best solution to Equation 4.1 after the n^{th} evaluation of f. This will help us understand which algorithm performs optimization the quickest which is a favourable quality when evaluations are expensive.
2. **Variability in Final Results:** How different the solutions to Equation 4.1 are after the $n = 50^{\text{th}}$ evaluation of f. This will help us understand how sensitive each algorithm is to the starting locations used to initialize it. Low variability will mean an algorithm is robust and can be used with confidence.

In what follows we briefly describe each test function listed in Table 1 and what makes them difficult. Next we visualise each algorithms progression across the function surface given a *single* random initialization. We then present the results of our Monte Carlo simulation in the form of line and box-plots which give an indication of algorithm performance against our metrics of success. Finally, we summarise the results across each test function in the form of a table along with a brief discussion on what was learnt.

4.4 Scaled Goldstein-Price Function

This function is evaluated on the square $x_i \in [0, 1]$ and has several local minima which lie in ridges above the single global minimum of -3.129 located at $\mathbf{x} = (0.50, 0.25)$. Although the function slopes are smooth and easy to travel down, many algorithms get trapped in local minima and therefore do not converge to the true global minimum. We present the analytical form below along with surface and contour plots in Figure 8. After this, we visualize how each algorithm moves across the function surface starting with the benchmarks in Figure 9 after which the Bayesian optimization algorithms with EF, UCB and EI shown in Figures 10, 11 and 12 respectively.

$$f(\mathbf{x}) = \frac{1}{2.427}\Big[\log\Big([1 + (\bar{x}_1 + \bar{x}_2 + 1)^2(19 - 14\bar{x}_1 + 3\bar{x}_1^2 - 14\bar{x}_2 + 6\bar{x}_1\bar{x}_2 + 3\bar{x}_2^2)] \\ [30 + (2\bar{x}_1 - 3\bar{x}_2)^2(18 + 32\bar{x}_1 + 12\bar{x}_1^2 + 48\bar{x}_2 - 36\bar{x}_1\bar{x}_2 + 27\bar{x}_2^2)]\Big) - 8.693\Big]$$

where $\bar{x}_i = 4x_i - 2, \quad i = 1, 2$

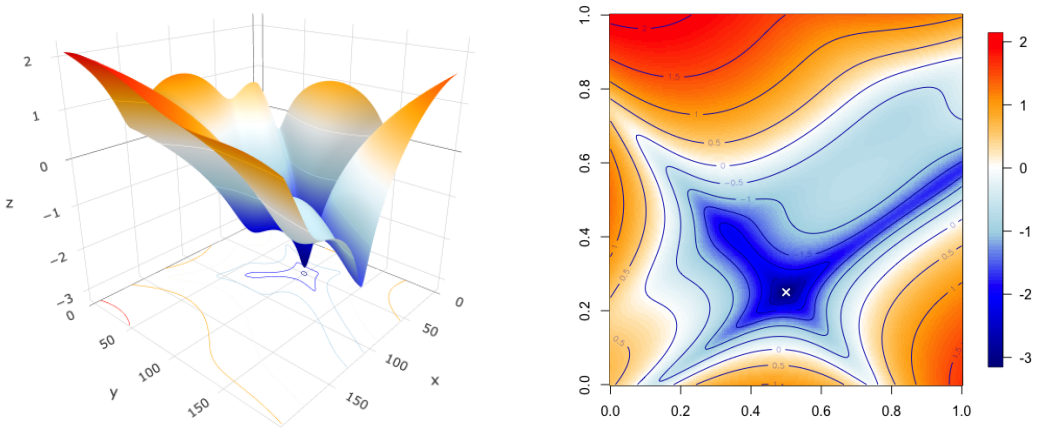
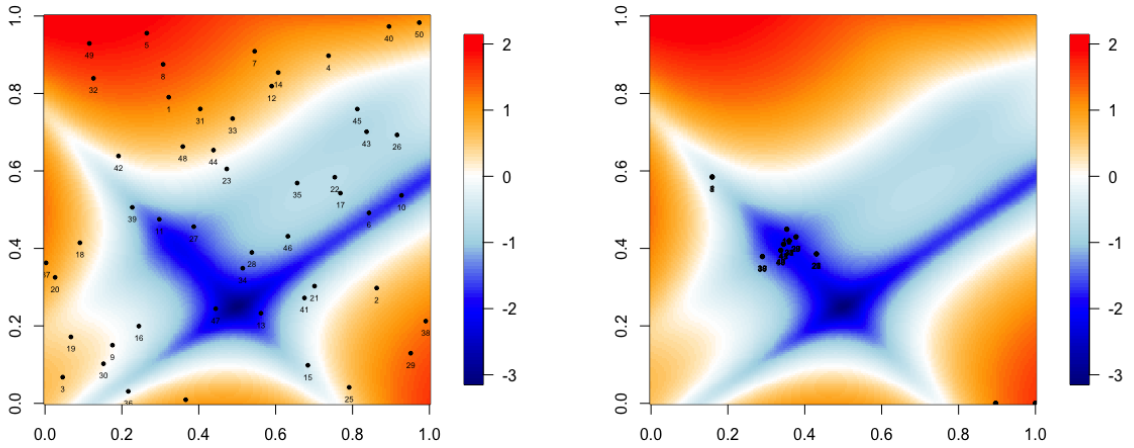


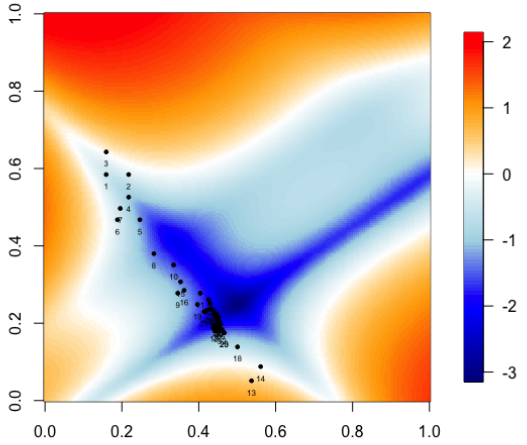
Figure 8: Surface (left) and contour (right) plots of the Scaled Goldstein-Price function with the location of the global minimum marked by a white x.

Firstly, we notice that none of the four benchmark algorithms have found the global minimum. Both L-BFGS-B and NLM have become trapped in the local minima which lie in the ridge above the global minimum. L-BFGS-B wastes valuable function evaluations calculating gradients which is why we see it clustered in a few locations while NLM moves up and down the slopes above the local minima giving it a false sense of convergence. Although initialized in the same location, SANN has started evaluating *neighbouring* locations outside the bounds of the search space meaning it has completely missed even the local minima. The RS algorithm seems to have found the best $\mathbf{x}$ location but this is purely by chance and has wasted over half of its evaluations in areas of the function surface which are far from the minimum. Note that all of these algorithms would benefit from more evaluations but have been constrained by the budget $n = 50$.

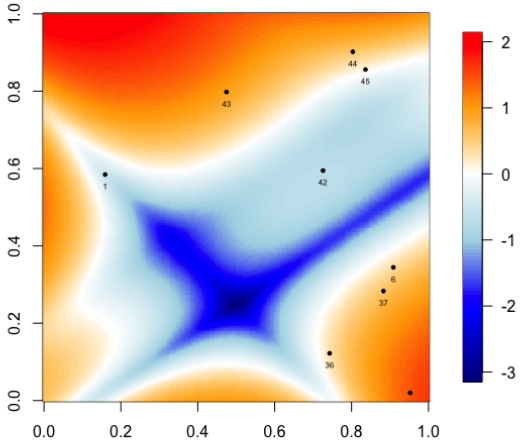


(a) RS

(b) L-BFGS-B



(c) NLM



(d) SANN

Figure 9: Progression of each of the 4 benchmark algorithms across the Scaled Goldstein-Price function surface with a limited budget of $n = 50$ using the same random initialization. Note that the contour lines have been removed to improve readability. Different starting values can lead to better results in all cases but we present these as examples of the algorithms failing when used naively.

We now turn our attention to the progression of the Bayesian optimization algorithms across the Scaled Goldstein-Price function surface. Recall that at each step, these algorithms create a Gaussian Process surrogate of the function surface (see Algorithm 1). This means that after having evaluated the function at several initial locations we can compute the posterior mean and variance of the surrogate model to gain an understanding of its form. Ideally this surrogate will match the true function surface as closely as possible thereby allowing optimization of the surrogate to provide good solutions for Equation 4.1. In Figures 10, 11 and 12 we present the true function surface (top row) posterior mean (middle row) and posterior variance (bottom row) of the EF, UCB and EI acquisition functions respectively. To show how these surfaces change as the algorithm progresses we also present 3 snapshots (values of n) within each of the corresponding rows. Note that each algorithm was initialized using 12 function evaluations (shown as triangles) at locations sampled through a latin-hypercube scheme.

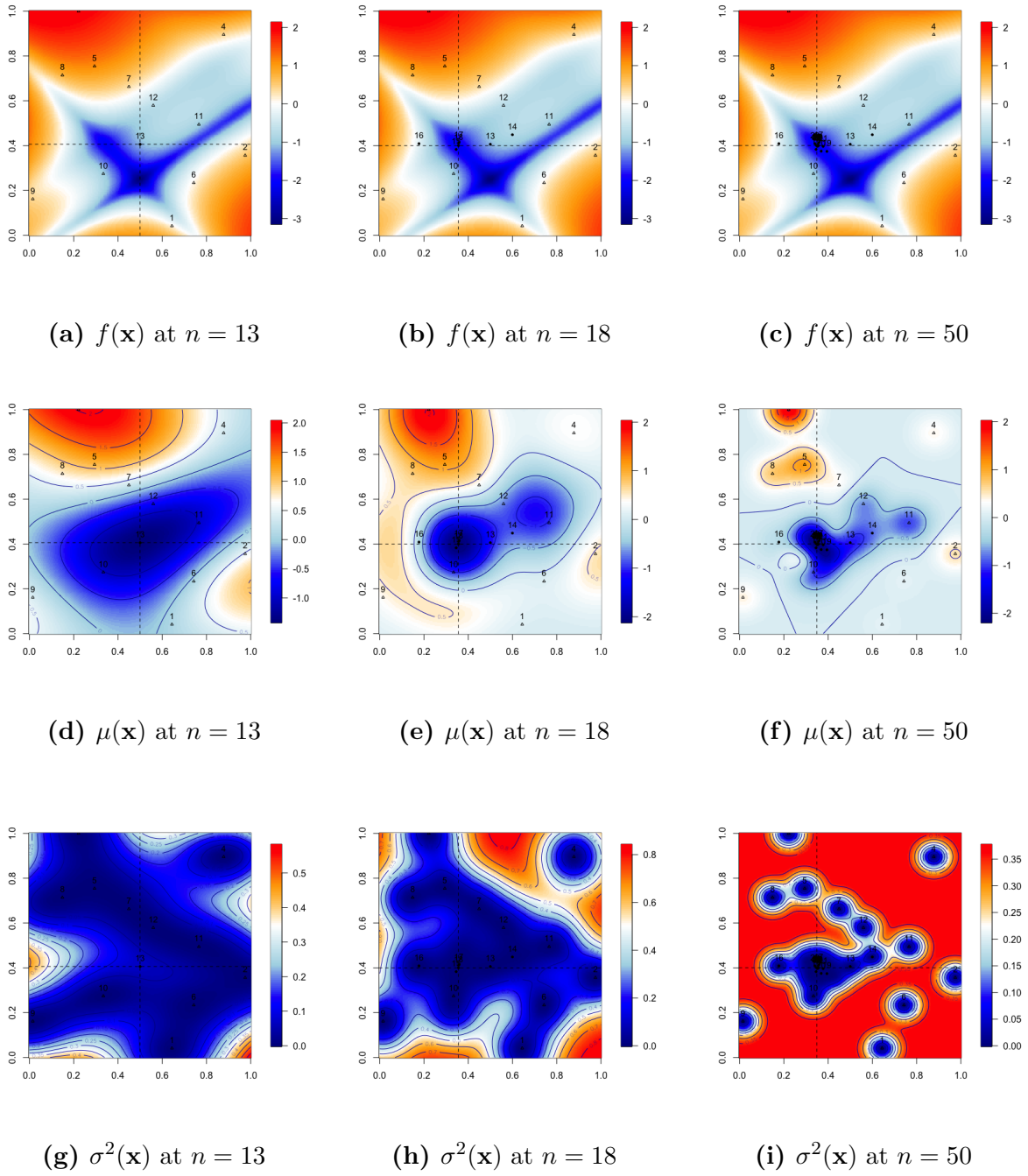
Progression of EF in Figure 10 At $n = 13$ the surrogate already has a reasonable picture of the function surface. The posterior mean predicts that the smaller values are located towards the middle of the surface while larger values are located around the edges. Note that because most of the initial function evaluations seem to have fallen in areas with similar values (not too high or too low) the posterior variance is roughly equal across the surface meaning that it has low uncertainty in the locations it has not evaluated, believing they are similar to the locations it has. Note that the selected $\mathbf{x}_{n+1}$ (marked by dotted black lines) is located at the point where the posterior mean is at a minimum. At $n = 18$ we see the algorithm has found a local minima which lies in the ridge above the global minimum. The posterior mean seems to have become more accurate when compared to the true function surface but believes the global minimum lies within the ridge and therefore seems to be over-exploiting this area of the function surface. The posterior variance remains similar to that at $n = 13$ but has started to to have higher uncertainty in regards to the areas it has not explored. At $n = 50$ we can see that just as the benchmark algorithms, the EF algorithm has become stuck in a local minima. This is because it inherently favours exploitation of the surrogate surface meaning if initialized with values which lie near a local minimum it will not see the need to explore elsewhere. We can see that the surrogate surface ends with an fairly incorrect picture of the true surface as it believes anything outside of the local minima it sits inside has somewhat equal value. The posterior variance shows that apart from the locations it has evaluated it now has high uncertainty in all other locations. This is because it has remained in the same location and not explored elsewhere.

Progression of UCB in Figure 11 At $n = 13$ the surrogate has the same picture as was seen before with EF. However, notice that the selected $\mathbf{x}_{n+1}$ is no longer at the minimum predicted by the posterior mean. Instead, the algorithm favours exploring areas of high uncertainty, as shown by the posterior variance which itself is similar to what was seen before. At $n = 32$ we can see that through exploration the algorithm has found where the global minimum is located! This shows how UCB solves the problem of over-exploitation, allowing it to search for better solutions to Equation 4.1 after having found some local minimum. At this stage, the posterior mean resembles the true function

surface far more closely, although still not perfectly. The posterior variance seems similar to what was seen before but now has high levels of uncertainty towards the middle as opposed to the edges. Although the algorithm has already found the global minimum, we see that at $n = 50$ it has continued to explore the surface in the hope of finding an even better solution. We can see that it has evaluated the function in a far more diverse set of locations than EF and therefore ends with a much more accurate picture of the true surface. The posterior variance at this stage again shows that as the algorithm has progressed it has become more confident in the locations it has evaluated and less confident in the locations it has not. Note that UCB does waste function evaluations in areas where there is little return. As shown in 3.5 the level of exploration can be tuned but comes with extra overhead.

Progression of EI in Figure 12 Again we see that at $n = 13$ the surrogate has a the same posterior mean and variance as with EF and UCB. However now we see that instead of exploring an area of high uncertainty, EI selects $\mathbf{x}_{n+1}$ in the location the posterior mean is at a minimum much like what we saw with EF as it is the area which maximizes expected improvement. By $n = 24$ the algorithm has already found the global minimum, note that this is after 8 fewer iterations than UCB. We can see that the posterior mean has a fairly good picture of the true surface with low value areas predicted towards the center and high value areas around the edges. However, it has not learnt the shape of the ridges above the global minimum. Note that the posterior variance is similar to what was seen only after 50 iterations in the previous two algorithms meaning that EI quantifies uncertainty far quicker. The real power of the EI strategy is seen at $n = 50$. Just as with UCB we see that it has evaluated the function at a far more diverse set of locations than EF meaning it has certainly explored the surface. However, notice than in contrast to UCB it has only explored areas within the ridge where all the low value areas lie. As was seen in Equation 3.7, EI provides an organic balance between exploitation and exploration meaning that the algorithm is smart enough to explore areas that have a high probability of improvement. Because of this it ends with the most accurate picture of the true surface out of the the three Bayesian optimization algorithms. Although it has not perfectly accurate it knows that the lowest values lie in the ridge in the center of the surface with higher values all around it.

Having seen examples of how each algorithm sequentially moves across the function surface given a single random initialization we now investigate their average performance given $N = 1000$ such random initializations. To reiterate what was said in Section 4.3.1 this will give us a better sense as to the reliability (or robustness) of each algorithm, many of which are extremely sensitive to starting locations used due to the the non-convex and multi-modal nature of the Scaled Goldstein-Price function surface.

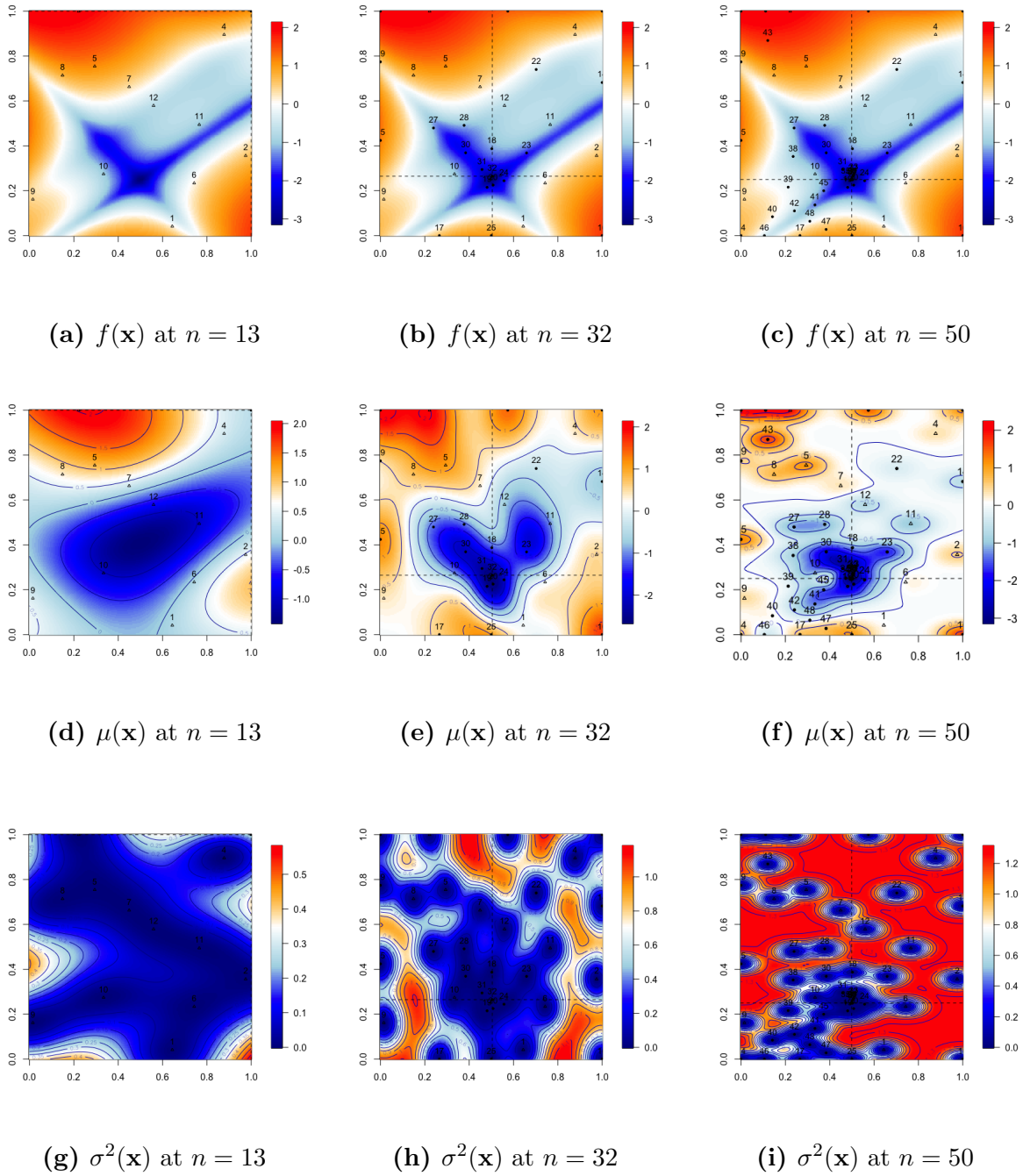


(a) $f(\mathbf{x})$ at $n = 13$ **(b)** $f(\mathbf{x})$ at $n = 18$ **(c)** $f(\mathbf{x})$ at $n = 50$

(d) $\mu(\mathbf{x})$ at $n = 13$ **(e)** $\mu(\mathbf{x})$ at $n = 18$ **(f)** $\mu(\mathbf{x})$ at $n = 50$

(g) $\sigma^2(\mathbf{x})$ at $n = 13$ **(h)** $\sigma^2(\mathbf{x})$ at $n = 18$ **(i)** $\sigma^2(\mathbf{x})$ at $n = 50$

Figure 10: Bayesian optimization algorithm at steps 13, 18 and 50 using the EF acquisition strategy across the Scaled Goldstein-Price function surface. The top row shows the actual function surface, the middle row shows the posterior mean of our surrogate $\hat{f}$ while the bottom row shows its posterior variance. In each case the $\mathbf{x}_{n+1}$ selected by the algorithm is shown by dotted black lines.

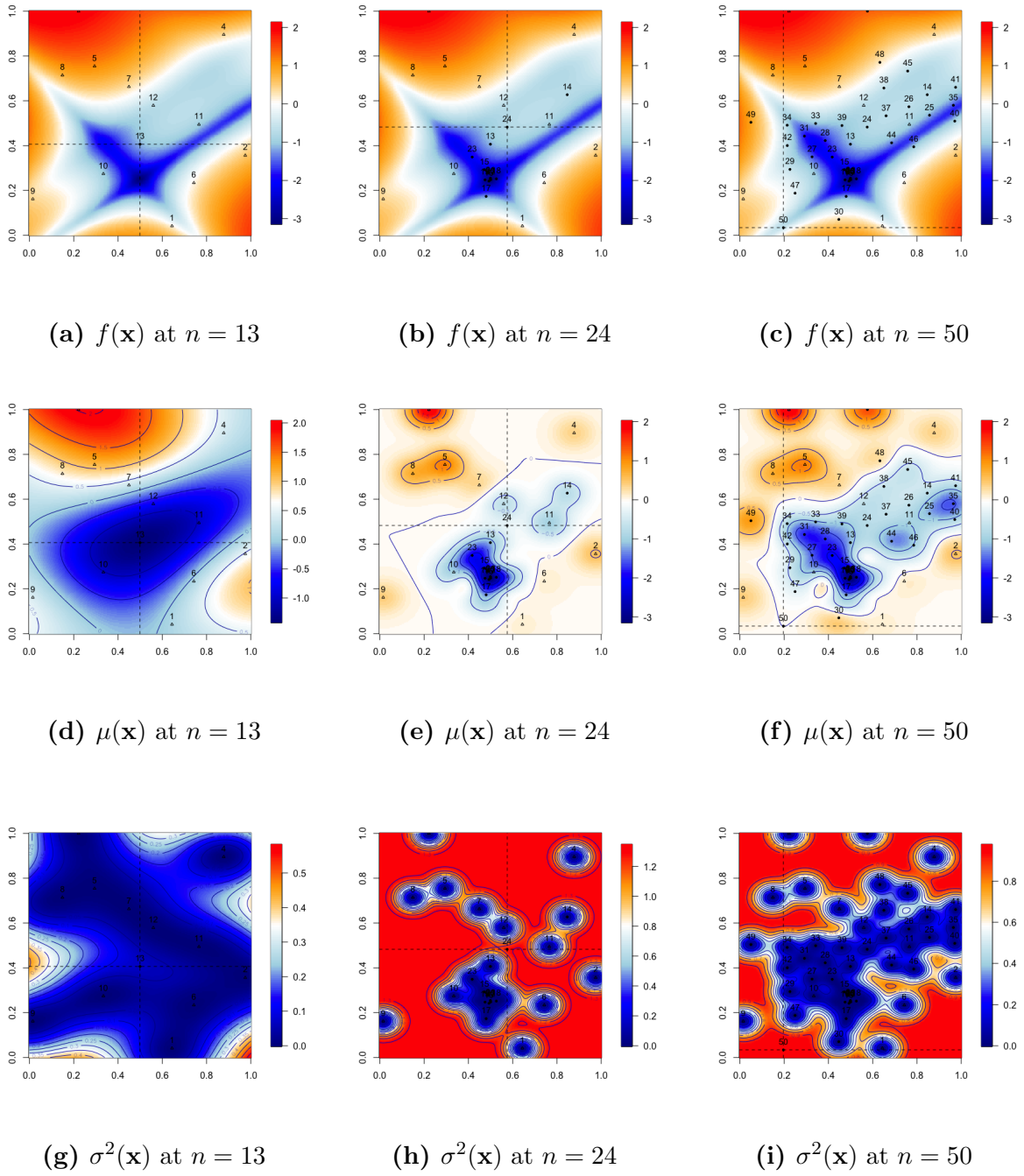


(a) $f(\mathbf{x})$ at $n = 13$ **(b)** $f(\mathbf{x})$ at $n = 32$ **(c)** $f(\mathbf{x})$ at $n = 50$

(d) $\mu(\mathbf{x})$ at $n = 13$ **(e)** $\mu(\mathbf{x})$ at $n = 32$ **(f)** $\mu(\mathbf{x})$ at $n = 50$

(g) $\sigma^2(\mathbf{x})$ at $n = 13$ **(h)** $\sigma^2(\mathbf{x})$ at $n = 32$ **(i)** $\sigma^2(\mathbf{x})$ at $n = 50$

Figure 11: Bayesian optimization algorithm at steps 13, 32 and 50 using the UCB acquisition strategy across the Scaled Goldstein-Price function surface. The top row shows the actual function surface, the middle row shows the posterior mean of our surrogate $\hat{f}$ while the bottom row shows its posterior variance. In each case the $\mathbf{x}_{n+1}$ selected by the algorithm is shown by dotted black lines.



(a) $f(\mathbf{x})$ at $n = 13$ **(b)** $f(\mathbf{x})$ at $n = 24$ **(c)** $f(\mathbf{x})$ at $n = 50$

(d) $\mu(\mathbf{x})$ at $n = 13$ **(e)** $\mu(\mathbf{x})$ at $n = 24$ **(f)** $\mu(\mathbf{x})$ at $n = 50$

(g) $\sigma^2(\mathbf{x})$ at $n = 13$ **(h)** $\sigma^2(\mathbf{x})$ at $n = 24$ **(i)** $\sigma^2(\mathbf{x})$ at $n = 50$

Figure 12: Bayesian optimization algorithm at steps 13, 24 and 50 using the EI acquisition strategy across the Scaled Goldstein-Price function surface. The top row shows the actual function surface, the middle row shows the posterior mean of our surrogate $\hat{f}$ while the bottom row shows its posterior variance. In each case the $\mathbf{x}_{n+1}$ selected by the algorithm is shown by dotted black lines.

Presented below in Figure 13 are the results of our Monte Carlo simulation for each algorithm on the Scaled Goldstein-Price function. The line plot to the left shows the average best objective function value (BOV) after n evaluations of f across $N = 1000$ randomly initialized algorithm runs. The box plot to the right shows the variance in the final BOV returned by each algorithm after $n = 50$ evaluations of f. The goal is to have the the BOV curve dip towards the true minimum as quickly as possible, thus lines which occupy the most space towards the bottom left of the plot are favoured as this means they have quickly converged to the minimum value. Because each algorithm is randomly initialized we also want to see as little variance in the final results as possible, thus making us more confident in results returned by a single run.

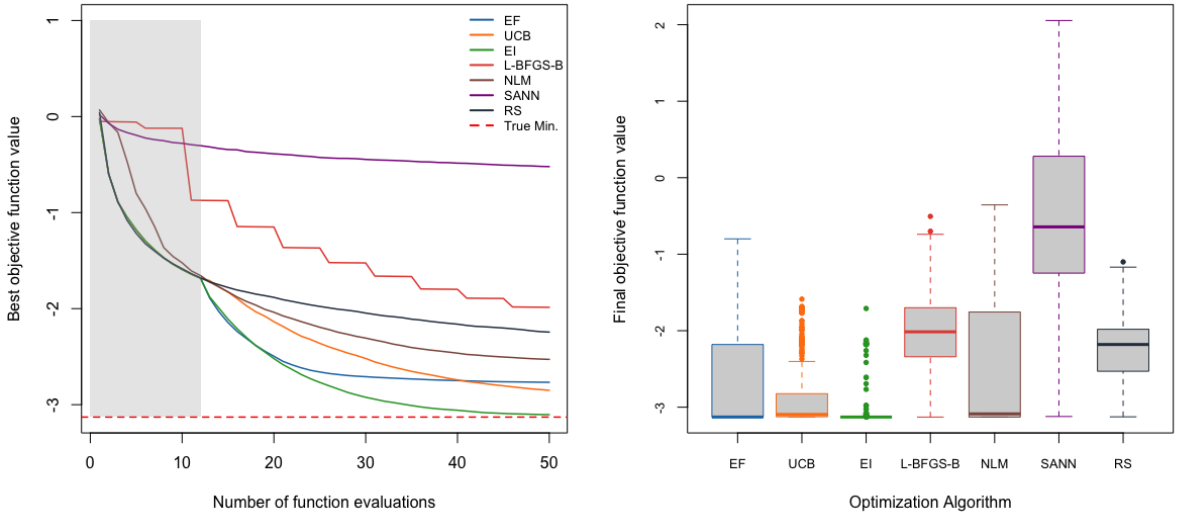


Figure 13: Plots showing the average BOV (across $N = 1000$ runs) achieved by each algorithm after n evaluations of f (left) as well as the variation in the final BOV returned after $n = 50$ (right). The initialization period for the Bayesian optimization algorithms is shown by the shaded region in the line plot.

Performance of Benchmark Algorithms We see that SANN performs consistently poorly and hardly ever achieves a good BOV. This mostly due to the constrained budget as the algorithm requires many function evaluations to obtain a reasonable picture of the function surface. This is also reflected in the fact that it has the highest variability in final results across all algorithms. Next we see that L-BFGS-B has a distinct step like pattern, this is due to the algorithm expending function evaluations on gradient calculations at various points on the function surface and thus clusters in certain locations. Because of this it also does fairly poorly in the given budget although does provide less variable final results than SANN, possibly due to its ability to use the calculated gradient information. RS does better than both of these algorithms although still struggles to consistently find the global minimum. It shows a nice smooth curve which decreases rapidly after only a few function evaluations but plateaus with increased values of n meaning that it struggles

to consistently find low function values by chance. Interestingly it has less variable final results than both L-BFGS-B and SANN. NLM shows the best results out of all the benchmark algorithms with a nice curve which continues to drop as function evaluations increase. This is in line with what we saw in Figure 9 as the function can take advantage of good starting positions to quickly climb into the function minima. That being said it still plateaus well above the true minimum as it often gets caught in local minima.

Performance of Bayesian Optimization Algorithms Firstly we see that until the end of the initialization period (shown as a shaded grey region) they all follow the progression of RS. This is because they are initialized with a set of 12 random values and are essentially just a random search themselves up until this point. Interestingly, after the initialization period we see a noticeable dip in the BOV curves. This shows that after switching to optimization of the surrogate surface it can immediately begin to find improvements given the evaluations it has already made. When looking at EF we see that it immediately dips far below RS before seemingly converging at a point well above the true minimum. This is in line with what we saw in Figure 10 where it became trapped inside a local minima. Note that certain initializations of the algorithm can lead to it finding the true minimum as is shown in the variability of its final results being fairly large, perhaps similar to what we saw with NLM. Both of these have final results which are heavily skewed away from the true minimum as each suffer from over-exploitation. This decreases our confidence in them producing good results in a single run. Looking at the performance of UCB after initialization we can see that it decreases at a far slower rate than EF but still much faster than any other algorithm. This is because the algorithm seeks to explore the search space rather than exploit areas of predicted improvement. However, after around 42 function evaluations it dips below EF meaning that through exploration has found a more optimum solution. In regards to its final results, we see that UCB is far less variable than any of the algorithms seen so far but is still skewed away from the true minimum. EI is the only algorithm that consistently finds the true minimum within 50 function evaluations. After initialization it dips down at a similar rate to EF but just as we saw in Figure 12, at around 20 function evaluations it discovers the global minimum through balancing the competing objectives of exploration and exploitation. Because of this it consistently outperforms every algorithm across the $N = 1000$ runs. This has also led to it having the least variance in final results where we can see it finds the true minimum in the vast majority of runs with only a few outlying results. We can therefore be confident in it finding the global minimum given most random initializations.

Summary of Algorithm Performance We have seen that all benchmark algorithms have been outperformed by their Bayesian optimization counterparts in the given budget of $n = 50$ function evaluations. Because the surface of the Scale Goldstein-Price function has many local minima we saw that algorithms which balance both exploration and exploitation performed the best across the $N = 1000$ runs. EI consistently outperformed all other algorithms in all aspects and was shown to be the most robust to random initialization.

4.5 Ackley Function

This bowl shaped function is evaluated on the square $x_i \in [-5, 5]$ and slopes downwards towards the single global minimum of 0 located at $\mathbf{x} = (0, 0)$. The slopes are not smooth and contain many, also bowl shaped local minima making it difficult for algorithms to travel downwards. We present the analytical form below along with surface and contour plots in Figure 14. After this, we visualize how each algorithm moves across the function surface starting with the benchmarks in Figure 15 after which the Bayesian optimization algorithms with EF, UCB and EI shown in Figures 16, 17 and 18 respectively.

$$f(\mathbf{x}) = -a \exp\left(-b\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos(cx_i)\right) + a + \exp(1)$$

using the recommended values of $a = 20, b = 0.2, c = 2\pi$.

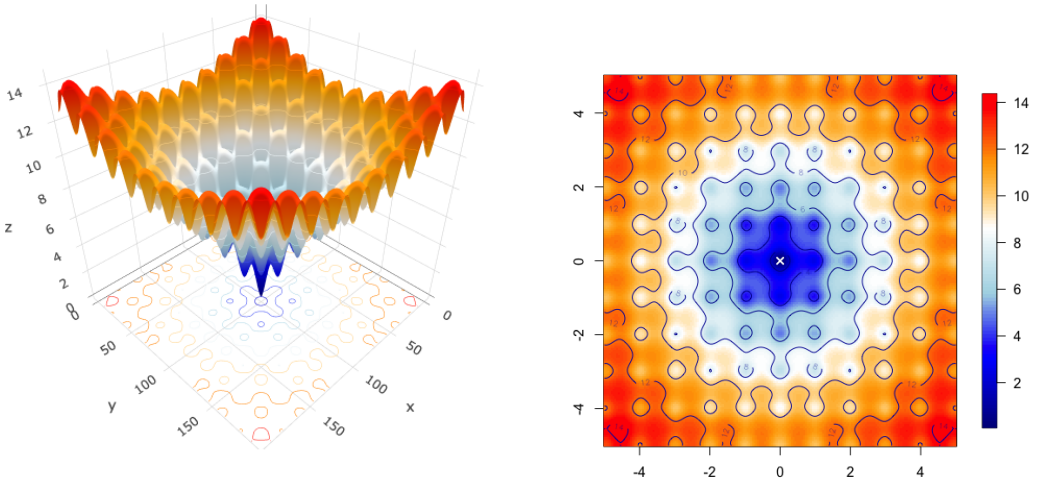
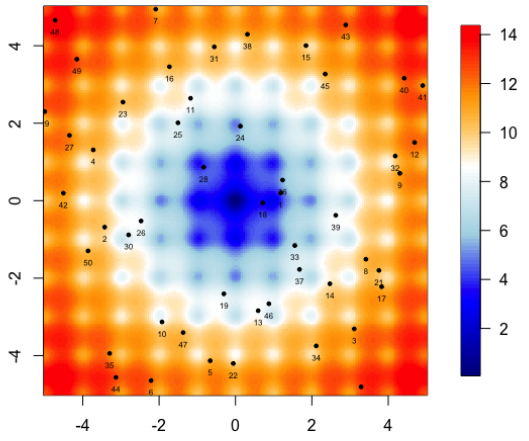
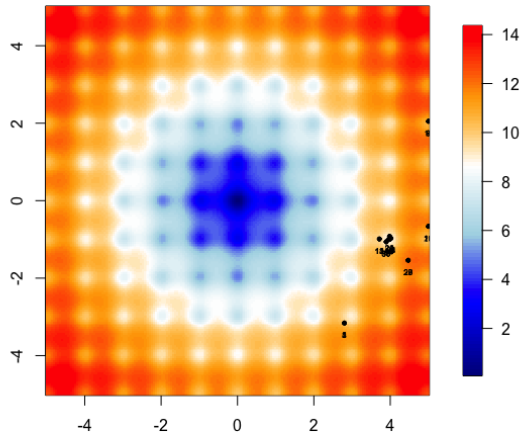


Figure 14: Surface (left) and contour (right) plots of the Ackley function with the location of the global minimum marked by a white x.

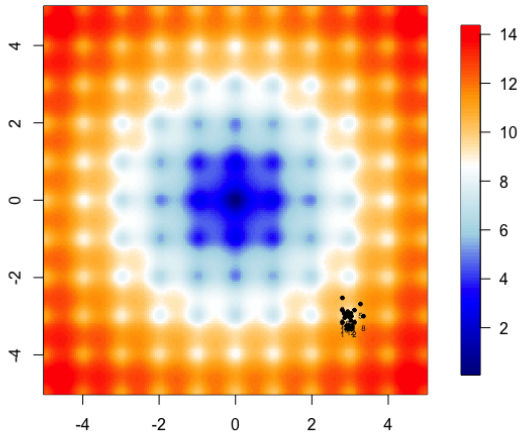
Again we see that each benchmark algorithm has failed to find the global minimum however the performance seems far worse than on the Scale Goldstein-Price function. Both L-BFGS-B and NLM are immediately caught inside one of the bowl shaped local minima which cover the slopes. They then spend all their function evaluations in essentially the same location meaning they have a false sense of convergence. SANN seems to perform far better on this function surface as it has the ability to jump out of the local minima and therefore gets a sense of the surface sloping downwards. It seems to be heading in the right direction but wastes evaluations exploring the surface meaning it does not find the global minimum in the given budget. RS again provides relatively good performance but wastes evaluations in locations that have high values. Only 3 out of the 50 evaluations seem usable and it has also not found the global minimum.



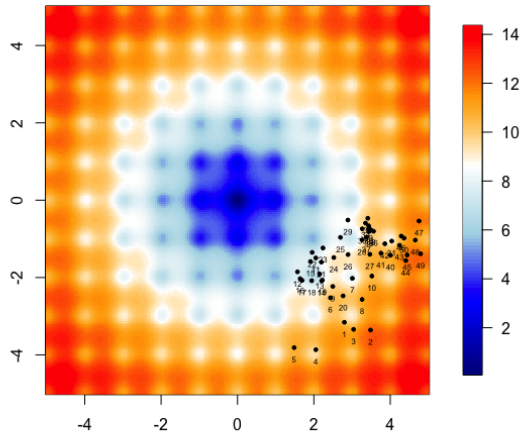
(a) RS



(b) L-BFGS-B



(c) NLM



(d) SANN

Figure 15: Progression of each of the 4 benchmark algorithms across the Ackley function surface with a limited budget of $n = 50$ using the same random initialization. Note that the contour lines have been removed to improve readability. Different starting values can lead to better results in all cases but we present these as examples of the algorithms failing when used naively.

Just as before, we now our attention to the progression of the Bayesian optimization algorithms on the Ackley Function. In Figures 16, 17 and 18 we present the true function surface (top row) posterior mean (middle row) and posterior variance (bottom row) of the EF, UCB and EI acquisition functions respectively. All of which use a Gaussian Process surrogate. To show how each of these surfaces change as the algorithm progresses, we also present 3 snapshots (values of n) within the corresponding rows. Note that each algorithm was again initialized using 12 function evaluations (shown as triangles) at locations sampled through a latin-hypercube scheme.

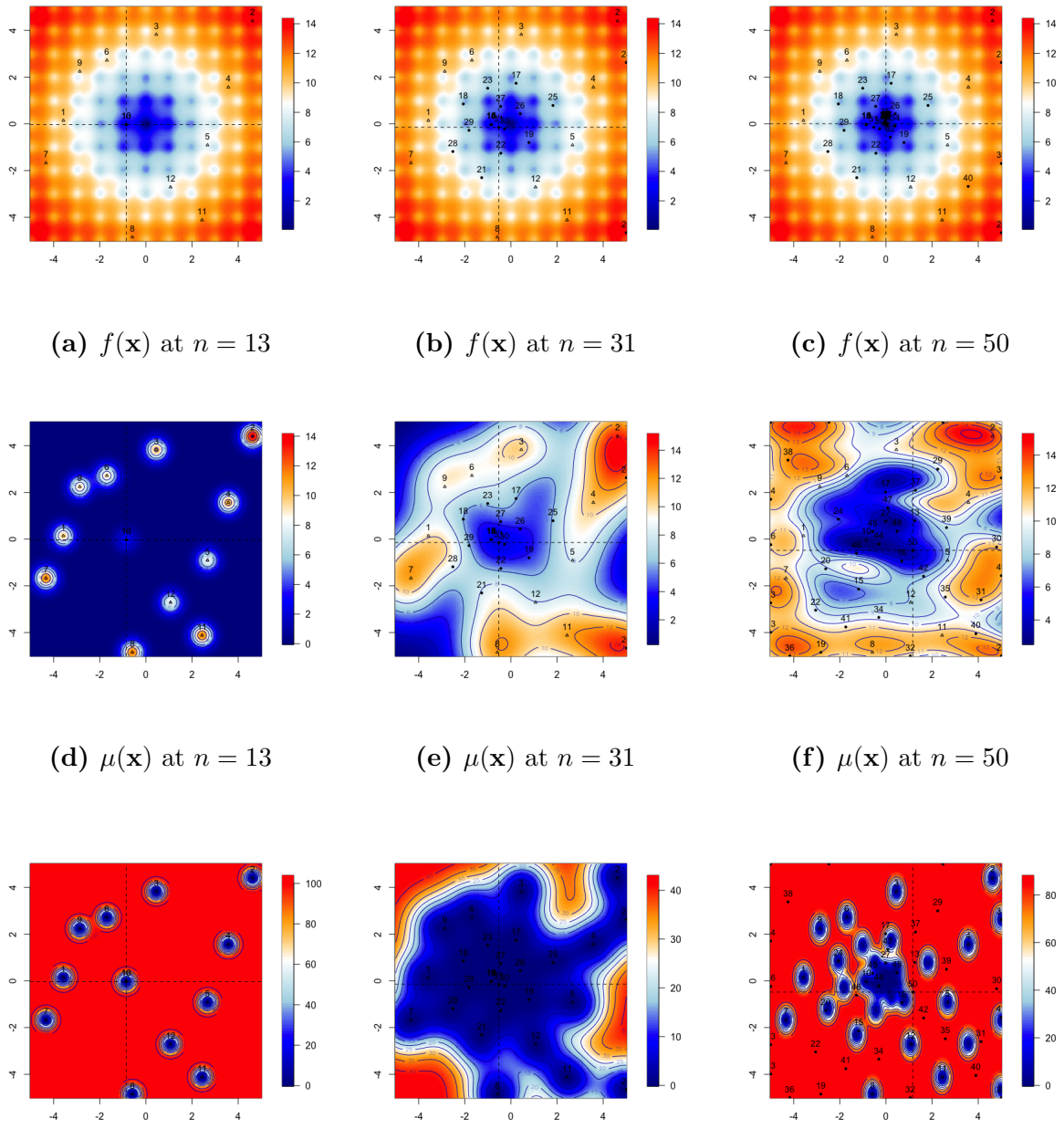
Progression of EF in Figure 16 At $n = 13$ we can see that the surrogate has a relatively poor picture of the function surface. The posterior variance shows that any location far away from those already evaluated has extremely high uncertainty. By chance there has been a single evaluation made inside one of the low lying areas of the function surface. This has led to the posterior mean predicting low values across the all unseen locations while only the locations which have been evaluated have high function values. Because of this the algorithm begins to aggressively exploit the surface and the selected $\mathbf{x}_{n+1}$ is right on top of where the lowest function value was found. At $n = 31$ the posterior mean shows a slightly more accurate picture of the function surface with low value areas predicted towards the center of the surface but has not been able to identify that the surface slopes upwards from this position. We can see that it has begun to aggressively exploit the low lying areas as most function evaluations have taken place around one of the local minima above the global minimum. The posterior variance now shows low uncertainty across the function surface possibly due to its past several evaluations giving fairly similar results. At $n = 50$ we can see that the exploitation in the center of the surface has led to the algorithm getting pretty close to the global minimum. The accuracy of the surrogate surface has also greatly improved and now has a fairly good idea of the true surface given the limited amount of function evaluations. It predicts low value locations towards the center surrounded by contours which increase in value the closer they are towards the edges of the function surface. This is reflected in the posterior variance which shows the algorithm has the highest confidence in the areas it has exploited towards the center.

Progression of UCB in Figure 17 At $n = 13$ we see that the surrogate has a similar picture of the function surface as before. Interestingly, instead of exploring an area of high uncertainty the algorithm has selected $\mathbf{x}_{n+1}$ in the same location as EF. Recall from Equation 3.5 that UCB is a weighted sum between the the posterior mean and variance meaning that this location has maximised this objective. Possibly due to the posterior mean predicting such low values across the function surface. At $n = 32$ we see that the the algorithm has explored a far more diverse set of locations than EF which expended most of its function evaluations in the same location. Because of this the posterior mean has a more accurate picture of the true function surface with low values predicted towards the center of the surface and high value areas surrounding it. The posterior variance is similar to what we saw before and shows the algorithm has low uncertainty in most areas around the actual function evaluations. This could be due to the bowl shaped local minima giving results which are similar in the evaluated locations. At $n = 50$ we see that

there have been a few function evaluations near the global minimum but the algorithm has unfortunately missed its exact location. The posterior mean has a much improved picture of the true function surface and seems more accurate than EF at the same stage. The low value areas are predicted towards the center while the contours show that the algorithm has identified the sloped nature of the surface leading up to the high value areas around the edges. The posterior variance remains similar to what was seen at $n = 32$ and shows that the algorithm is possibly too confident in its predictions given the posterior mean is not exactly accurate. Again this could be due to the bowl shaped nature of the local minima.

Progression of EI in Figure 18 At $n = 13$ we see that the posterior mean and variance are exactly what was seen in both EF and UCB. However, now the selected $\mathbf{x}_{n+1}$ is in a new area of the function surface far from any of the previous evaluations. This location would've provided the maximum expected improvement and thus maximised Equation 3.7. At $n = 34$ we see that the algorithm has done a fair amount of exploration of the function surface but the posterior mean and variance remain similar to what was seen at $n = 13$. At this stage the algorithm has a far worse picture of the true function surface than both EF and UCB which is quite different to what we saw on the Scale Goldstein-Price function. Clearly the many local minima have confused the algorithm and thus it needs more function evaluations to improve performance. At $n = 50$ we can see that the algorithm has both explored a large part of the function surface but exploited the low lying areas far more often than that of UCB. The posterior mean resembles the true function surface much more closely and seems similar to UCB in that the contours show it has identified the bowl shaped nature of the function surface which slopes upwards towards the edges. Unfortunately it has also missed the global minimum but has far more evaluations near its location than UCB. The posterior variance is also similar to what was seen before although shows higher levels of uncertainty in locations which lie far from actual function evaluations.

Now having seen examples of how each algorithm optimizes the Ackley function given a single random initialization, we shift our focus towards their average performance across $N = 1000$ random initializations. Once again this is to provide us with an idea as to the robustness of each algorithm in low budget situations. Because this function surface has many more local minima than what we saw previously, the starting location for each algorithm should play a far larger part in its overall performance.

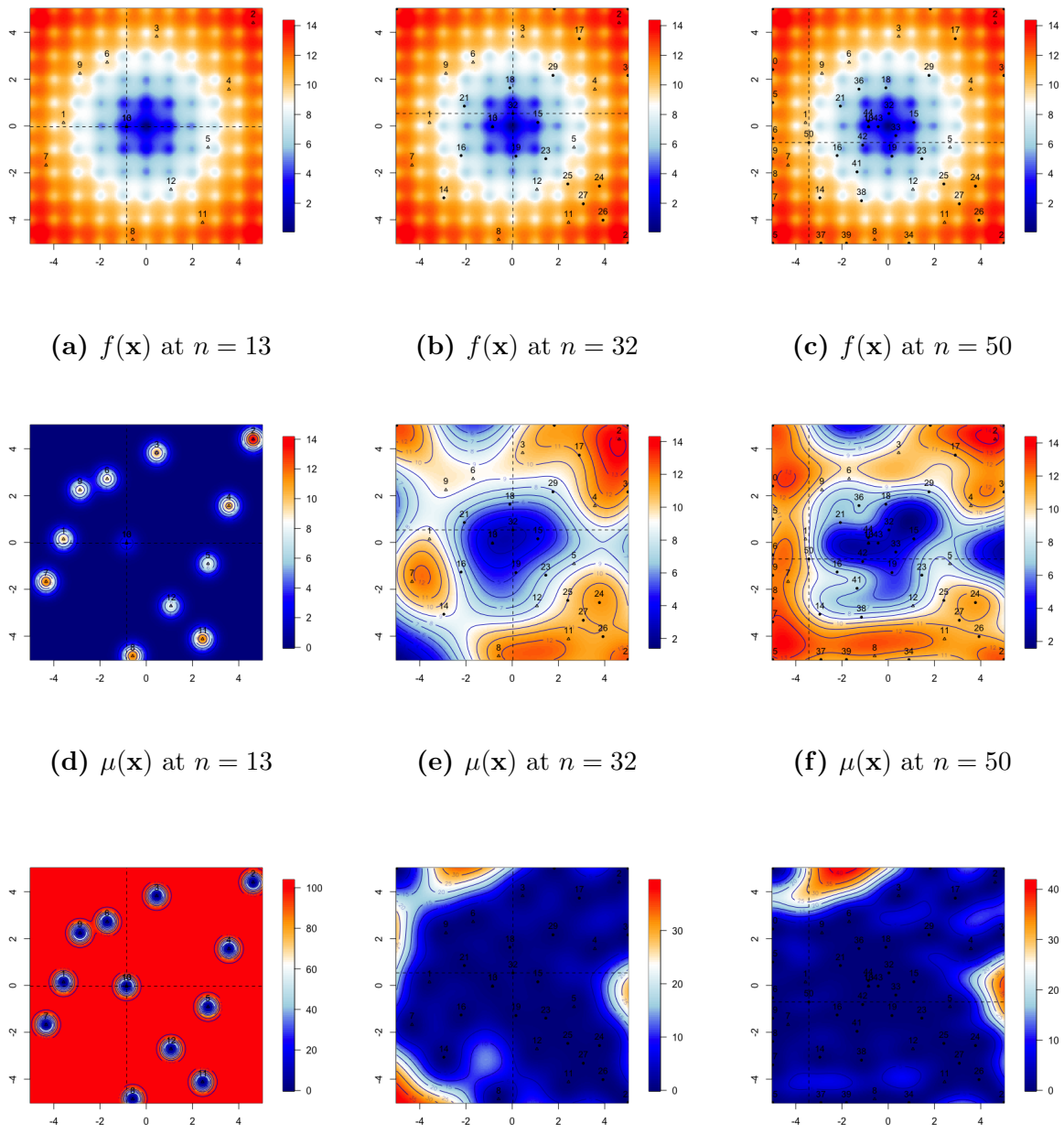


(a) $f(\mathbf{x})$ at $n = 13$ **(b)** $f(\mathbf{x})$ at $n = 31$ **(c)** $f(\mathbf{x})$ at $n = 50$

(d) $\mu(\mathbf{x})$ at $n = 13$ **(e)** $\mu(\mathbf{x})$ at $n = 31$ **(f)** $\mu(\mathbf{x})$ at $n = 50$

(g) $\sigma^2(\mathbf{x})$ at $n = 13$ **(h)** $\sigma^2(\mathbf{x})$ at $n = 31$ **(i)** $\sigma^2(\mathbf{x})$ at $n = 50$

Figure 16: Bayesian optimization algorithm at steps 13, 31 and 50 using the EF acquisition strategy across the Ackley function surface. The top row shows the actual function surface, the middle row shows the posterior mean of our surrogate $\hat{f}$ while the bottom row shows its posterior variance. In each case the $\mathbf{x}_{n+1}$ selected by the algorithm is shown by dotted black lines.

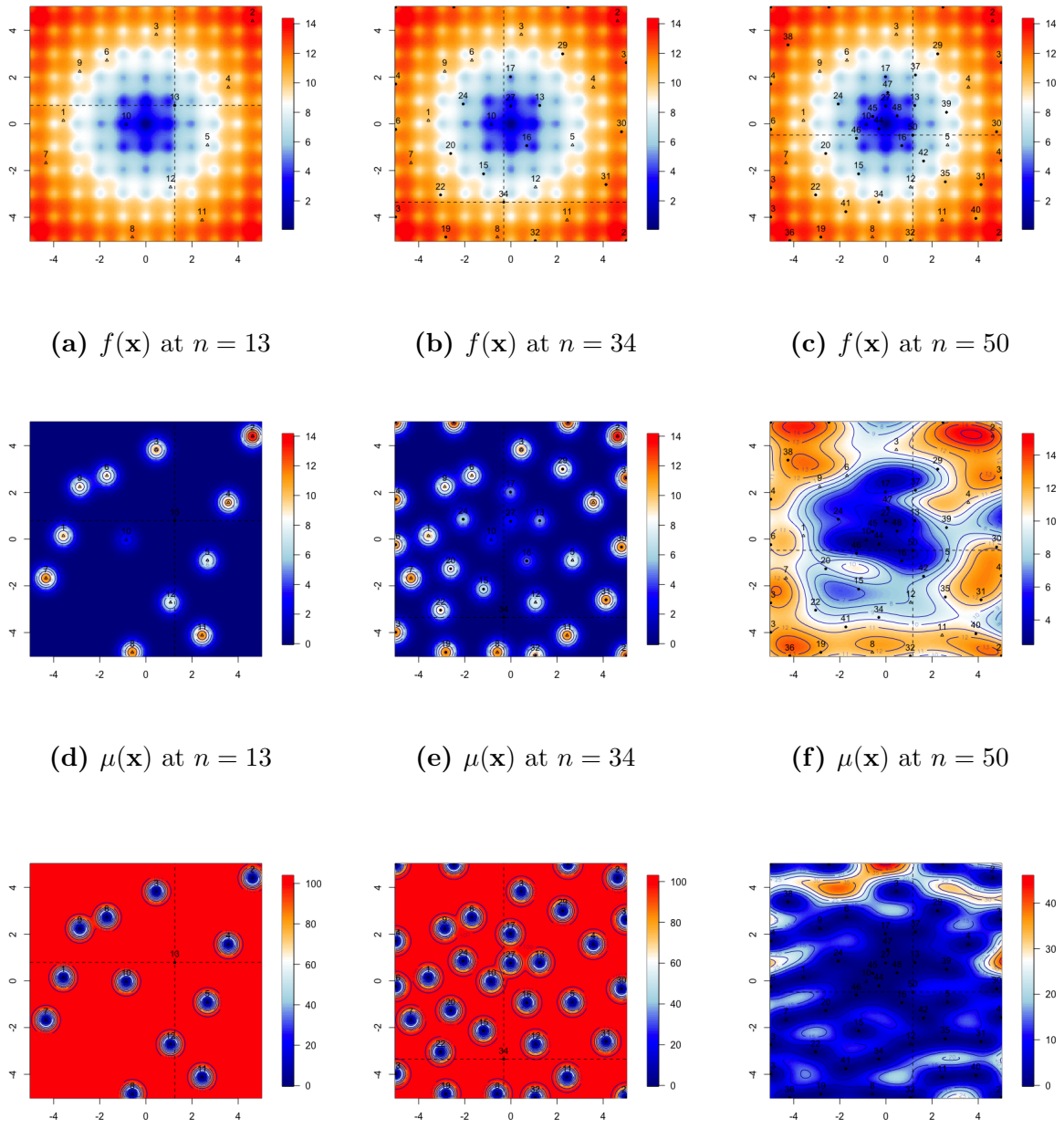


(a) $f(\mathbf{x})$ at $n = 13$ **(b)** $f(\mathbf{x})$ at $n = 32$ **(c)** $f(\mathbf{x})$ at $n = 50$

(d) $\mu(\mathbf{x})$ at $n = 13$ **(e)** $\mu(\mathbf{x})$ at $n = 32$ **(f)** $\mu(\mathbf{x})$ at $n = 50$

(g) $\sigma^2(\mathbf{x})$ at $n = 13$ **(h)** $\sigma^2(\mathbf{x})$ at $n = 32$ **(i)** $\sigma^2(\mathbf{x})$ at $n = 50$

Figure 17: Bayesian optimization algorithm at steps 13, 32 and 50 using the UCB acquisition strategy across the Ackley function surface. The top row shows the actual function surface, the middle row shows the posterior mean of our surrogate $\hat{f}$ while the bottom row shows its posterior variance. In each case the $\mathbf{x}_{n+1}$ selected by the algorithm is shown by dotted black lines.



(a) $f(\mathbf{x})$ at $n = 13$ **(b)** $f(\mathbf{x})$ at $n = 34$ **(c)** $f(\mathbf{x})$ at $n = 50$

(d) $\mu(\mathbf{x})$ at $n = 13$ **(e)** $\mu(\mathbf{x})$ at $n = 34$ **(f)** $\mu(\mathbf{x})$ at $n = 50$

(g) $\sigma^2(\mathbf{x})$ at $n = 13$ **(h)** $\sigma^2(\mathbf{x})$ at $n = 34$ **(i)** $\sigma^2(\mathbf{x})$ at $n = 50$

Figure 18: Bayesian optimization algorithm at steps 13, 34 and 50 using the EI acquisition strategy across the Ackley function surface. The top row shows the actual function surface, the middle row shows the posterior mean of our surrogate $\hat{f}$ while the bottom row shows its posterior variance. In each case the $\mathbf{x}_{n+1}$ selected by the algorithm is shown by dotted black lines.

Presented below in Figure 19 are the results of our Monte Carlo simulation for each algorithm on the Ackley function. To re-iterate, the line plot to the left shows the average BOV after n evaluations of f across $N = 1,000$ randomly initialized algorithm runs. The box plot to the right shows the variance in the final BOV returned by each algorithm after $n = 50$ evaluations of f. Recall that we want to see is quick convergence towards the true minimum in the line plots and as little variability in final results as possible in the box plots. An algorithm with both of these qualities is will provide good performance while remaining robust to random initialization.

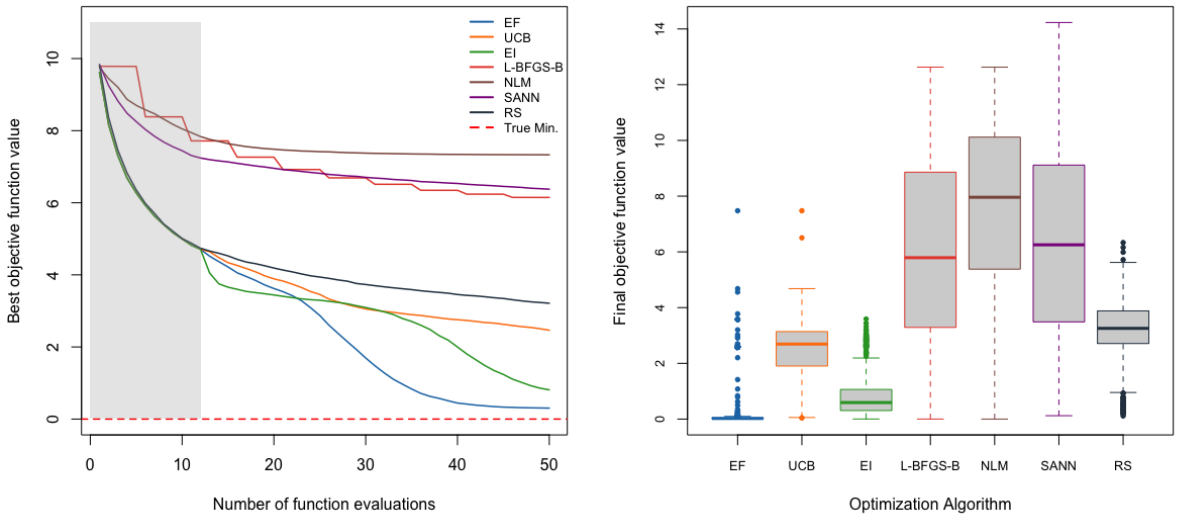


Figure 19: Plots showing the average BOV (across $N = 1000$ runs) achieved by each algorithm after n evaluations of f (left) as well as the variation in the final BOV returned after $n = 50$ (right). The initialization period for the Bayesian optimization algorithms is shown by the shaded region in the line plot.

Performance of Benchmark Algorithms The first thing we notice is the large difference between the benchmark and the Bayesian optimization algorithms with the exception of RS. This speaks to the difficulty of trying to optimize a highly multi-modal function surface such as the Ackley function. Because such a small portion of the function surface contains the low values these algorithms see little benefit from advantageous starting positions which happen far more infrequently than on the Scaled Goldstein-Price function. In contrast to what we saw before, NLM now has the worst performance out of all algorithms. This is unsurprising as we know it takes a geometric approach to the optimization of a function surface. Because of this, it climbs down into one of the many local minima and cannot escape giving the algorithm a false sense of convergence towards the global minimum. Next we see that both SANN and L-BFGS-B provide similarly bad performance, with L-BFGS-B having lower final BOV results slightly more often. Again we notice the distinctive step like pattern shown by L-BFGS-B from making successive gradient calculations in clustered locations across the function surface. In contrast to what

we saw before, it does not look like these algorithms would benefit from more function evaluations as by around $n = 20$ their BOV seems to have plateaued. Again this is due to these algorithms all becoming stuck inside the local minima which lie on the slopes of the function surface. As one would expect, RS far outperforms the other benchmark algorithms. This is because it has no sense of the surface itself and therefore does not become stuck inside any local minima. We see it has a similar BOV curve to what we saw before and slopes rapidly after a few function evaluations but begins to plateau well above the true minimum. NLM, L-BFGS-B and SANN all have similarly large variability in their final results while RS has far less but still relatively variable final results.

Performance of Bayesian Optimization Algorithms Again we see that until that end of the initialization period of 12 function evaluations (shown as a shaded grey region) each of the Bayesian optimization algorithms exactly follow the BOV curve of RS. This already takes them way below what was achieved by the other benchmarks. Just as before, we see after the initialization period each Bayesian optimization algorithm dips well below RS curve meaning that it immediately finds improvements on the BOV's found so far without becoming stuck in local minima. Firstly, we see that EI finds the biggest reduction in BOV and dips far below both EF and UCB. However, this reduction is still far above the true minimum value and seemingly plateaus until around $n = 35$ where it sees another large reduction which unfortunately fails to reach the true minimum value. Although the BOV curve of EF initially dips at a far slower rate than EI we can see that it does not begin to plateau and steadily decreases towards the true minimum which it too fails to consistently achieve. This shows how on this specific surface the favouring of exploitation as opposed to exploration has led to better results, possibly due to the fact the that the highly variable function surface misleads policies which continue to explore. This sentiment is reflected in the fact that the exploration based policy UCB dips at a far slower rate than that of both EF and EI. It therefore plateaus well above the true minimum and provides the worst performance of the three. In regards the the variability of the final results we can see that EF provides the most robust performance across the $N = 1000$ algorithm runs with its median value sitting fairly close to the true minimum value. This is followed by the more variable results from EI which seems to achieve the function minimum in far fewer runs and UCB whose median value is far from the true minimum and the most variable of the three. That being said, each Bayesian optimization algorithm provides both better function values and far more robust final results than each of the benchmarks.

Summary of Algorithm Performance We have seen that once again that all Bayesian optimization algorithms have outperformed the benchmarks given the budget of $n = 50$ function evaluations. This is because most algorithms became stuck in one of the many local minima along the slopes of the function surface. In contrast to the previous function, we saw that EF outperformed both UCB and EI meaning that the policy of exploitation achieved the highest levels of success in this situation as it was able to capitalise on the good function evaluations found during the initialization period.

4.6 Bukin Number 6 Function

This valley shaped function is evaluated on the square $x_i \in [-5, 5]$ and is categorized by smooth, steep slopes which converge into a ridge at the bottom of the valley which has several local minima within it. There is a single global minimum of 0 located at $\mathbf{x} = (5, 0.25)$ which sits inside the ridge but on the edge of the surface. The smooth slopes make it easy for algorithms to head towards the minimum but will often get trapped in one of the many local minima inside the ridge without being able to escape. In addition to this, the ridge where the minima lie is extremely narrow making it difficult to locate. This makes the Bukin Number 6 Function one of the hardest 2D function surfaces to optimize. We present the analytical form below along with surface and contour plots in Figure 20. After this, we visualize how each algorithm moves through the search space starting with the benchmarks in Figure 21 after which the Bayesian optimization algorithms with EF, UCB and EI shown in Figures 22, 23 and 24 respectively.

$$f(\mathbf{x}) = 100\sqrt{|x_2 - 0.01x_1^2|} + 0.01|x_1 + 10|$$

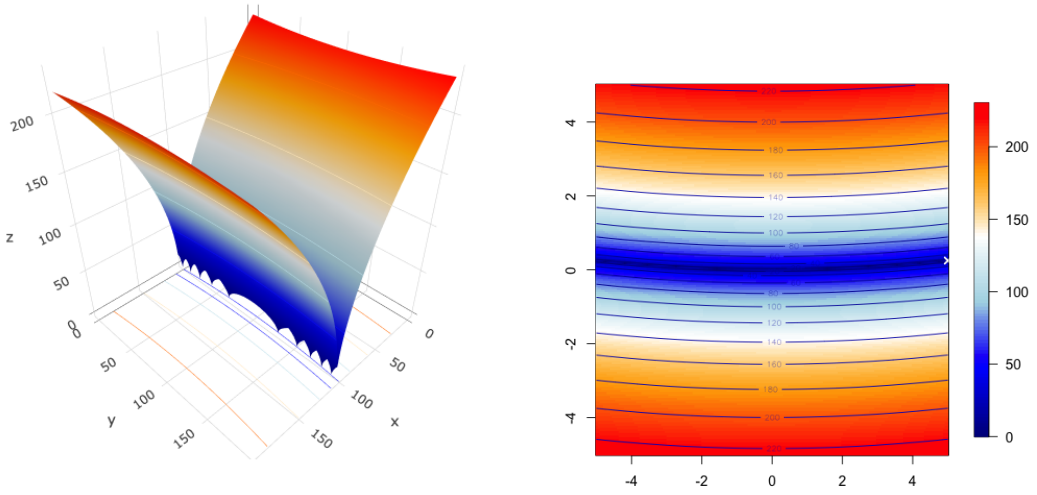
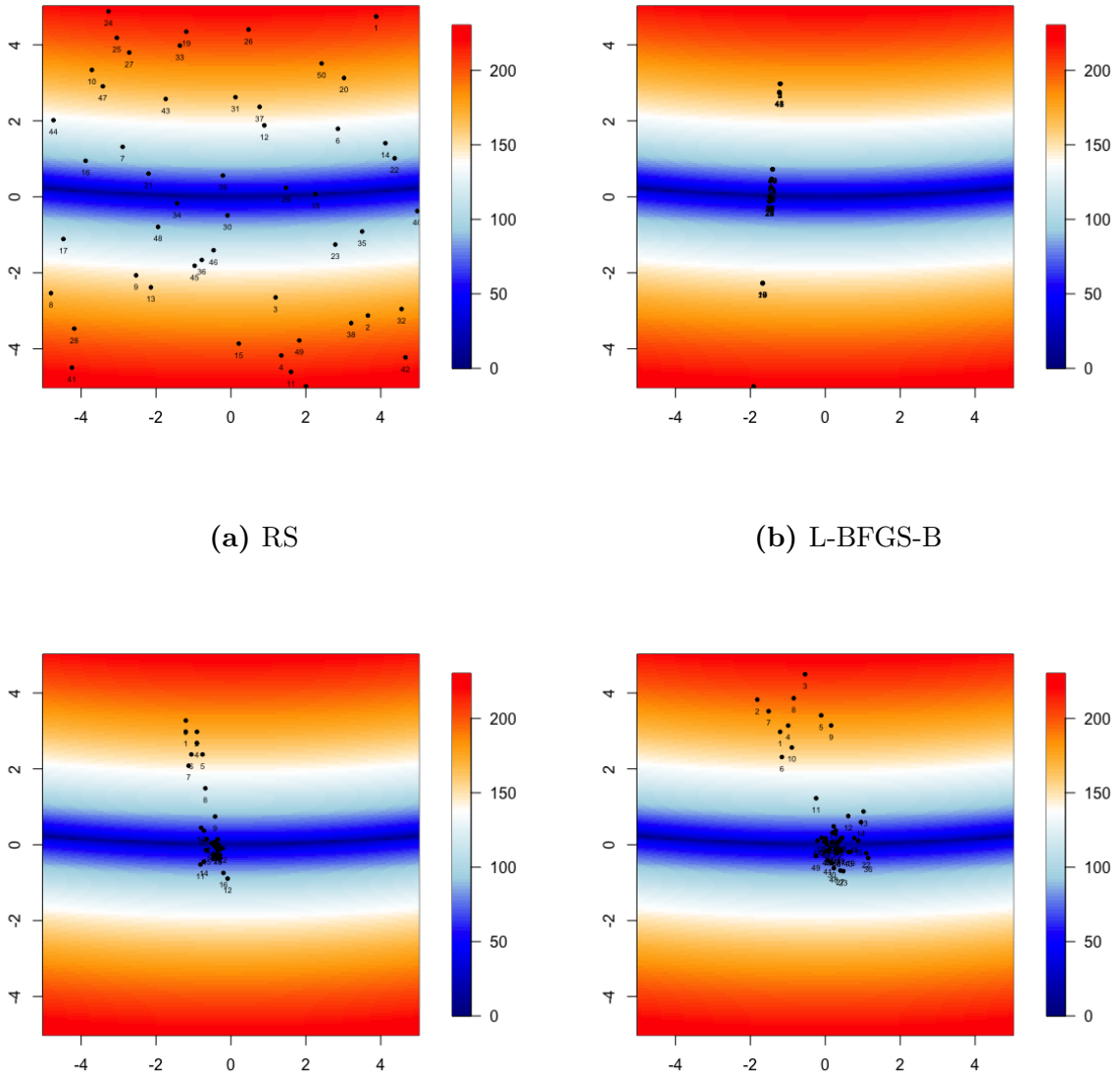


Figure 20: Surface (left) and contour (right) plots of the Bukin Number 6 function with the location of the global minimum marked by a white x.

Here we can see that all of L-BFGS-B, NLM and SANN find it easy to travel down the steep slopes towards the ridge. However when they arrive within one of the local minima they cannot escape due to the steep slopes surrounding it. L-BFGS-B and NLM seem to get stuck in a single minima while SANN is able to explore other local minima although because there are so many, remains fairly close by. Because heading down the slope is so easy the initial starting point plays a massive role in whether or not the algorithm will find the global minimum. Finally, in contrast to the previous two functions the RS algorithm performs quite poorly on this surface. This is because the minima lie in such a narrow ridge that RS fails to consistently evaluate locations within it.



(a) RS

(b) L-BFGS-B

(c) NLM

(d) SANN

Figure 21: Progression of each of the 4 benchmark algorithms across the Bukin Number 6 function surface with a limited budget of $n = 50$ using the same random initialization. Note that the contour lines have been removed to improve readability. Different starting values can lead to better results in all cases but we present these as examples of the algorithms failing when used naively.

Again we turn our attention to the progression of the Bayesian optimization algorithms on the Bukin Number 6 Function. In Figures 22, 23 and 24 we present the true function surface (top row) posterior mean (middle row) and posterior variance (bottom row) of the EF, UCB and EI acquisition functions respectively. All of which use a Gaussian Process surrogate. To show how each of these surfaces change as the algorithm progresses, we present 3 snapshots (values of n) within the corresponding rows. Note that each algorithm was again initialized using 12 function evaluations (shown as triangles) at locations sampled through a latin-hypercube scheme.

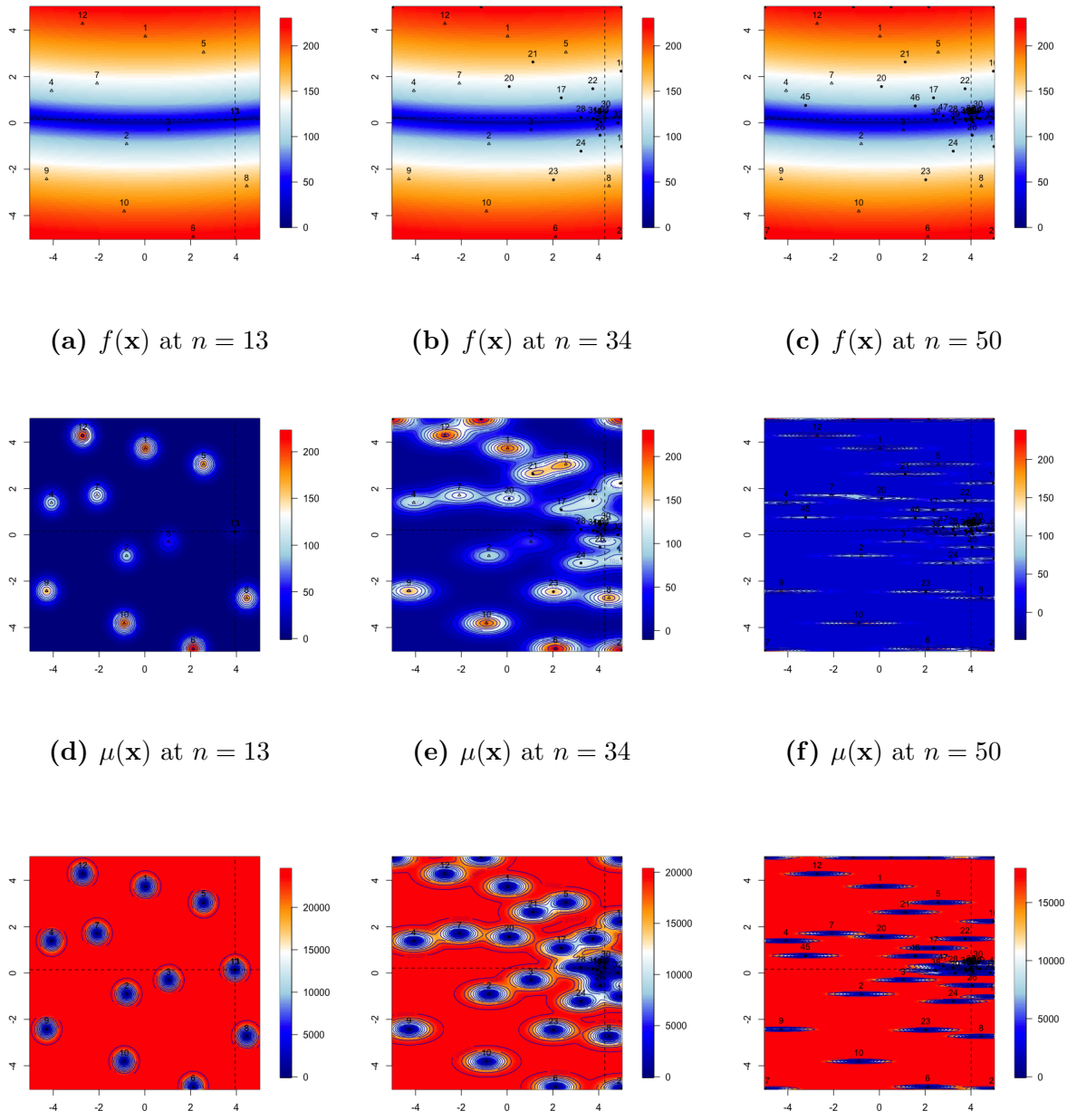
Progression of EF in Figure 22 At $n = 13$ we see that the posterior mean provides a relatively poor reflective of the true function surface. This is due to the high variability in the locations the function has been evaluated at given the steep slopes. Just as before we see that EF incorrectly thinks that most of the surface provides low value function values with the locations that have been evaluated sitting as peaks above everything else. Interestingly we see that the selected $\mathbf{x}_{n+1}$ sits within the ridge at the bottom of the slopes but unfortunately not near the global minimum at the edge of the surface. The posterior variance at this stage is also similar to what we have seen before with high levels of uncertainty across all unevaluated locations. Not much has changed at after $n = 34$ function evaluations as the posterior mean and variance seem fairly similar to what was seen at $n = 13$. Notice that the algorithm is already over exploiting the single location it discovered inside the ridge which is unfortunately inside one of the many local minima within that region. The algorithm seems to have tried a few locations it predicted to have low values but these ended up on the function slopes causing it to quickly return to the area it has evaluated many times before. At $n = 50$ we see that the posterior mean has probably the worst picture of the true function surface we have seen so far. It seems that the characteristics of the narrow ridge has confused the surrogate somewhat and lead to a terrible approximation to the true function surface. This is also reflected in the posterior variance which is similar to $n = 34$ and $n = 13$ but where the the areas of low uncertainty around the evaluated locations have become narrow and elongated, mimicking the characteristics of the ridge. Sadly, the global minimum is never truly found.

Progression of UCB in Figure 23 At $n = 13$ we see that the posterior mean and variance are again similar to that of EF with the selected $\mathbf{x}_{n+1}$ in the same location, just as with the Ackley function. That being said, at $n = 34$ we can see that the algorithm has begun to explore the surface and has evaluated the function at a far more diverse set of locations than EF which just continued to exploit a single area. This exploration has led to the posterior mean becoming a more accurate approximation of the true function surface with high function values predicted at the top and bottom of the function surface with a ridge running through the middle. Although this is more accurate than what we saw with EF, it is still far from being a true reflection. Because the slopes are so steep and the function values so variable it struggles to identify the narrowness of the ridge at this stage. The posterior variance has also changed drastically and seems to have low uncertainty around most locations it has evaluated in contrast to what was seen at $n = 13$. At $n = 50$ we see that the posterior mean is a slightly more accurate approximation of the true function surface with it predicting a narrower ridge along the middle than before.

Although it has evaluated the function at locations which are fairly close, it has also not located the global minimum. Notice that the algorithm has explored many locations towards the top and bottom edges of the surface which has possibly led to the improved understanding of the ridge. The posterior variance is similar to before with the highest levels of uncertainty being located towards the left of the surface away from any actual function evaluations.

Progression of EI in Figure 24 At both $n = 13$ and $n = 34$ we see the same posterior mean and variance as with EF. That is, we see large areas of uncertainty in locations far from those that have been evaluated. We also see that at $n = 34$ this algorithm has explored more of the function surface but this exploration has taken place lower down the slopes than what we saw with UCB because these locations balance the objectives of exploration and exploitation. At this stage the surrogate is a poor approximation of the true function surface and has not spent any evaluations near the global minimum. At $n = 50$ we see that the surrogate is a far better reflection of the function surface and has identified the ridge structure which runs through the middle of the search space. Because it has evaluated the function in more areas along the ridge than UCB the posterior mean seems to be more accurate but is still far from being perfect. We can also see that it has located a similar local minimum to the left of the global minimum which was found by EF, although at a much later stage. The posterior variance is also similar to what we saw with UCB although has more areas of high uncertainty.

Now having seen examples of how each algorithm optimizes the Bukin Number 6 function given a single random initialization, we again shift our focus towards their average performance across $N = 1000$ random initializations. This provides us with an idea as to the robustness of each algorithm in low budget situations. As mentioned earlier the characteristics of this function surface make it fairly difficult to optimize fully. Most algorithms find it easy to travel down the slopes and into the ridges but if given a poor starting location, this can lead to it missing the global minimum completely.

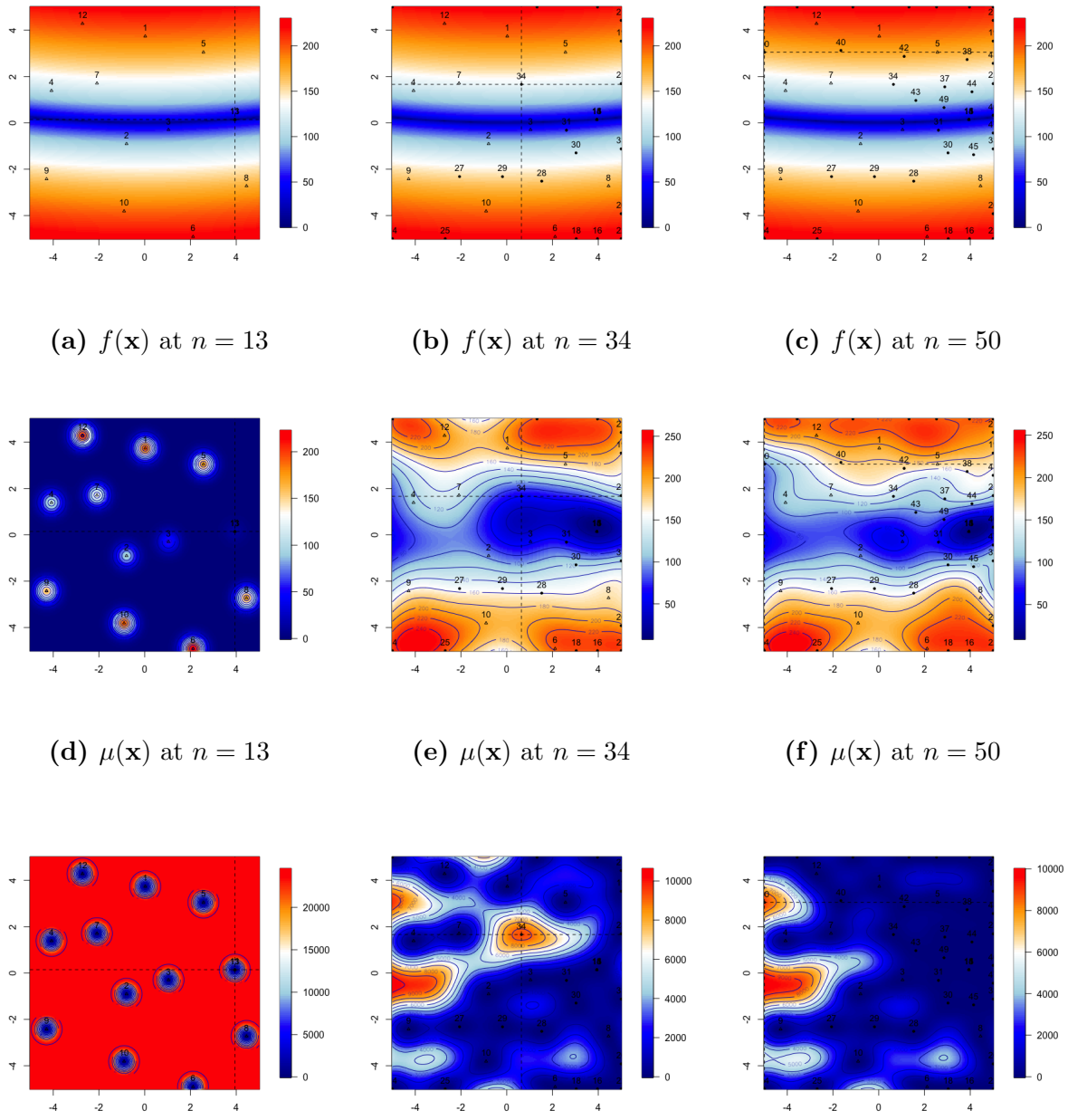


(a) $f(\mathbf{x})$ at $n = 13$ **(b)** $f(\mathbf{x})$ at $n = 34$ **(c)** $f(\mathbf{x})$ at $n = 50$

(d) $\mu(\mathbf{x})$ at $n = 13$ **(e)** $\mu(\mathbf{x})$ at $n = 34$ **(f)** $\mu(\mathbf{x})$ at $n = 50$

(g) $\sigma^2(\mathbf{x})$ at $n = 13$ **(h)** $\sigma^2(\mathbf{x})$ at $n = 34$ **(i)** $\sigma^2(\mathbf{x})$ at $n = 50$

Figure 22: Bayesian optimization algorithm at steps 13, 34 and 50 using the EF acquisition strategy across the Bukin Number 6 function surface. The top row shows the actual function surface, the middle row shows the posterior mean of our surrogate $\hat{f}$ while the bottom row shows its posterior variance. In each case the $\mathbf{x}_{n+1}$ selected by the algorithm is shown by dotted black lines.

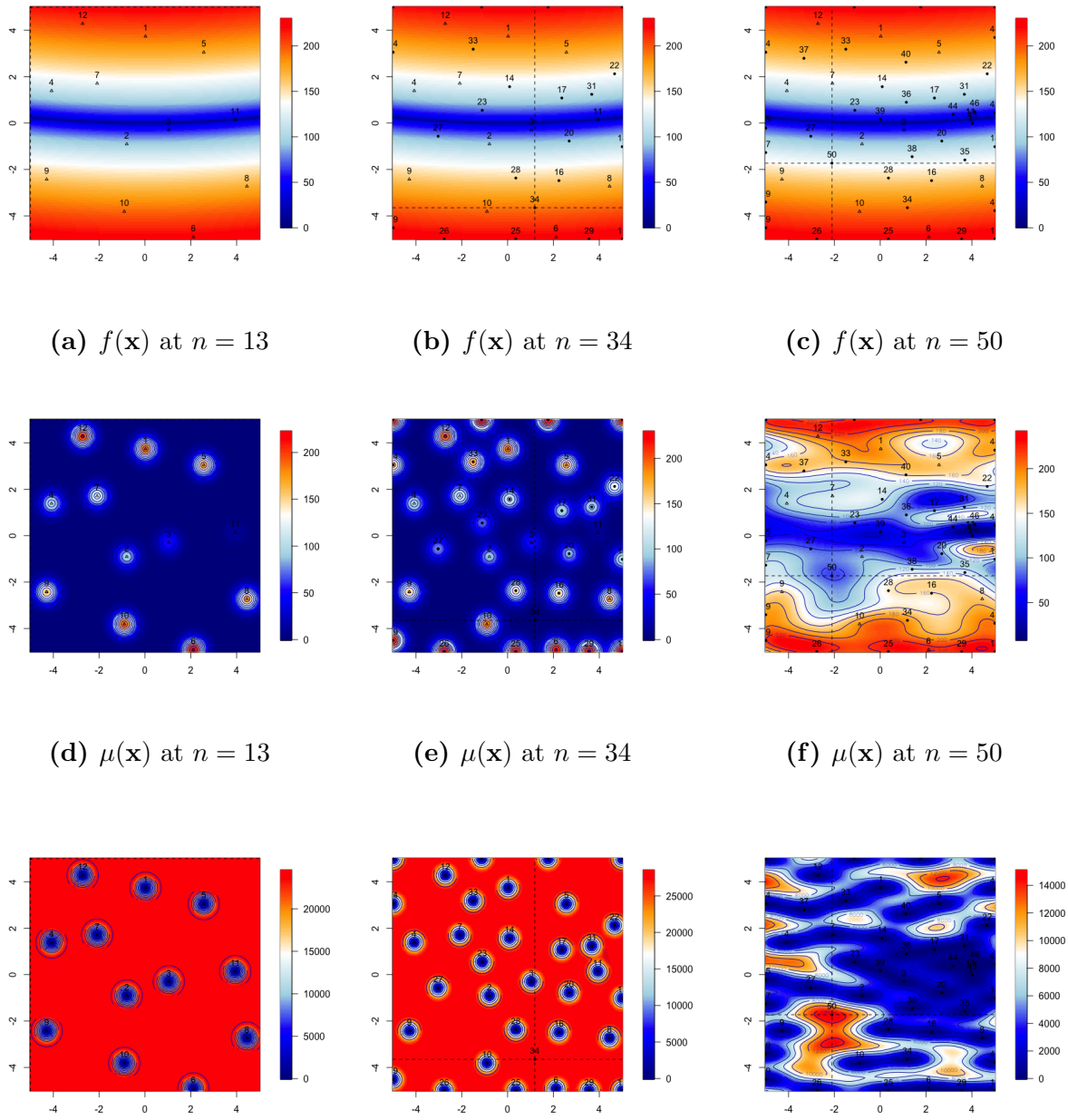


(a) $f(\mathbf{x})$ at $n = 13$ **(b)** $f(\mathbf{x})$ at $n = 34$ **(c)** $f(\mathbf{x})$ at $n = 50$

(d) $\mu(\mathbf{x})$ at $n = 13$ **(e)** $\mu(\mathbf{x})$ at $n = 34$ **(f)** $\mu(\mathbf{x})$ at $n = 50$

(g) $\sigma^2(\mathbf{x})$ at $n = 13$ **(h)** $\sigma^2(\mathbf{x})$ at $n = 34$ **(i)** $\sigma^2(\mathbf{x})$ at $n = 50$

Figure 23: Bayesian optimization algorithm at steps 13, 34 and 50 using the UCB acquisition strategy across the Bukin Number 6 Function. The top row shows the actual function surface, the middle row shows the posterior mean of our surrogate $\hat{f}$ while the bottom row shows its posterior variance. In each case, the $\mathbf{x}_{n+1}$ selected by the algorithm is shown by dotted black lines.



(a) $f(\mathbf{x})$ at $n = 13$ **(b)** $f(\mathbf{x})$ at $n = 34$ **(c)** $f(\mathbf{x})$ at $n = 50$

(d) $\mu(\mathbf{x})$ at $n = 13$ **(e)** $\mu(\mathbf{x})$ at $n = 34$ **(f)** $\mu(\mathbf{x})$ at $n = 50$

(g) $\sigma^2(\mathbf{x})$ at $n = 13$ **(h)** $\sigma^2(\mathbf{x})$ at $n = 34$ **(i)** $\sigma^2(\mathbf{x})$ at $n = 50$

Figure 24: Bayesian optimization algorithm at steps 13, 34 and 50 using the EI acquisition strategy across the Bukin Number 6 Function. The top row shows the actual function surface, the middle row shows the posterior mean of our surrogate $\hat{f}$ while the bottom row shows its posterior variance. In each case, the $\mathbf{x}_{n+1}$ selected by the algorithm is shown by dotted black lines.

Presented below in Figure 25 are the results of our Monte Carlo simulation for each algorithm on the Bukin Number 6 function. To re-iterate, the line plot to the left shows the average BOV after n evaluations of f across $N = 1,000$ randomly initialized algorithm runs. The box plot to the right shows the variance in the final BOV returned by each algorithm after $n = 50$ evaluations of f. Recall that we want to see quick convergence towards the true minimum in the line plots and as little variability in final results as possible in the box plots. An algorithm with both of these qualities will provide good performance while remaining robust to various starting locations.

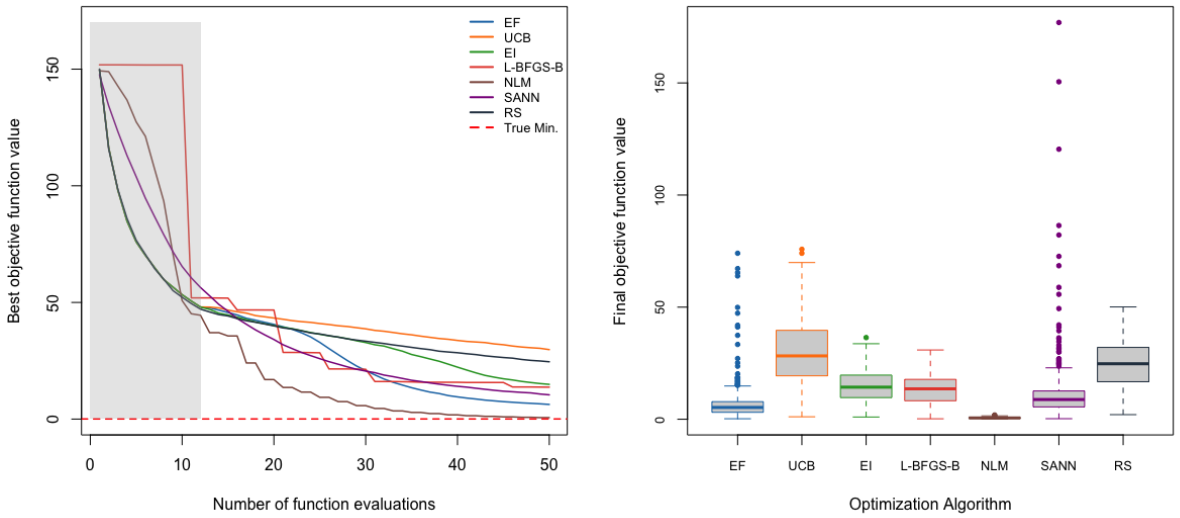


Figure 25: Plots showing the average BOV (across $N = 1000$ runs) achieved by each algorithm after n evaluations of f (left) as well as the variation in the final BOV returned after $n = 50$ (right). The initialization period for the Bayesian optimization algorithms is shown by the shaded region in the line plot.

Performance of Benchmark Algorithms The first thing to notice is that in contrast to the first two test functions, NLM is the algorithm which has consistently performed the best across the $N = 1000$ runs. This is quite a surprising result since the Bayesian optimization algorithms far outperformed all benchmarks in what we have seen up until this point. We can see that the BOV curve for NLM decreases at a rapid rate towards the true minimum which it comes close to on average at around 45 function evaluations. This is possibly due to the shape of the Bukin Number 6 function whose steep slopes are tailor made for the way NLM goes about optimization. The algorithm quickly climbs down the slopes into the ridge where it still has the budget available to search within it. As is shown in the variability of the final results, most random initializations lead to the same outcome meaning the algorithm far outperforms all its counterparts. Both SANN and L-BFGS-B have BOV curves which decrease rapidly after a few function evaluations but plateau just above the true minimum value. This could be due to them both wasting function evaluations either calculating gradients (in the case of L-BFGS-B) or mapping

the function surface (in the case of SANN) and therefore both perform slightly worse than NLM which wastes no time heading down the slopes and into the ridge. This is also reflected in the variability of final results where both are far more variable than that of NLM. SANN is quite skewed towards high function values while L-BFGS-B is slightly less variable but shows most runs failed to find the true minimum value. Another interesting thing to note is that RS has for the first time performed far worse than all other benchmark algorithms. This is due to the the local minima being located in such a narrow ridge along the center of the surface meaning that most randomly sampled locations end up on the slopes above. This is shown in the variability of its final results which is far higher than even SANN bar some major outliers.

Performance of Bayesian Optimization Algorithms Firstly we note that UCB has the worst performance out of all algorithms. This is unsurprising as it inherently favours exploration of the function surface. Because the Bukin Number 6 function has such steep slopes, any function evaluations outside the narrow ridge containing the minimum values return high values. Therefore the fact that UCB is exploring a large part of the function surface has lead to it providing consistently high objective function values. After the initialization period (again shown as a shaded grey region) we see that even RS dips below UCB which seems to immediately plateau and only decrease very slowly until using up its entire evaluation budget. Next we see that after the initialization period, EI does not immediately dip below the RS BOV curve as we saw before. Instead it follows it closely until around 30 function evaluations at which point it seems to identify an area within the ridge which it can more readily exploit. Again note that EI seeks to balance the competing objectives of exploration and exploitation and therefore has performed far better than UCB. In contrast to both of these, EF is a purely exploitation based strategy and once finding a single low value area it began to aggressively exploit this location. This has led to it outperforming both EI and UCB as it has far fewer function evaluations on the function slopes skewing its results. After the initialization period, EF follows the BOV curve of RS closely until it evaluates a location inside the ridge at which point it quickly dips far below all other algorithms - only outperformed by NLM. In terms of the variability in final results we see that EI has similar variability to L-BFGS-B and does not consistently find the true minimum. UCB has the most variable results out of all algorithms due to its policy of exploration while EF has reasonably small variability in its results, only skewed by a few outlying observations which are possibly the result of particularly bad random initializations.

Summary of Algorithm Performance In contrast to previous functions, the simplex based NLM algorithm has outperformed all algorithms on this function surface due to its shape. This highlights a weakness in the Bayesian optimization algorithm and shows that although it works well in many situations can sometimes perform worse than more traditional alternatives. In addition we have seen that exploitation based Bayesian optimization strategies have shown better performance than those which favour exploration.

4.7 Summary of Results

Having analysed the performance of both benchmark and Bayesian optimization algorithms across each test function in the previous sections we now present the results of the Monte Carlo simulations in table format. Note that the numbers seen below in Table 3 are what was visualised by the box plots in the previous sections and are now given as a more precise description of algorithm performance.

Table 3: Summary statistics of the Monte Carlo simulation showing the final BOV achieved by both the benchmarh and Bayesian optimization algorithms across the evaluated test functions. In each case the function which performed the best is shown in bold.

Test Function	Algorithm	True Min.	Std. Dev.	Min	Pctl. 25	Median	Pctl. 75	Max
Scaled Goldstein-Price	EF	-3.1	0.54	-3.1	-3.1	-3.1	-2.2	-0.8
	UCB	-3.1	0.45	-3.1	-3.1	-3.1	-2.8	-1.6
	EI	**-3.1**	**0.15**	**-3.1**	**-3.1**	**-3.1**	**-3.1**	**-1.7**
	L-BFGS-B	-3.1	0.65	-3.1	-2.3	-2	-1.7	-0.51
	NLM	-3.1	0.72	-3.1	-3.1	-3.1	-1.8	-0.36
	SANN	-3.1	1	-3.1	-1.2	-0.64	0.28	2.1
	RS	-3.1	0.39	-3.1	-2.5	-2.2	-2	-1.1
Ackley	**EF**	**0**	**0.85**	**0.001**	**0.011**	**0.022**	**0.047**	**7.5**
	UCB	0	0.96	0.045	1.9	2.7	3.1	7.5
	EI	0	0.71	0.003	0.31	0.59	1.1	3.6
	L-BFGS-B	0	3.4	0.001	3.3	5.8	8.9	13
	NLM	0	3.2	0.001	5.4	8	10	13
	SANN	0	3.4	0.12	3.5	6.3	9.1	14
	RS	0	0.99	0.11	2.7	3.3	3.9	6.3
Bukin Number 6	EF	0	6	0.22	3.1	5.3	7.8	74
	UCB	0	14	1.1	19	28	40	76
	EI	0	6.9	0.99	9.7	14	20	36
	L-BFGS-B	0	7.3	0.21	8.3	14	18	31
	NLM	**0**	**0.32**	**0.076**	**0.37**	**0.59**	**0.84**	**1.9**
	SANN	0	11	0.25	5.5	8.8	13	177
	RS	0	11	2.1	17	25	32	50

In the case of the Scaled Goldstein-Price function we have seen that the EI algorithm performed by far the best across the $N = 1,00$ simulated runs. This is because it was able to strike a balance between exploitation and exploration leading to quick convergence towards the global minimum. Next we saw that the exploitation heavy EF performed slightly better than EI on the Ackley function. In both these cases the Bayesian optimization algorithms greatly outperformed the benchmarks. Finally we saw a pathological example for Bayesian optimization in the form of the Bukin Number 6 function. The shape of this function with its steep and smooth slopes meant it was well suited to NLM which outperformed all other algorithms. This shows how there exists no panacea for truly global optimization and that special care needs to be taken when dealing with non-convex and highly multi-modal function surfaces.

We note that our study has made use of test functions for which there is a known analytical form and therefore Bayesian optimization would mostly be unnecessary. Although this allowed for a robust analysis of algorithm performance when dealing with real world situations like those described in Section 3.1, we would not have an analytical function to optimize. This is where Bayesian optimization will shine as it allows us to fit a non-parametric surrogate model to a dataset containing only a set of inputs and outputs. What we have shown is that this process can sometimes even outperform optimization routines

which do require some functional form to be effective. This is why Bayesian optimization is so celebrated as a *black-box* optimization technique. In addition to this Equation 3.7 showed us that in low budget settings EI can give us guarantees that the next location sampled will be one that maximises expected improvement providing us with a considered approach to the sequential design of expensive experiments through active learning.

4.8 Suggestions for Future Study

In this study of algorithm performance we have compared the Bayesian optimization algorithm using three different acquisition functions which were chosen to show a diverse range of optimization policies. However, as we saw in Section 3.3 there are many more policies in existence - some of which might provide improved performance. Future studies could build upon what was presented in the previous sections and compare new acquisition functions to those already evaluated. In addition to this, we briefly mentioned how Bayesian optimization is often used in the optimization of noisy functions however this study analysed only deterministic function surfaces. During the analysis we found that adding Gaussian noise to the outputs from each function did not change the results obtained in the Monte Carlo simulation in terms of which algorithm performed best. It seemed to only decrease each algorithms BOV curves while they remained the same shape. It was therefore excluded as it did not add to our discussion. Future work can look to find a better way to include the effects of noisy function optimization across each of the algorithms and test functions. Finally we suggest that future work apply the Bayesian optimization algorithm to more real world problems, for example the hyperparameter tuning of a large scale machine learning algorithm. Many papers have shown this to be an area where Bayesian optimization algorithms can excel and even outperform human experts through automatic hyperparameter selection (Shahriari et al., 2016). This was investigated briefly using the MNIST dataset but ultimately also excluded in favour of the use of the analytical test functions listed in Table 1 which provided a more precise way to benchmark algorithm performance.

5 Conclusion

We began this project with a discussion on the goal of statistical learning being to approximate the underlying functional relationship between a set of predictors and some quantitative response. The Gaussian Process was then shown to provide a Bayesian framework with which to model this relationship through the creation of a probabilistic mapping known as a surrogate function. This was achieved through the specification of a kernel which captured our prior beliefs on the functional form of the underlying relationship and saw how it could be learnt from the data. An emphasis was then made on the practical importance of a surrogate function in situations where data collection (or evaluation of the functional relationship) is either computationally or commercially expensive. This lead to a discussion on the Bayesian optimization algorithm and its popularity as a *black-box* optimization technique. Here, the Gaussian Process surrogate was used in conjunction with an acquisition function to balance the competing objectives of exploration and exploitation, providing a framework for efficient global optimization of non-convex and multi-modal function surfaces. After this we compared the performance of the Bayesian optimization algorithm to several popular optimization routines across three analytical test functions through a Monte Carlo simulation experiment. This showed that Bayesian optimization performs extraordinarily well in low budget settings where function surfaces are believed to contain many local minima. However, we also saw a case in which the shape of the function surface lead to the Nelder-Mead optimization routine outperforming its Bayesian counterparts highlighting the fact that their use may sometimes be overkill. One should first consider their objectives and assess the suitability of any chosen method. For example given a smooth, convex surface for which there exists an analytical form it would be fairly unnecessary to implement a Bayesian optimizer when out-of-the-box routines like Nelder-Mead can provide similar, and even sometimes better performance. However, if the function we are trying to optimize has no analytical form as well as evaluations of this function are expensive or impossible to acquire, the Bayesian optimization framework should be favoured.

References

D. Ackley. *A Connectionist Machine for Genetic Hillclimbing*, volume SECS28 of *The Kluwer International Series in Engineering and Computer Science*. Kluwer Academic Publishers, Boston, 1987.

J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012. ISSN ISSN 1533-7928. URL http://www.jmlr.org/papers/v13/bergstra12a.html.

A. J. Booker, J. E. Dennis, P. D. Frank, D. B. Serafini, V. Torczon, and M. W. Trosset. A rigorous framework for optimization of expensive functions by surrogates. *Structural optimization*, 17(1):1–13, 1999. doi: 10.1007/BF01197708. URL https://doi.org/10.1007/BF01197708.

R. Byrd, P. Lu, J. Nocedal, and C. Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16:1190–1208, Sept. 1995. ISSN 1064-8275. doi: 10.1137/0916069.

C. J. P. Bélisle. Convergence theorems for a class of simulated annealing algorithms on $\mathbb{R}^d$. *Journal of Applied Probability*, 29(4):885–895, 1992. doi: 10.2307/3214721. URL https://www.jstor.org/stable/3214721.

I. M. Christopher K. I. Williams. A framework for evaluating approximation methods for gaussian process regression. *Journal of machine learning research*, 14(1):333–350, 2013. ISSN 1532-4435.

M. de Benito Delgado and P. Wacker. Bayesian model selection for linear regression, 2015.

L. C. W. Dixon and G. P. Szego. *The Global Optimization Problem: An Introduction*, pages 1–15. North-Holland Pub. Co, Amsterdam, 1978.

C. C. Drovandi. Principles of experimental design for big data analysis. *Statistical science*, 32(3):385–404, 2017. ISSN 0883-4237.

P. I. Frazier. A tutorial on bayesian optimization, 2018.

J. H. Friedman. Multivariate Adaptive Regression Splines. *The Annals of Statistics*, 19(1):1 – 67, 1991. doi: 10.1214/aos/1176347963. URL https://doi.org/10.1214/aos/1176347963.

R. Garnett. *Bayesian Optimization*. Cambridge University Press, 2023.

J. Gonzalvez, E. Lezmi, T. Roncalli, and J. Xu. Financial applications of gaussian processes and bayesian optimization, 2019.

R. B. Gramacy. laGP: Large-scale spatial modeling via local approximate gaussian processes in R. *Journal of Statistical Software*, 72(1):1–46, 2016a. doi: 10.18637/jss.v072.i01.

R. B. Gramacy. laGP: Large-scale spatial modeling via local approximate gaussian processes in R. *Journal of Statistical Software*, 72(1):1–46, 2016b. doi: 10.18637/jss.v072.i01.

R. B. Gramacy. *Surrogates: Gaussian Process Modeling, Design and Optimization for the Applied Sciences*. Chapman Hall/CRC, Boca Raton, Florida, 2020. http://bobby.gramacy.com/surrogates/.

R.-R. Griffiths. Applications of gaussian processes at extreme lengthscales: From molecules to black holes. 2023. doi: 10.17863/CAM.93643. URL https://www.repository.cam.ac.uk/handle/1810/346223.

L. Grippo, F. Lampariello, and S. Lucidi. A truncated newton method with nonmonotone line search for unconstrained optimization. *Journal of Optimization Theory and Applications*, 60(3):401–419, 1989. doi: 10.1007/BF00940345. URL https://doi.org/10.1007/BF00940345.

Y.-S. O. Haitao Liu. When gaussian process meets big data: A review of scalable gps. *IEEE transaction on neural networks and learning systems*, 31(11):4405–4423, 2020. ISSN 2162-237X.

J. S. HaiYing Wang, Min Yang. Information-based optimal subdata selection for big data linear regression. *Journal of the American Statistical Association*, 114(525):393–405, 2019. ISSN 0162-1459.

P. Hennig, M. A. Osborne, and M. Girolami. Probabilistic numerics and uncertainty in computations. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 471(2179):20150142, jul 2015. doi: 10.1098/rspa.2015.0142. URL https://doi.org/10.1098%2Frspa.2015.0142.

G. James, D. Witten, T. Hastie, and R. Tibshirani. *An Introduction to Statistical Learning: with Applications in R*. Springer, 2013. URL https://faculty.marshall.usc.edu/gareth-james/ISL/.

D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black-box functions. *J. of Global Optimization*, 13(4):455–492, dec 1998. ISSN 0925-5001. doi: 10.1023/A:1008306431147. URL https://doi.org/10.1023/A:1008306431147.

D. G. Krige. A statistical approaches to some basic mine valuation problems on the witwatersrand. *Journal of the Chemical, Metallurgical and Mining Society of South Africa*, 52:119–139, 1951.

G. Matheron. Principles of geostatistics. *Economic Geology*, 58:1246–1266, 1963.

N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953. doi: 10.1063/1.1699114. URL http://link.aip.org/link/?JCP/21/1087/1.

J. Močkus. On bayesian methods for seeking the extremum. In G. I. Marchuk, editor, *Optimization Techniques IFIP Technical Conference Novosibirsk, July 1–7, 1974*, pages 400–404, Berlin, Heidelberg, 1975. Springer Berlin Heidelberg. ISBN 978-3-540-37497-8.

J. A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

M. Z. Ngwenya. Investigating 'optimal' kriging variance estimation: Analytic and bootstrap estimators. Master's thesis, University of Cape Town, South Africa, 2011.

F. Nogueira. Bayesian Optimization: Open source constrained global optimization tool for Python, 2014. URL https://github.com/fmfn/BayesianOptimization.

T. O'Malley, E. Bursztein, J. Long, F. Chollet, H. Jin, L. Invernizzi, et al. Keras Tuner. https://github.com/keras-team/keras-tuner, 2019.

V. Picheny, T. Wagner, and D. Ginsbourger. A benchmark of kriging-based infill criteria for noisy optimization. *Struct. Multidiscip. Optim.*, 48(3):607–626, sep 2013. ISSN 1615-147X. doi: 10.1007/s00158-013-0919-4. URL https://doi.org/10.1007/s00158-013-0919-4.

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/.

C. E. Rasmussen. *Gaussian Processes in Machine Learning*, pages 63–71. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. ISBN 978-3-540-28650-9. doi: 10.1007/978-3-540-28650-9_4. URL https://doi.org/10.1007/978-3-540-28650-9_4.

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

T. Santner, B. Williams, B. Williams, and W. Notz. *The Design and Analysis of Computer Experiments*. Springer Series in Statistics. Springer, 2003. ISBN 9780387954202. URL https://books.google.co.za/books?id=O1itua2QzFkC.

M. Schonlau. Computer experiments and global optimization, 1997. URL http://hdl.handle.net/10012/190.

B. Shahriari, K. Swersky, Z. Wang, R. Adams, and N. De Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, Jan. 2016. ISSN 0018-9219. doi: 10.1109/JPROC.2015.2494218. Publisher Copyright: © 1963-2012 IEEE.

Z. K. Silagadze. Finding two-dimensional peaks. *Physics of Particles and Nuclei Letters*, 4(1):73–80, 2007. doi: 10.1134/S154747710701013X. URL https://doi.org/10.1134/S154747710701013X.

N. Srinivas, A. Krause, S. M. Kakade, and M. W. Seeger. Information-theoretic regret bounds for gaussian process optimization in the bandit setting. *IEEE Transactions on Information Theory*, 58(5): 3250–3265, may 2012. doi: 10.1109/tit.2011.2182033. URL https://doi.org/10.1109%2Ftit.2011.2182033.

S. Surjanovic and D. Bingham. Virtual library of simulation experiments. Accessed September 2023, 2013. URL https://www.sfu.ca/~ssurjano/index.html.

M. N. Thombre, H. A. Preisig, and M. B. Addis. Developing surrogate models via computer based experiments. In K. V. Gernaey, J. K. Huusom, and R. Gani, editors, *12th International Symposium on Process Systems Engineering and 25th European Symposium on Computer Aided Process Engineering*, volume 37 of *Computer Aided Chemical Engineering*, pages 641–646. Elsevier, 2015. doi: https://doi.org/10.1016/B978-0-444-63578-5.50102-X. URL https://www.sciencedirect.com/science/article/pii/B978044463578550102X.

C. Williams and C. Rasmussen. Gaussian processes for regression. In D. Touretzky, M. Mozer, and M. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8. MIT Press, 1995. URL https://proceedings.neurips.cc/paper_files/paper/1995/file/7cce53cf90577442771720a370c3c723-Paper.pdf.