University of Glasgow

School of Mathematics and Statistics

Level-5 Honours project

Bayesian Optimisation Using Gaussian Processes

Author:
Paulius Leniauskas
2207785L

Supervisor:
Dr. Mu Niu

Declaration of Originality
I confirm that this assignment is my own work and that I have:

- *Read and understood the guidance on plagiarism in the Student Handbook, including the University of Glasgow Statement on Plagiarism*
- *Not made use of the work of any other student(s) past or present without acknowledgment. This includes any of my own work, that has been previously, or concurrently, submitted for assessment, either at this or any other educational institution, including school*
- *Not sought or used the services of any professional agencies to produce this work*
- *In addition, I understand that any false claim in respect of this work will result in disciplinary action in accordance with University regulations*
- *I am aware of and understand the University's policy on plagiarism and I certify that this assignment is my own work, except where indicated by referencing, and that I have followed the good academic practices noted above.*

Thursday 25th March, 2021

Abstract

Many optimisation methods help us to develop autonomous systems which allow us to work faster, more efficiently and more precisely. However, in many cases not only the explicit form of the function being optimised is unknown, but it is often expensive and/or time consuming to evaluate that objective function, making the optimisation impractical. This paper offers an in-depth analysis of the method known as Bayesian optimisation, which defines a strategy of dealing with unknown, multidimensional functions and provides us with tools required to solve this kind of optimisation problem efficiently, avoiding the unnecessary monetary costs or consumption of time.

The paper will start with a brief discussion of Gaussian distribution properties followed by an introduction of Gaussian processes and Gaussian process regression - the most important tools used to implement Bayesian optimisation algorithm. Analysis of every step of Bayesian optimisation algorithm is performed and examples of applications of this method for one and two-dimensional problems are presented using computationally derived data - dedicated packages within programming language Python are implemented to calculate and visualise the results. Additionally, performance of Bayesian optimisation at each one and two-dimensional cases is compared to the gradient-based optimisation tool *optim* in R. Finally, the paper concludes with a discussion of possible further analysis, applications and future development of the Bayesian optimisation method.

Contents

List of Figures

1 Introduction

Consider this hypothetical situation: you are the head of an oil drilling company and you have located an area which contains an unknown varying amount of oil resources deep underneath the surface of the earth. Since building drilling rigs cost millions of pounds and drilling a hole deep into the earth might take months, in order to get maximum rewards, one ideally wants to build the least possible number of drilling platforms and place them at those locations within the area, which contain the richest oil reservoirs. So how does one decide where to build the drilling rigs that would extract the greatest amount of oil in the given area without exact knowledge of where oil lies beneath the surface of the earth? Finding an answer to this question is an example of solving an optimisation problem - finding the best solution from all feasible solutions.

The majority of the optimisation problems [6] in many different ways aim to optimise (often maximise or minimise) some kind of objective function with respect to various possible constraints and design parameters. There exist many approaches of solving these problems based on the properties of the objective function itself and the general question of interest. For instance, in the example above, if we treat the unknown distribution of oil as our continuous objective function and possible drilling locations as the input of that function, the goal of the optimisation simplifies to finding a value which maximises our objective (or corresponds to the richest oil reservoir spot in the area). However, not knowing the exact form our objective function takes is what makes this global optimisation problem complex - we can evaluate it (by building an extremely expensive and time-consuming drilling rig and actually drilling the surface of the earth) at our point of choice and only then find out what value (amount of oil) it takes.

An excellent method of solving this and other similar problems, known as Bayesian optimisation, was first introduced by Lithuanian mathematician Jonas Mockus in his series of publications on global optimisation in the 1980s and 1990s [7]. In his work J. Mockus describes a strategy of using powerful Bayesian statistics tools together with Gaussian processes (sec. 2.3) to minimise the amount of evaluation points required to locate the global maximum (or minimum) of the unknown objective function and thus solve optimisation problems like the one we formulated at the start of this chapter.

We will briefly introduce the most important properties of multivariate Gaussian distribution in sections 2.1, 2.4 and define Gaussian processes together with Gaussian processes regression in sections 2.3 and 2.5. We will then analyse Bayesian optimisation method in-depth in section 3. In addition to that, we will apply Bayesian optimisation for one and two-dimensional global optimisation problems and compare its performance with gradient-based optimisation tool *optim* (used in programming language R) in section 4. Finally, the paper will conclude with a discussion of possible limitations and further extensions of Bayesian optimisation method in section 5.

2 Gaussian Processes

2.1 Gaussian basics

A Gaussian random variable is defined as $X \sim \mathcal{N}(\mu, \sigma^2)$, where μ is the mean and σ^2 is the variance. If $X \sim \mathcal{N}(0, 1)$, we say X follows a standard normal distribution. In general, Gaussian variables have one of the most well known probability densities defined as

$$p(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{(x-\mu)^2}{2\sigma^2}\right\} \tag{2.1}$$

and visualised in Figure 1. Gaussian variables also posses two important properties:

1) The sum of Gaussian variables $x_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$ is also Gaussian: [2]

$$\sum_{i=1}^{n} x_i \sim \mathcal{N}(\sum_{i=1}^{n} \mu_i, \sum_{i=1}^{n} \sigma_i^2). \tag{2.2}$$

2) Scaling a Gaussian $x_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$ by some constant c leads to a Gaussian: [2]

$$cx_i \sim \mathcal{N}(c\mu_i, c^2\sigma_i^2). \tag{2.3}$$

In addition to that, one can define a D dimensional *multivariate* Gaussian as [4]

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) \triangleq \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right\}, \tag{2.4}$$

where $\boldsymbol{\mu}$ is the mean vector and $\boldsymbol{\Sigma}$ is the $D \times D$ covariance matrix. The inverse of covariance matrix $\boldsymbol{\Sigma}^{-1}$ is also known as *precision* or *concentration matrix* $\boldsymbol{\Lambda} = \boldsymbol{\Sigma}^{-1}$.

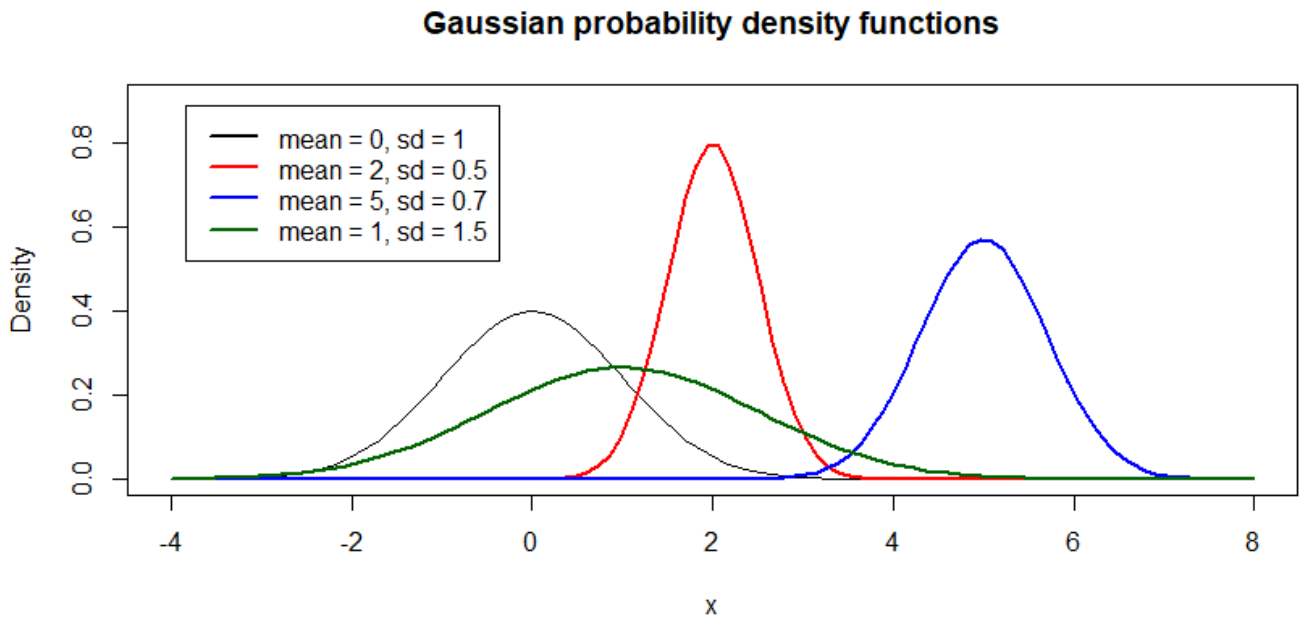

Figure 1: Comparison of Gaussian probability density functions with different means and standard deviations (sd). The mean clearly controls the location of density's peak value on x axis and the standard deviation determines the width of the density function.

2.2 Gaussian distribution in Bayesian linear regression

Consider simple linear regression model, where the output is a linear combination of the inputs:

$$f(\mathbf{x}) = \mathbf{x}^T\boldsymbol{\beta}, \qquad y = f(\mathbf{x}) + \epsilon.$$

Here $\mathbf{x}$ is the input vector, $\boldsymbol{\beta}$ is a vector of parameters of the linear model, f is the function value, ϵ is the additive noise and y is the observed response value. In the Bayesian point of view, linear regression is formulated using probability distributions rather than point estimates (like it is done in the frequentist approach). Thus the aim of Bayesian linear regression is to determine the distribution for the model parameters (also known as *posterior*). Consequently, not only is the response generated

from a probability distribution, but the model parameters are assumed to come from a distribution as well. Hence for our linear model, let us assume that the additive noise ϵ follows an independent, identically distributed Gaussian distribution with zero mean and variance σ_n^2: [1]

$$\epsilon \sim \mathcal{N}(0, \sigma_n^2).$$

This implies that the *likelihood*, or the probability density of the observations given the parameters also follow an independent Gaussian distribution defined as [1]

$$p(\boldsymbol{y}|X, \boldsymbol{\beta}) \;=\; \prod_{i=1}^{n} p(y_i|\boldsymbol{x}_i, \boldsymbol{\beta}) \;=\; \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_n} \exp\left\{-\frac{(y_i - \boldsymbol{x}_i^T\boldsymbol{\beta})^2}{2\sigma_n^2}\right\} \;=\; \mathcal{N}(X^T\boldsymbol{\beta},\, \sigma_n^2 I), \tag{2.5}$$

where I is an n dimensional identity matrix. Now if we freely specify our initial beliefs about the parameters (in other words, define a *prior* of our choice) and denote them as $p(\boldsymbol{\beta})$, we can use Bayes' rule to compute posterior distribution of the model parameters: [1]

$$\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{marginal likelihood}} \quad \Rightarrow p(\boldsymbol{\beta}|\boldsymbol{y}, X) = \frac{p(\boldsymbol{y}|X, \boldsymbol{\beta})p(\boldsymbol{\beta})}{p(\boldsymbol{y}|X)}, \tag{2.6}$$

where $p(\boldsymbol{y}|X)$ can also be seen as normalising constant since it is simply given by [1]

$$p(\boldsymbol{y}|X) = \int p(\boldsymbol{y}|X, \boldsymbol{\beta})p(\boldsymbol{\beta})d\boldsymbol{\beta}.$$

One can see a possible graphical representation of Bayes' rule (eq. 2.6) in the figure below.

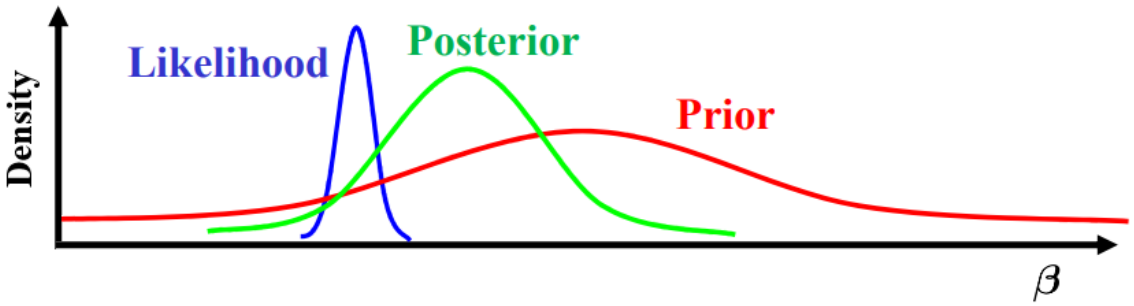

Figure 2: An example of Bayes' rule: having calculated the probability density of the observations given the parameters (in blue) and having defined prior distribution over parameter $\boldsymbol{\beta}$ (in red), one can derive posterior distribution (in green) using Bayes' rule (see eq. 2.6). [5]

2.3 Gaussian Processes in non parametric regression

An alternative way of doing Bayesian regression is possible by considering inference directly in function space rather than considering it for some specific parameters β. In order to define a distribution over functions, we introduce *Gaussian process* - a collection of random variables, any finite number of which have a joint multivariate Gaussian (see eq. 2.4) distribution [1]. A Gaussian process (later GP) defines a prior over functions, which can be converted into a posterior over functions once we have seen some data. This process is completely specified by its mean function and covariance function. Let

$$f(\boldsymbol{x}) \;\sim\; \mathcal{GP}(\mu(\mathbf{x}),\, k(\mathbf{x}, \mathbf{x}')) \tag{2.7}$$

be the prior distribution, where $\mu(\mathbf{x})$ is the mean function and $k(\mathbf{x}, \mathbf{x}')$ is the covariance function (also known as kernel), i.e., [4]

$$\mu(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})], \tag{2.8}$$

$$k(\mathbf{x}, \mathbf{x}') = \mathbb{E}[(f(\mathbf{x}) - \mu(\mathbf{x}))(f(\mathbf{x}') - \mu(\mathbf{x}'))^T]. \tag{2.9}$$

Then for finite collection of points $x_1, ..., x_n \in \mathcal{S}$ (some arbitrary set $\mathcal{S}$, i.e. $\mathbb{R}$), the associated finite set of random variables $f(x_1), ..., f(x_n)$ are distributed as

$$\begin{bmatrix} f(x_1) \\ \vdots \\ f(x_n) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu(x_1) \\ \vdots \\ \mu(x_n) \end{bmatrix}, \begin{bmatrix} k(x_1, x_1) & \dots & k(x_1, x_n) \\ \vdots & \ddots & \vdots \\ k(x_n, x_1) & \dots & k(x_k, x_n) \end{bmatrix} \right), \tag{2.10}$$

where $\mu(\cdot)$ and $k(\cdot, \cdot)$ are mean and covariance functions respectively. Depending on the problem or question of interest, one can specify these functions differently. Perhaps the most common choice for the mean function is simply a constant value c, $\mu(x_i) = c$ (frequently also chosen to be equal to zero). On the other hand, there are many different popular choices for the covariance function k, or simply a *kernel*. Kernel defines a measure of similarity between different pairs of the output $f(x_i)$ and $f(x_j)$, where $i, j \in [1, n]$. In other words, kernel encapsulates covariance between the outputs $f(x_1), ..., f(x_k)$ as a function of the different pairs of inputs $x_1, ..., x_k$. It is often the case that the points closer in the input space are more strongly correlated than the points further away from each other. For instance, $f(x_2)$ is usually defined to be much more correlated with the point $f(x_1)$ or $f(x_3)$ than with the point $f(x_7)$. To illustrate this, let us consider one of the simplest and most commonly used kernels called *exponential* or *RBF* kernel: [2]

$$k(x_i, x_j) = \alpha \exp\left\{ -\frac{||x_i - x_j||^2}{2l^2} \right\}, \tag{2.11}$$

where α and l are arbitrary kernel parameters. In the simplest case where $\alpha = l = 1$, we can visualise the shape of the kernel by plotting $k(\mathbf{x}, \mathbf{0})$ over freely chosen sample space $\mathbf{x}$ (see the plot on the left in Figure 3). Alternatively, we can construct full covariance matrix $k(\mathbf{x}, \mathbf{x}')$ defined in equation 2.10 by evaluating kernel function 2.11 at each pair of points x_i, x_j in our sample space (see the plot on the right in Figure 3).

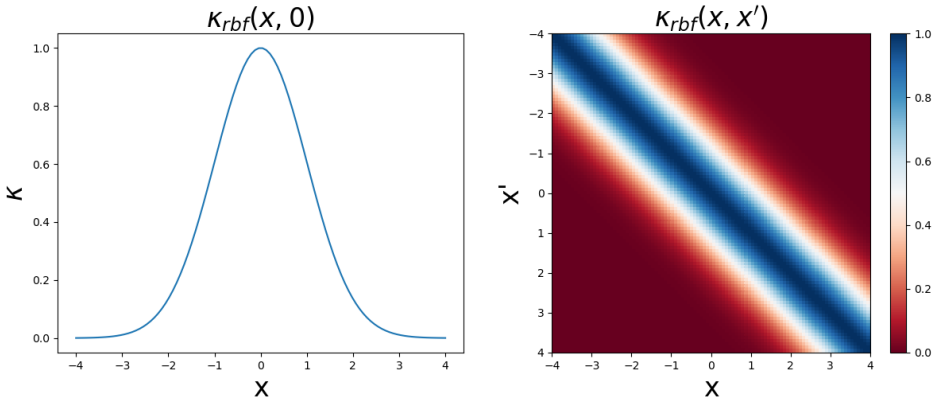

Figure 3: Plot on the left visualise the shape of RBF kernel $k(\mathbf{x}, \mathbf{0})$ with parameter values $\alpha = l = 1$. This kernel describes the covariance between each sample $\mathbf{x}$ location and 0. It also clearly resembles Gaussian distribution shape presented earlier in Figure 1. Using the same kernel parameter values, plot on the right illustrates covariance matrix $k(\mathbf{x}, \mathbf{x}')$. The elements of this matrix represent the covariance between respective points in $\mathbf{x}$ and $\mathbf{x}'$ and it clearly indicates that the highest covariance values take at the points close to each other. [15]

In a similar manner, we can show how the RBF kernel parameters α and l control the shape of the covariance function, which affects the value of the covariance between points $\mathbf{x}$ and $\mathbf{x}'$ (see the Figures 4 and 5 below).

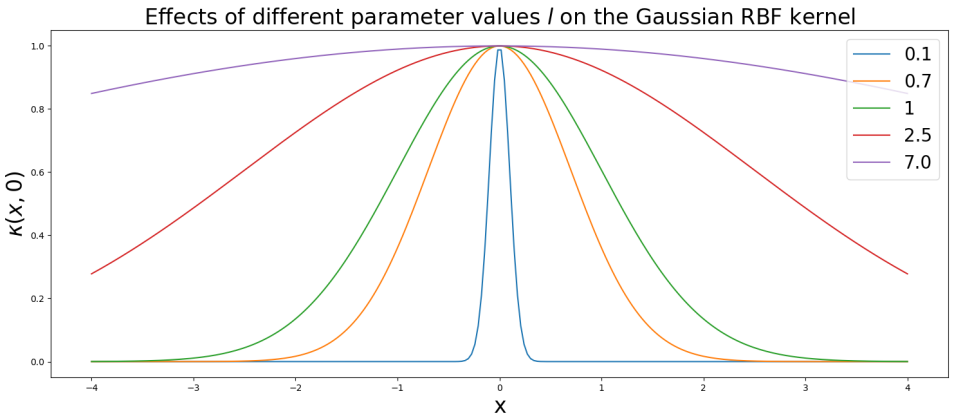

Figure 4: The shape of RBF kernel $k(\mathbf{x}, \mathbf{0})$ with parameter values $\alpha = 1$ and different values of l. Parameter l clearly controls the width of the kernel. [15]

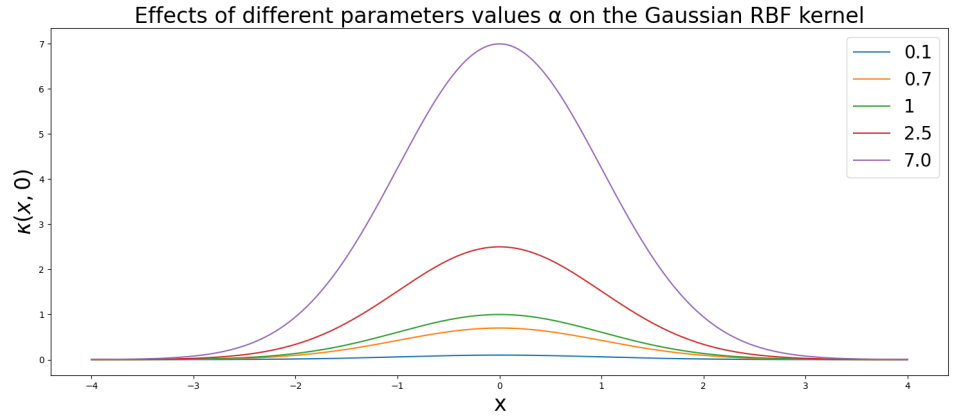

Figure 5: The shape of RBF kernel $k(\mathbf{x}, \mathbf{0})$ with different parameter values of α and the same parameter value $l = 1$. Parameter α clearly controls the height of the kernel. [15]

Now that we have discussed what form the mean $\mu(\mathbf{x})$ and covariance $k(\mathbf{x}, \mathbf{x}')$ functions can take (please note that we have only discussed one of many different types of kernels, detailed analysis of other possible choices can be found on chapter 4 in [1]), we may draw samples from GP prior distribution introduced in equation 2.7. For instance, consider a sample space $\mathbf{x}$ consisting of 100 equally spaced points $x_1, ..., x_{100} \in [-5, 5]$. Now for simplicity, let us choose our mean function to be constant, i.e. $\mu(\mathbf{x}) = \mathbf{0}$ and assume we use RBF kernel (with parameters $l = \alpha = 1$) analysed above (eq. 2.11) to construct our covariance matrix for every pair of $x_i, x_j \in \mathbf{x}$. Hence using equation 2.10 we can calculate every $f(x_1), ..., f(x_{100})$ and draw multiple samples from a GP prior. An illustration of 10 randomly drawn samples is shown in Figure 6.

2.4 Multivariate Gaussian property

[1]Recall the definition of multivariate Gaussian defined in subsection 2.1 (equation 2.4). Now suppose $\mathbf{x} = (\mathbf{x_1}, \mathbf{x_2})$ is jointly Gaussian with parameters

[1]Details about this property are taken from [4]

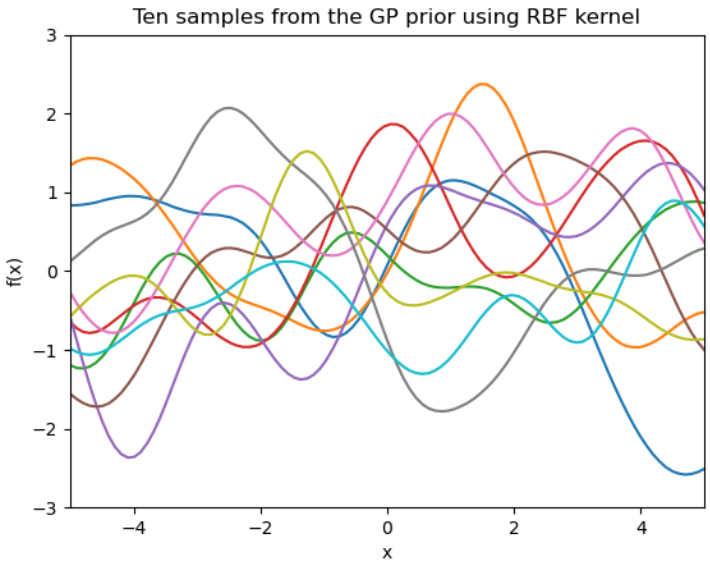

Figure 6: 10 samples drawn from GP prior using RBF kernel with parameters $l = \alpha = 1$.

$$\boldsymbol{\mu} = \begin{pmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{pmatrix}, \quad \boldsymbol{\Sigma} = \begin{pmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{pmatrix}, \quad \boldsymbol{\Lambda} = \boldsymbol{\Sigma}^{-1} = \begin{pmatrix} \boldsymbol{\Lambda}_{11} & \boldsymbol{\Lambda}_{12} \\ \boldsymbol{\Lambda}_{21} & \boldsymbol{\Lambda}_{22} \end{pmatrix}.$$

Then the marginal distributions are given by

$$\begin{aligned} p(\mathbf{x}_1) &= \mathcal{N}(\mathbf{x}_1 | \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_{11}), \\ p(\mathbf{x}_2) &= \mathcal{N}(\mathbf{x}_2 | \boldsymbol{\mu}_2, \boldsymbol{\Sigma}_{22}) \end{aligned}$$

and the posterior conditional is given by[2]

$$\boxed{\begin{aligned} p(\mathbf{x}_1 | \mathbf{x}_2) &= \mathcal{N}(\mathbf{x}_1 | \boldsymbol{\mu}_{1|2}, \boldsymbol{\Sigma}_{1|2}), \\ \boldsymbol{\mu}_{1|2} &= \boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\mathbf{x}_2 - \boldsymbol{\mu}_2), \\ \boldsymbol{\Sigma}_{1|2} &= \boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21} = \boldsymbol{\Lambda}_{11}^{-1}. \end{aligned}} \tag{2.12}$$

One can notice that both the marginal and conditional distributions are themselves Gaussian. This important property together with expressions for conditional mean and variance in boxed equations 2.12 will be very useful for Gaussian Process regression discussed in the next subsection 2.5.

2.5 Regression using Gaussian Processes

Consider a situation where we are only given three inputs x_1, x_2, x_3 (known as *training* points) and three outputs $f(x_1), f(x_2), f(x_3)$ of some function f, which we do not know anything else about. Now given any other input $x_\star$ (known as *test* point), let's try to find a way to measure unevaluated point $f_\star = f(x_\star)$ using Gaussian Process tools we have discussed in previous sections[3].

Assume that our initial observed data $f(x_1), f(x_2), f(x_3)$ come from a Gaussian distribution with (for simplicity) zero mean $\mathbf{f} \sim \mathcal{N}(\boldsymbol{\mu} = \mathbf{0}, \mathbf{K})$, where $\mathbf{K}$ is the covariance matrix explicitly written in equation 2.10 and calculated using RBF kernel (eq. 2.11). Then it is safe to assume that our new

[2]For the proof, please refer to Section 4.3.4 in [4].

[3]Idea of this example is adapted from [5]

point $f_\star$ will come from a Gaussian distribution too, i.e., $f_\star \sim \mathcal{N}(0, K_{\star\star})$, where $K_{\star\star}$ is again the RBF kernel $k(x_\star, x_\star)$ expressed in equation 2.11. Hence, we can write $\mathbf{f}$ and $f_\star$ as a joint Gaussian:

$$\begin{bmatrix} f(x_1) \\ f(x_2) \\ f(x_3) \\ f(x_\star) \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ f_\star \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu} = \mathbf{0} \\ \mu_\star \end{bmatrix}, \begin{bmatrix} K_{11} & K_{12} & K_{13} & K_{1\star} \\ K_{21} & K_{22} & K_{23} & K_{2\star} \\ K_{31} & K_{32} & K_{33} & K_{3\star} \\ K_{\star 1} & K_{\star 2} & K_{\star 3} & K_{\star\star} \end{bmatrix} \right) = \mathcal{N}\left(\begin{bmatrix} \mathbf{0} \\ \mu_\star \end{bmatrix}, \begin{bmatrix} \mathbf{K} & \mathbf{K}_\star \\ \mathbf{K}_\star^T & \mathbf{K}_{\star\star} \end{bmatrix} \right),$$

where $K_{ij} = k(x_i, x_j) = \alpha \exp\left\{ -\frac{||x_i - x_j||^2}{2l^2} \right\}$ for $i, j \in \{1, 2, 3, \star\}$ and $\alpha = l = 1$. Now since we have joint Gaussian, we can simply apply multivariate Gaussian property introduced in section 2.4 (eq. 2.12) to find:

$$p(f_\star | x_1, x_2, x_3, x_\star, \mathbf{f}) = \mathcal{N}(f_\star | \mu_\star, \Sigma_\star), \tag{2.13}$$

$$\mu_\star = \mathbf{0} + \mathbf{K}_\star^T \mathbf{K}^{-1} (\mathbf{f} - \mathbf{0}), \tag{2.14}$$

$$\boldsymbol{\Sigma}_\star = \mathbf{K}_{\star\star} - \mathbf{K}_\star^T \mathbf{K}^{-1} \mathbf{K}_\star. \tag{2.15}$$

Thus, using expressions 2.13–2.15 we can predict mean location of unknown output $f(x_\star)$ within some variance range $\boldsymbol{\Sigma}_\star$. In fact, we can extend this approach for any given amount of new inputs $x_\star$ rather than just one. If we collect our training points into a vector $\mathbf{X}$ and our test points into a vector $\mathbf{X}_\star$, we can define a joint Gaussian as

$$\begin{bmatrix} \mathbf{f} \\ \mathbf{f}_\star \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu} \\ \boldsymbol{\mu}_\star \end{bmatrix}, \begin{bmatrix} \mathbf{K} & \mathbf{K}_\star \\ \mathbf{K}_\star^T & \mathbf{K}_{\star\star} \end{bmatrix} \right) \tag{2.16}$$

and once again use multivariate Gaussian property (eq. 2.12) to obtain

$$p(\mathbf{f}_\star | \mathbf{X}_\star, \mathbf{X}, \mathbf{f}) = \mathcal{N}(\mathbf{f}_\star | \boldsymbol{\mu}_\star, \boldsymbol{\Sigma}_\star), \tag{2.17}$$

$$\boldsymbol{\mu}_\star = \boldsymbol{\mu}(\mathbf{X}_\star) + \mathbf{K}_\star^T \mathbf{K}^{-1} (\mathbf{f} - \boldsymbol{\mu}(\mathbf{X})), \tag{2.18}$$

$$\boldsymbol{\Sigma}_\star = \mathbf{K}_{\star\star} - \mathbf{K}_\star^T \mathbf{K}^{-1} \mathbf{K}_\star, \tag{2.19}$$

where the conditional distribution in equation 2.17 is called the *posterior* probability distribution in the nomenclature of Bayesian statistics. Hence equations 2.17 – 2.19 can be used to perform *Gaussian process regression.*

Let us illustrate GP regression by looking at a quick example. Assume we have collected 7 training data points $\mathcal{D} = \{(x_i, f(x_i)), x_i \in [-5, 5], i = 1, 2, ..., 7\}$ using a function $f(x) = \sin(0.8x)$ and suppose we are again using RBF kernel with parameters $\alpha = l = 1$. Then using only the training points we can predict the value of $f(x)$ at, for instance, 65 new test points, all equally spread throughout domain $[-5, 5]$. If we collect all these test points into a vector $\mathbf{X}_\star$ and denote training points as $\mathbf{X} = x_1, ..., x_7$, we can define a joint Gaussian as in equation 2.16 and use multivariate Gaussian property to calculate mean (eq. 2.18) and variance (eq. 2.19) at all new 65 test points (see the regression results illustrated in Figure 7). In addition to that, similarly as we did in section 2.3, we can sample from GP prior (eq. 2.7) and GP posterior (eq. 2.17). Exemplary samples are plotted in Figure 8.

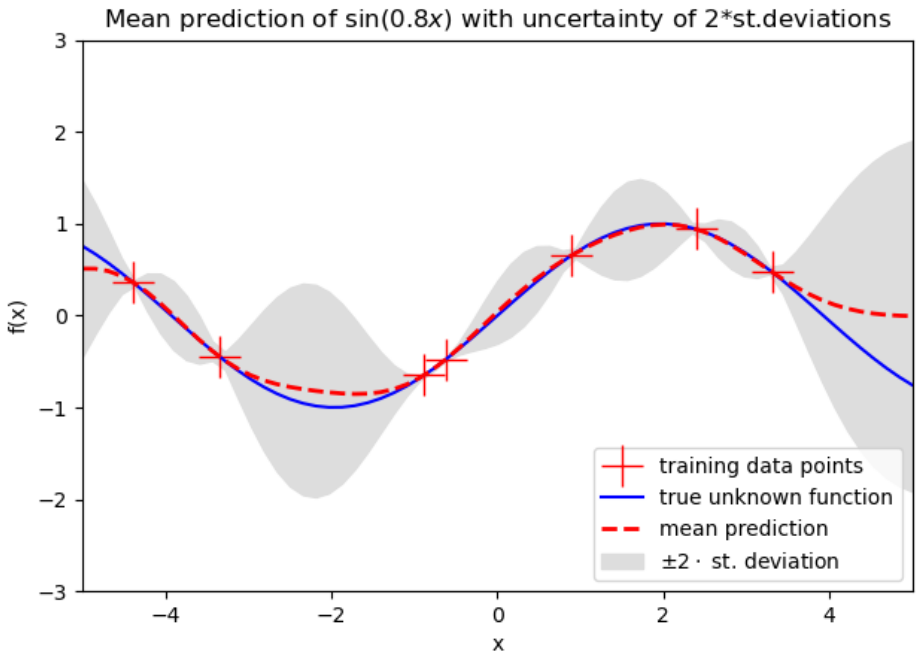

Figure 7: Gaussian Process regression example: approximating function $f(x) = \sin(0.8x)$ using 7 test points. One can notice that unexplored areas (where we do not have training data) have the highest uncertainty (largest grey shaded areas), whereas at the exact locations of training points there is no uncertainty at all. Thus increasing the amount of training points would decrease the uncertainty and improve the mean prediction fit. [16]

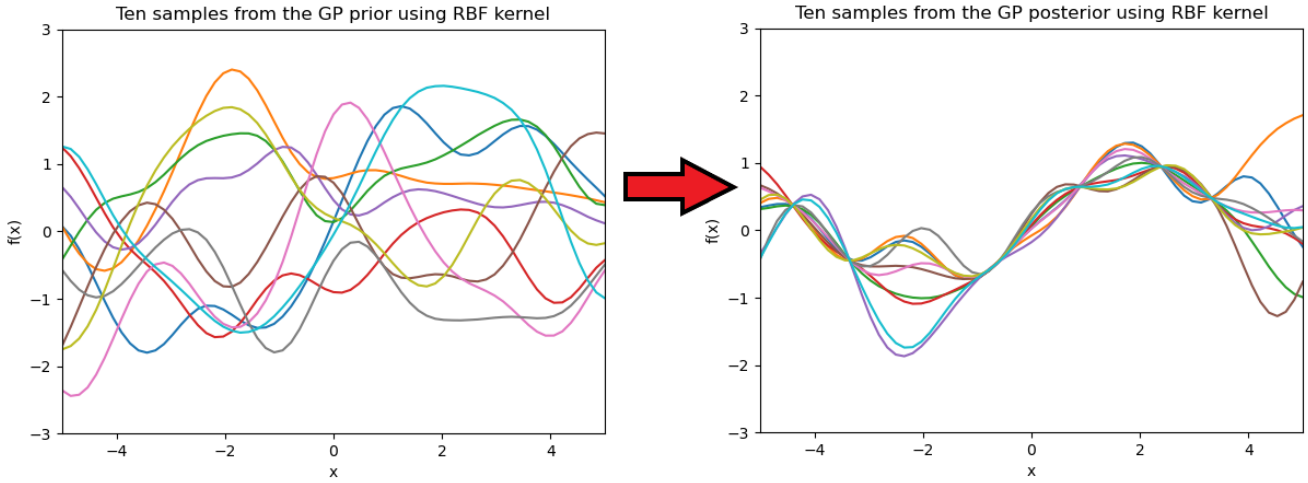

Figure 8: An example of sampling from Gaussian Process prior and posterior distributions using RBF kernel. Samples from the posterior distribution clearly resemble our objective function $f(x) = \sin(0.8x)$. [16]

3 Bayesian Optimisation

3.1 Bayesian Optimisation algorithm

Now that we have introduced and discussed the most important properties of Gaussian processes, we can turn back to our initial problem of global optimisation and try to find a solution to the situation described in section 1 (finding the global maximum of some unknown function f).

We adapt the methodology of performing global optimisation, also known as Bayesian Optimisation, first introduced by J. Mockus [7]. The algorithm follows five main steps: [8]

1. Choosing a Gaussian prior over the space of possible objectives f.
2. Updating prior to a posterior using available data.
3. Using the posterior to decide where to take the next evaluation of f according to some *acquisition function* (for explicit examples see section 3.2).
4. Adding the new evaluation to the existing data.
5. Repeating the steps 2-4 until evaluation budget is exhausted or convergence to a global optimum is reached.

Let us discuss individual steps in a bit more detail. In step 1, we can place a Gaussian process prior (eq. 2.7) on our function of interest f, similarly as we did in section 2.3, by choosing desired mean and covariance functions. Then, we have all the required tools (expressions 2.17-2.19) to calculate posterior distribution using available data and complete step 2, just as we did in the example on section 2.5. The posterior probability distribution provides us with a measure that describes potential values for $f(x)$ at some new candidate point x. The choice for this new x is made in step 3, using the preferred acquisition function. Acquisition functions are designed to represent our beliefs over the global optimum of $f(x)$. They enable us to choose a value of x, at which $f(x)$, according to our current knowledge, is most likely to be maximised (or minimised - it is important to note that this algorithm is originally defined for obtaining the global minimum of unknown functions but can easily be adapted to finding the global maximum instead, simply by minimising the negative of the objective function f). After choosing the new point x, we observe $f(x)$, add this new point to the existing data and repeat the algorithm from step 2. A conceptual illustration of the Bayesian optimisation procedure over three iterations can be seen in Figure 9.

3.2 Most common acquisition functions

The role of the acquisition function is to guide the search for the optimum. Typically, acquisition functions are defined such that high acquisition corresponds to potentially high values of the objective function, whether because the prediction mean is high, the uncertainty is great, or both. Hence maximisation of the acquisition function is used to select the next point at which to evaluate the objective function [10]. One should note that maximising acquisition function is much easier compared to maximising the original unknown objective function, since we know the exact form the acquisition function takes and it is cheap to evaluate.

In this section we will explore the most popular acquisition functions and find out how they affect the optimisation process. Perhaps one of the first acquisition functions ever designed for Bayesian optimisation was *probability of improvement*. It is defined as [9]

$$\mathrm{PI}(\mathbf{x}) = P\left(f(\mathbf{x}) \geq f(x_{\mathrm{best}}) + \xi\right) \tag{3.1}$$

$$= \Phi\left(\frac{\mu(\mathbf{x}) - f(x_{\mathrm{best}}) - \xi}{\sigma(\mathbf{x})}\right), \tag{3.2}$$

where $f(\mathbf{x})$ is a value of objective function which is going to be observed next, $f(x_{best})$ is the highest value observed so far, Φ is cumulative distribution function of the normal distribution, and ξ is an optional tuneable parameter (with default value of 0) that helps to make probability of improvement more flexible. Essentially, this acquisition function estimates the probability that objective function

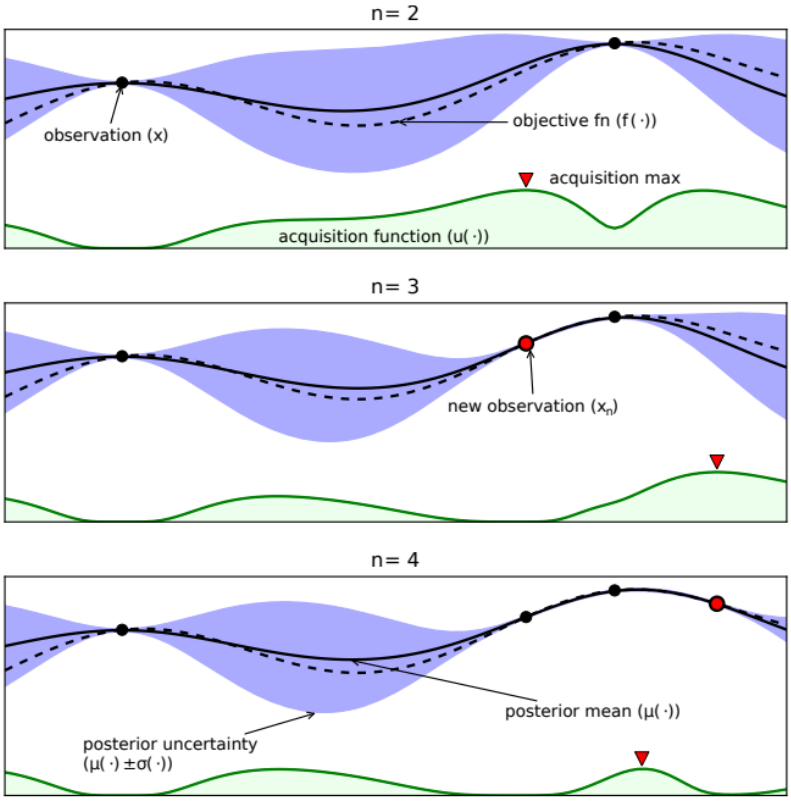

Figure 9: Visualisation of Bayesian optimisation algorithm over three iterations. Starting with only two observed points at iteration $n = 2$, one can see how at each iteration acquisition function suggests (by red triangle marker) the best location for the new observation in order to obtain the maximum of objective function f with the least possible number of iterations. The locations at which we have data have low uncertainty, whereas at the areas where we do not have the data, the uncertainty is high (indicated in light violet areas). Also note that although the objective function is visualised in this figure (in dashed line), in practice it is unknown. [9]

f evaluated at the new point is going to be higher than the best one we have seen so far. Since we are working with Gaussian processes, we know that posterior distribution of $f(\mathbf{x})$ is also Gaussian, hence we can analytically compute this probability (eq. 3.1) using cumulative function of Gaussian distribution (eq. 3.2), with $\mu(\mathbf{x})$ and $\sigma(\mathbf{x})$ being our mean and standard deviation, respectively. Thus when using probability of improvement, the main task is to find an x which maximises Φ, the cumulative of Gaussian. For a practical illustration of probability of improvement refer to Figure 10, which also shows how introducing the optional parameter ξ enables the user to control the minimum amount of desired improvement for each iteration.

Another commonly used acquisition function, which takes into account not only the probability of improvement, but also the magnitude of the improvement a point can potentially yield, is called *expected improvement*. It was first introduced by Mockus [11], who initially came up with an idea to define the improvement function as:

$$\mathrm{I}(\mathbf{x}) = \max\{0, f(\mathbf{x}) - f(x_{best})\}.$$

Clearly, $\mathrm{I}(\mathbf{x})$ is positive when the next evaluation is higher than the best value known thus far and

zero otherwise. Hence, given available data, the new query point is found by maximising the expected improvement: [10]

$$\mathbf{x} = \underset{\mathbf{x}}{\operatorname{argmax}}\ \mathbb{E}\{0, f(\mathbf{x}) - f(x_{best})\}.$$

Similarly as before, since $f(\mathbf{x})$ is normally distributed, the expected improvement function can be derived analytically [9, 11], yielding: [8]

$$\begin{cases} (\mu(\mathbf{x}) - f(x_{best}) - \xi)\Phi(Z) + \sigma(\mathbf{x})\phi(Z), & \text{if } \sigma(\mathbf{x}) > 0, \\ 0, & \text{if } \sigma(\mathbf{x}) = 0. \end{cases} \tag{3.3}$$

$$Z = \frac{\mu(\mathbf{x}) - f(x_{best}) - \xi}{\sigma(\mathbf{x})}, \tag{3.4}$$

where ϕ and Φ denote probability density function and cumulative distribution function of the standard normal distribution respectively. Once again, comparison of expected improvement acquisition functions with different values of optional control parameter ξ can be seen in Figure 10.

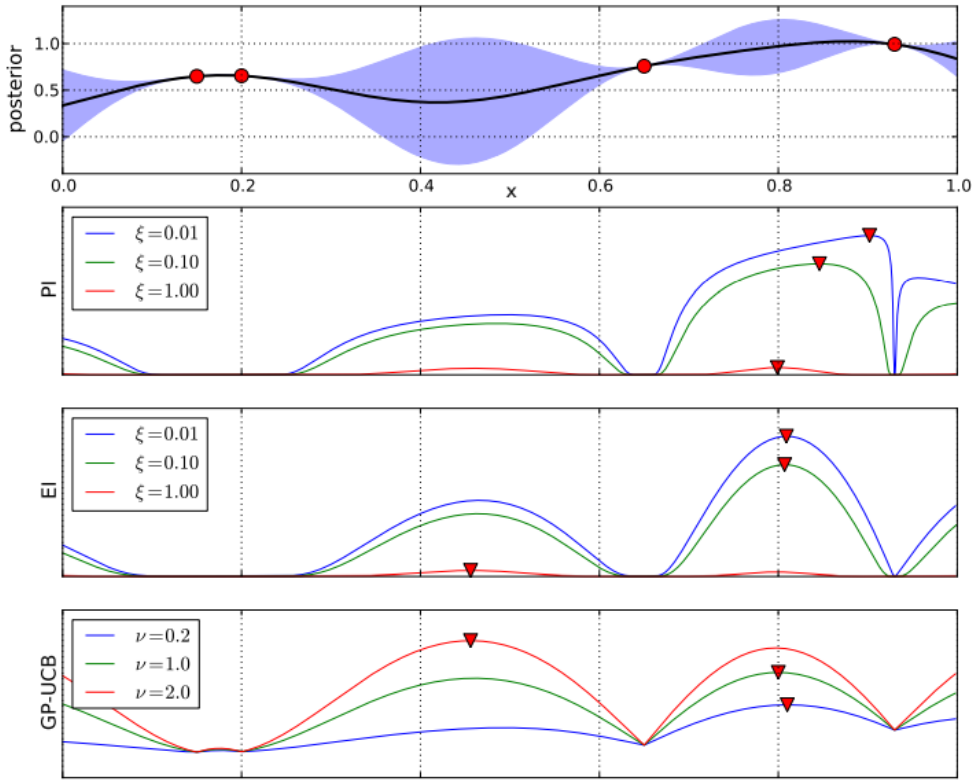

Figure 10: Examples of acquisition functions and their settings. The GP posterior with already 4 observed values (in red circles) is shown at the top panel. The other images show the acquisition functions for that GP. From the top: probability of improvement (eq. 3.1), expected improvement (eq. 3.3) and upper confidence bound (eq. 3.5). The maximum of each function is shown with a triangle marker. The plots indicate that higher values of control parameters ξ, ν tend to force exploration, meaning that acquisition functions suggest the next evaluation locations corresponding to higher uncertainty. [10]

One of the more recently developed (in 2010 [12]) acquisition functions for Bayesian optimisation using Gaussian processes, known as *upper confidence bound* selection criterion, is defined as: [8]

$$\text{GP-UCB}(\mathbf{x}) = \mu(\mathbf{x}) + \nu\sigma(\mathbf{x}). \tag{3.5}$$

Interestingly, the parameter ν allows us to control the balance between the importance we give to the mean $\mu(\mathbf{x})$ and the variance $\sigma(\mathbf{x})$ of the model. Hence, we can define exploration vs exploitation trade-off. In other words, increasing ν makes the acquisition function favour points with high variance, causing exploration of uncertain areas, whereas decreasing ν will make the acquisition function favour points with high mean, causing exploitation of the local areas after *only little* exploration was done instead. Hence, depending on the problem, the question of interest, and the choice of kernel function used for GP regression, users can freely specify appropriate value of ν in order to achieve their preferred balance between exploration and exploitation. A graphical representation of an upper confidence bound acquisition function as well as its comparison with probability of improvement and expected improvement is illustrated on Figure 10.

There exist many other acquisition functions which use different strategies for efficiently locating global maximum of unknown function f (some of which are even capable to take into account the presence of noise in the observations [3]). With that many different parameterised acquisition functions available, one might often find unclear which one to use, hence sometimes the best one is chosen simply by trial and error. Alternatively, one could implement an interesting acquisition function selection method presented by E. Brochu [13], who instead of using a single acquisition function, adopts a range of acquisition functions governed by so-called "online multi-armed bandit[4]" strategy, which almost always outperforms the best individual acquisition function of a suite of standard test problems.

4 One and two-dimensional global optimisation problems

In previous sections we studied in detail all the mathematical and statistical tools required to perform Bayesian optimisation and overcome our initial challenge of finding global maximum of an unknown function. In this chapter we will define one and two-dimensional functions (which in practice would obviously be unknown) and use them for global optimisation to demonstrate explicit examples of Bayesian optimisation applications. The GPyOpt [18] package, designed to perform Bayesian optimisation using Gaussian Processes in Python, will be used to calculate and illustrate our results.

4.1 Bayesian Optimisation for one-dimensional function

Let us define a one-dimensional function of interest f as

$$f(x) = x^2 \sin^6(5\pi x). \tag{4.1}$$

Assume that we are only interested in the region where $x \in [0, 1.6]$ and that we want to find global maximum within these bounds. Looking at the Figure 11, one can immediately notice that this function has many *local* maxima (at least 7 visible peaks), but only one distinct *global* maximum. Let us further assume that initially, all we know about this function f is a single data point[5], $f(0) = 0$. Before we can perform Bayesian optimisation algorithm defined in section 3.1, we need to set our evaluation budget and choose an acquisition function. As these choices are arbitrary, we will start by using probability of improvement (eq. 3.1) as our acquisition function and limit our budget to 35 iterations or stop the optimisation process if the distance between the last two observations is $< 10^{-6}$. Since the GPyOpt package by default is designed to find a global minimum of an objective function, we will simply minimise the negative of f (eq. 4.1) in order to find its global maximum instead (which does not affect our algorithm process). Running a **.run_optimization** [21] function (which is built-in within the GPyOpt package) with the settings (region of interest, evaluation budget,

[4]the details about this strategy go beyond this paper, but an interested reader is referred to the original paper [13]

[5]which in real life applications would most often be the case. In theory, this known data point could be any point within the region of interest, but once again, to be more realistic, we choose the point at the origin

acquisition function) defined above will perform the Bayesian optimisation. Once it is complete, we can extract the maximum obtained value, as displayed in Figure 12. One can check, by looking back at Figure 11, to confirm that indeed the global maximum was found, since it is located at $x = 1.5$ and has a value of $f(1.5) = 2.25$ (to the precision of two decimal places). More detailed results of the optimisation process can be seen in Figures 13 & 14.

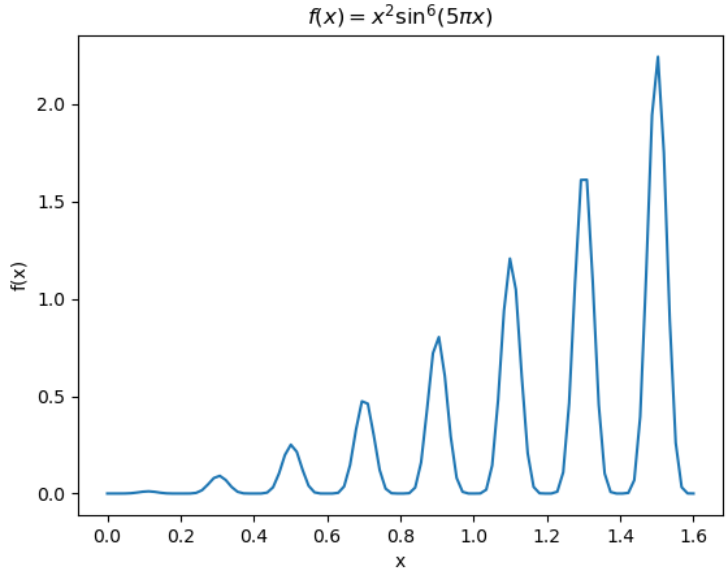

Figure 11: An illustration of one-dimensional objective function $f(x)$ (eq.4.1).

```
============================================================
Value of x that maximize the objective:[1.50149899]
Maximum value of the objective function:2.2507521830879265
============================================================
```

Figure 12: Location of the global maximum of $f(x)$ obtained using Bayesian optimisation with probability of improvement as acquisition function.

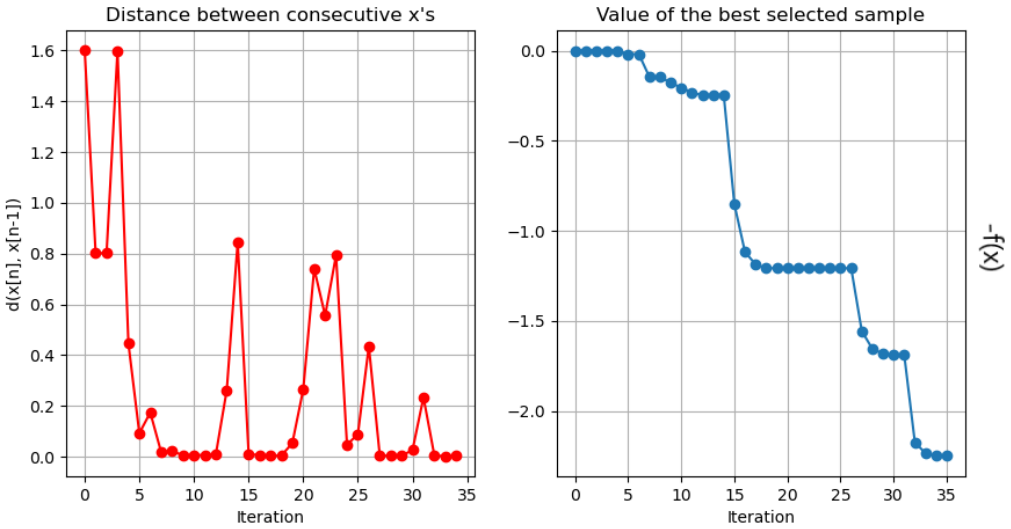

Figure 13: Convergence of the Bayesian optimisation algorithm using probability of improvement: on the left - the distance between the last two evaluation locations, on the right - the largest negative value of $-f(x)$ obtained at each iteration.

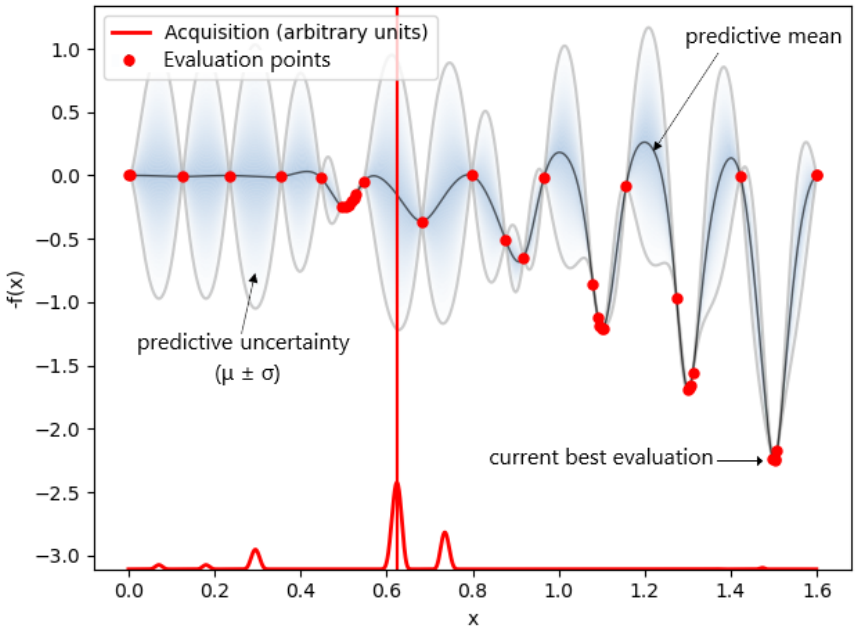

Figure 14: Bayesian optimisation results using probability of improvement with 35 iteration budget. This plot represents the current state of the algorithm after 35 iterations. Acquisition function (probability of improvement) located at the bottom of the graph indicate the next evaluation location (vertical red line) if we were to perform 36th iteration.

The plot on the right in the Figure 13 above clearly shows that almost the whole evaluation budget was exhausted before we obtained the global maximum (see the Figure 34 a) in the appendix for precise evaluations at each iteration). Hence using probability of improvement as acquisition function for this particular one-dimensional optimisation problem might not have been the most efficient choice.

Let us perform Bayesian optimisation using the same settings again, but instead of using probability of improvement, we will use expected improvement (eq. 3.3) as our choice of acquisition function and compare both results. Similarly as before, running the optimisation using *.run_optimization* function within the GPyOpt package yields almost identical (with a slight difference of few decimal places) global maximum value of $f(1.5) = 2.5$ (see Figure 15). However, the algorithm process (illustrated in Figure 16) and its convergence (Figure 17) are much different from our initial results. In contrast, by comparing Figures 16 and 14, one can notice that the areas of high uncertainty (where mean $\pm$ standard deviation is large) using expected improvement were explored more aggressively than using probability of improvement. This is also evident in the left plot of Figure 17, where the distance between consecutive x's using expected improvement has larger variation than the same distance using probability of improvement (visualised on the left plot in Figure 13). In addition to that, the plot on the right in Figure 17 indicates that the global maximum was found just after 9 iterations (see the Figure 34 b) for precise evaluations at each iteration), whereas using probability of improvement over 30 iterations was needed. We may convince ourselves that this behaviour is expected if we go back and look at the way these two acquisition functions are defined. One may notice, that using probability of improvement (eq. 3.2), the decision of the next best evaluation location at each iteration is solely based on the cumulative distribution function of the normal distribution (denoted as Φ). Since at the start of the optimisation the region of interest is yet to be explored and $\sigma(\mathbf{x})$ is large, the value of $\Phi\left(\frac{\mu(\mathbf{x})-f(x_{best})}{\sigma(\mathbf{x})}\right)$ is small, but with every iteration the uncertainty reduces and $\sigma(\mathbf{x})$ decreases, hence giving larger values of Φ. Consequently, when using probability of improvement, more iterations are spent to explore local areas before moving to the uncertain ones. Conversely, the expect improvement function (eq. 3.3) does not solely depend on cumulative distribution function

Φ and instead is defined to have two main terms - one controlling the importance of the mean and one controlling the variance at each iteration, this way providing a more balanced trade-off between exploration and exploitation. Thus, it seems that using the expected improvement over probability of improvement for this one-dimensional optimisation problem (where unknown function, besides one global maximum, has many local peaks) would be much more efficient choice. Alternatively, one could possibly increase the efficiency of the probability of improvement function by trying to implement different values of the optional parameter ξ (refer to eq. 3.2), which allows to increase the minimum amount of improvement needed for each iteration, hence forcing the optimisation to spend less iterations exploring local areas before moving to the uncertain ones.

```
============================================================
Value of x that maximize the objective:[1.50016856]
Maximum value of the objective function:2.25045838009533
============================================================
```

Figure 15: Location of the global maximum of $f(x)$ obtained using Bayesian optimisation with expected improvement as acquisition function.

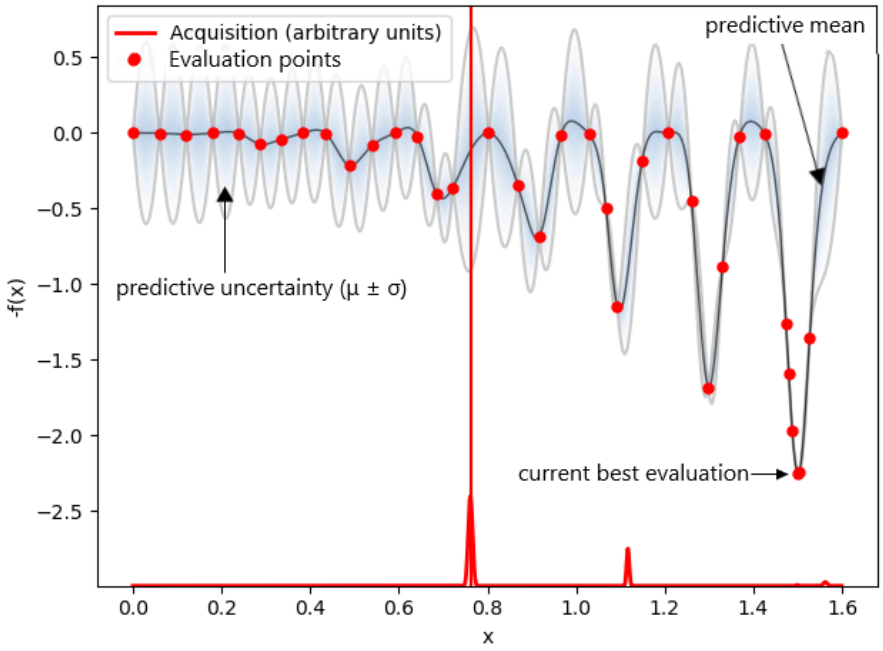

Figure 16: Bayesian optimisation results using expected improvement with 35 iteration budget. This plot represents the current state of the algorithm after 35 iterations. Acquisition function (expected improvement) located at the bottom of the graph indicate the next evaluation location (vertical red line) if we were to perform 36th iteration.

4.1.1 Comparison with the *optim* function in R

In this short subsection, we aim to compare Bayesian optimisation method with a gradient-based general-purpose optimisation tool *optim* [19], which is more commonly used amongst statisticians in programming language R. We continue with the same one-dimensional problem set up, where we aim to find the maximum of $f(x) = x^2 \sin^6(5\pi x)$ (Figure 11) with a 35 evaluation budget within the region of interest $x \in [0, 1.6]$. We initiate the optimisation function **optim()** with a starting

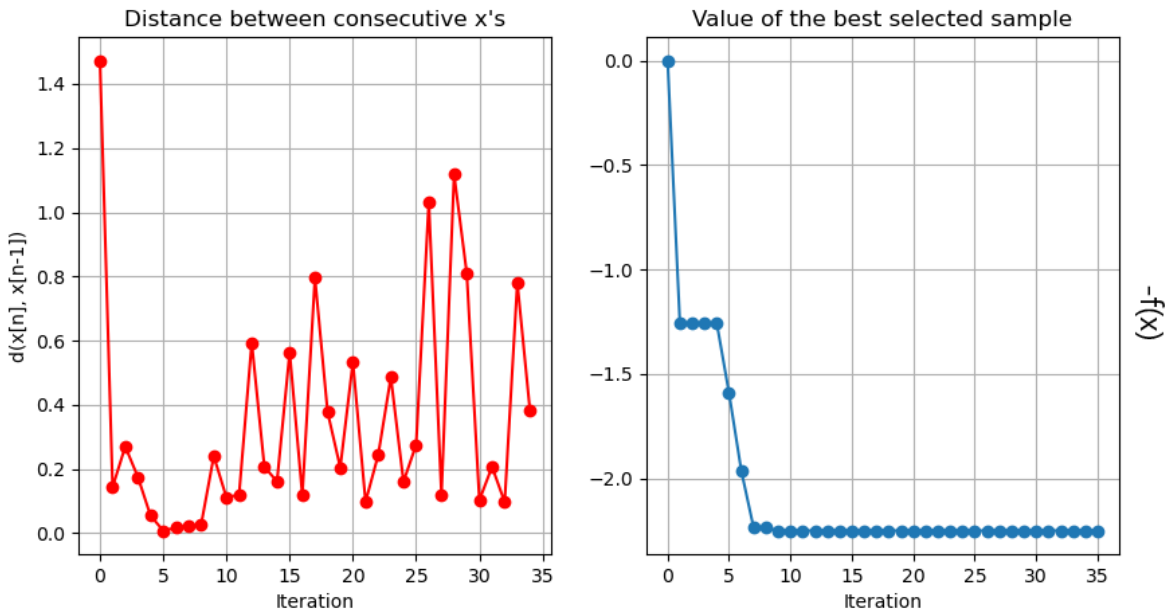

Figure 17: Convergence of the Bayesian optimisation algorithm using expected improvement: on the left - the distance between the last two evaluation locations, on the right - the largest negative value of $-f(x)$ obtained at each iteration.

value $x = 0.1$, specified within the brackets of **optim** as *par = c(0.1)* and use the "L-BFGS-B"[6] method [19] (written as *method = "L-BFGS-B"*), which allows us to choose boundaries $x \in [0, 1.6]$ by specifying values *lower = 0* and *upper = 1.6*. The optimisation process yield these results:

Iteration	Starting Point	1	2	3	4	5	6	7	8	9	10	11
x	0.1	0.10100	0.09900	0.29985	0.30085	0.29885	0.91879	0.91979	0.91779	0.32143	0.32243	0.32043
f(x)	0.01	0.01019	0.00979	0.08991	0.09046	0.08923	0.64753	0.63009	0.66436	0.07306	0.07107	0.07497

Iteration	12	13	14	15	16	17	18	19	20	21	22	23
x	0.74906	0.75006	0.74806	0.88393	0.88493	0.88293	0.91879	0.91979	0.91779	0.32143	0.32243	0.32043
f(x)	0.07650	0.06990	0.08349	0.64408	0.66090	0.62669	0.64753	0.63009	0.66436	0.07306	0.07107	0.07497

Iteration	24	25	26	27	28	29	30	31	32	33	34	35
x	0.74906	0.75006	0.74806	0.88393	0.88493	0.88293	0.90152	0.90252	0.90052	0.90149	0.90249	0.90049
f(x)	0.07650	0.06990	0.08349	0.64408	0.66090	0.62669	0.81135	0.81072	0.81077	0.81135	0.81075	0.81074

Figure 18: one-dimensional function optimisation results using the *optim* function with "L-BFGS-B" method in R. The best obtained value is highlighted in red. [23]

As one may notice, the highest value (highlighted in red) obtained during the optimisation was $f(x = 0.90149) = 0.81135$, which, looking at the Figure 11, is **not** the global maximum. In fact, if we were to increase evaluation budget to any amount of iterations (say 100, for instance), we would find that optimisation process stops halfway through, because it gets "stuck" at one of the local maximums and the distance between two consecutive evaluations becomes negligible. What is more, if we were to use our usual starting value of $x = 0$, we would not obtain any results at all, since the optimisation process stops just after few iterations, unable to find any values beyond $f(x = 0) = 0$ point (see Figure 35 in the appendix for detailed output in this case). One may also experiment with many other different starting values x, but there are very few points from which optimisation converges to the global maximum located at $f(x = 1.5) = 2.25$ - in most cases it converges to one of

[6] *optim* function allows us to choose several different gradient-based methods, but only L-BFGS-B enables us to define the region of interest. For more details, please refer to [19]

the local peaks instead (see appendix for optimisation results with some of the other starting values on Figure 36). Thus, for one-dimensional problems, Bayesian optimisation seems to be superior method, which is not only more efficient, but also much more effective algorithm in dealing with functions which contain many local peaks, but only a single global maximum.

4.2 Bayesian Optimisation for two-dimensional function

In a similar manner to the previous section, let us now consider two-dimensional objective function f defined as

$$f(x_1, x_2) = (x_1^2 + x_2^2)(\sin^2(x_1) - \cos(x_2)). \tag{4.2}$$

As illustrated in Figure 19, this function once again contains many local maxima, but only one global maximum within the region of interest $x_1, x_2 \in [0, 10]$, representing the complexity required to really test the effectiveness of the Bayesian optimisation algorithm. Analogously to section 4.1, we assume that all we know initially about our objective function f is a single data point, $f(x_1 = 0, x_2 = 0) = 0$. Additionally, we set our usual evaluation budget to be 35 iterations or stop the optimisation process if the distance between the last two observations is $< 10^{-6}$. In this section we will tackle this global optimisation problem using two different acquisition functions - expected improvement (eq. 3.3) and upper confidence bound (eq. 3.5), and compare them to each other to see which one performs best. Starting with expected improvement, we may run optimisation with a predefined evaluation budget using GPyOpt package in Python (and recall that, by default settings within the package, in order to obtain global maximum of $f(x_1, x_2)$ we minimise the negative of $f(x_1, x_2)$) to obtain optimisation results displayed in Figure 20. One may solve equation 4.2 analytically to convince oneself that, within given constraints of $x_1, x_2 \in [0, 10]$, the value displayed in Figure 20 is indeed a global maximum (up to an error of ± 0.1). Let us look at the optimisation results in a bit more detail by analysing Figures 21-23.

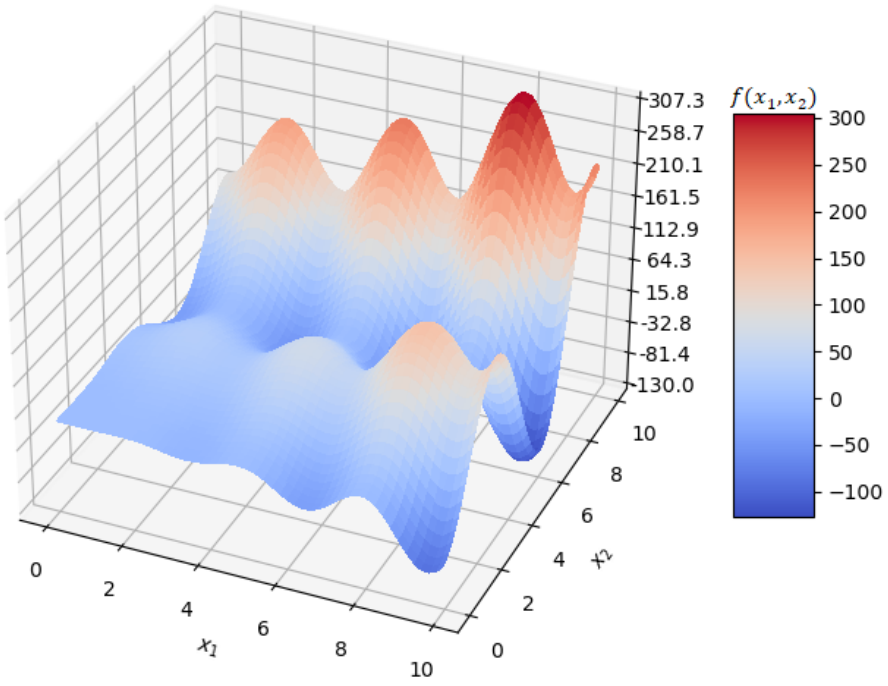

Figure 19: An illustration of two-dimensional objective function $f(x_1, x_2)$ (eq.4.2).

```
==========================================================================
Value of (x₁, x₂) that maximises the objective:[7.95145111 9.64315493]
Maximum value of the objective: 307.242544088916
==========================================================================
```

Figure 20: Location of the global maximum of $f(x_1, x_2)$ obtained using Bayesian optimisation with expected improvement as acquisition function.

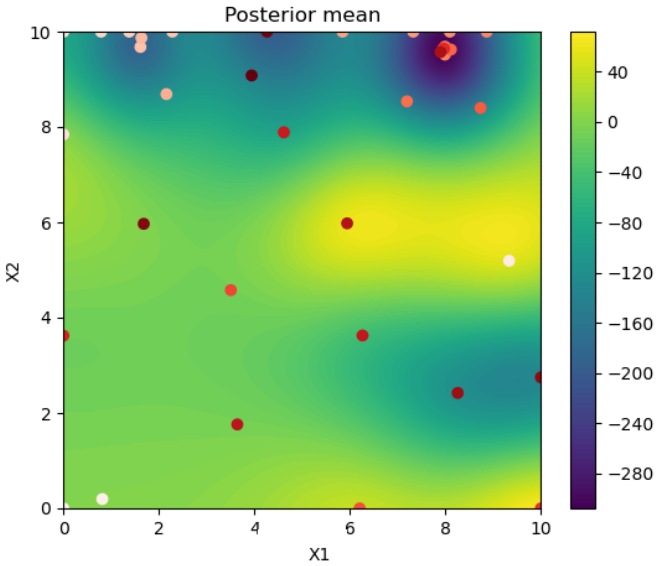

Figure 21: Two-dimensional posterior mean plot obtained after performing Bayesian optimisation using expected improvement and 35 iteration budget. The round dots represent location of each iteration. Note that, since we are minimising the negative of $f(x_1, x_2)$, posterior mean can take negative values in this plot.

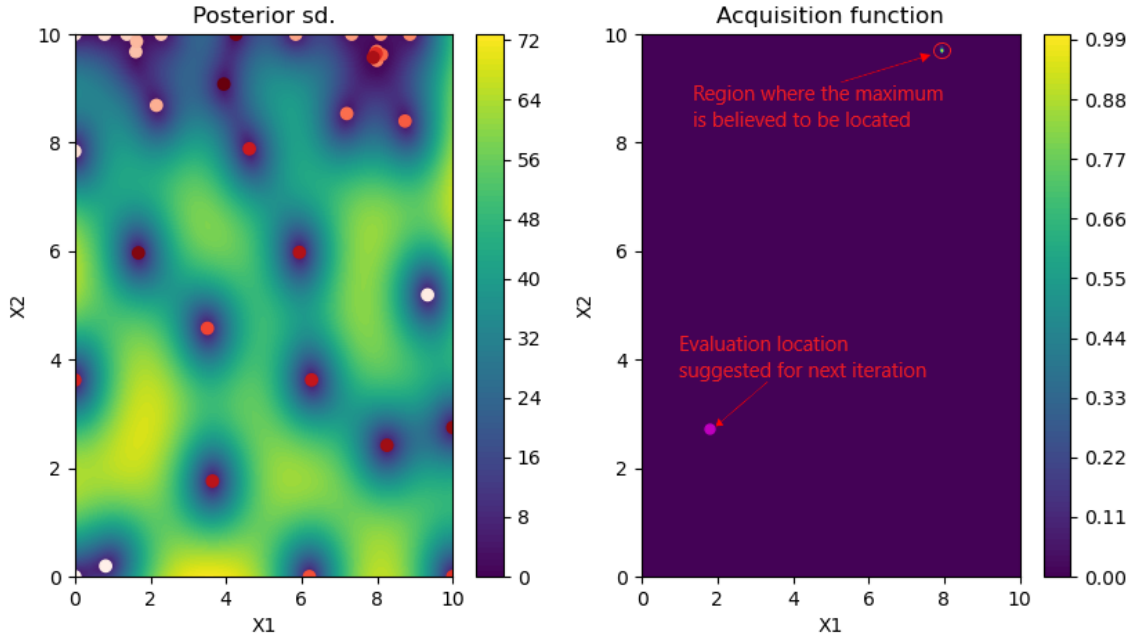

Figure 22: On the left - two-dimensional posterior standard deviation plot (where the round dots represent location of each iteration), on the right - the current state of expected improvement function after 35 iterations (where the pink point represent location of the next evaluation if we were to perform 36th iteration).

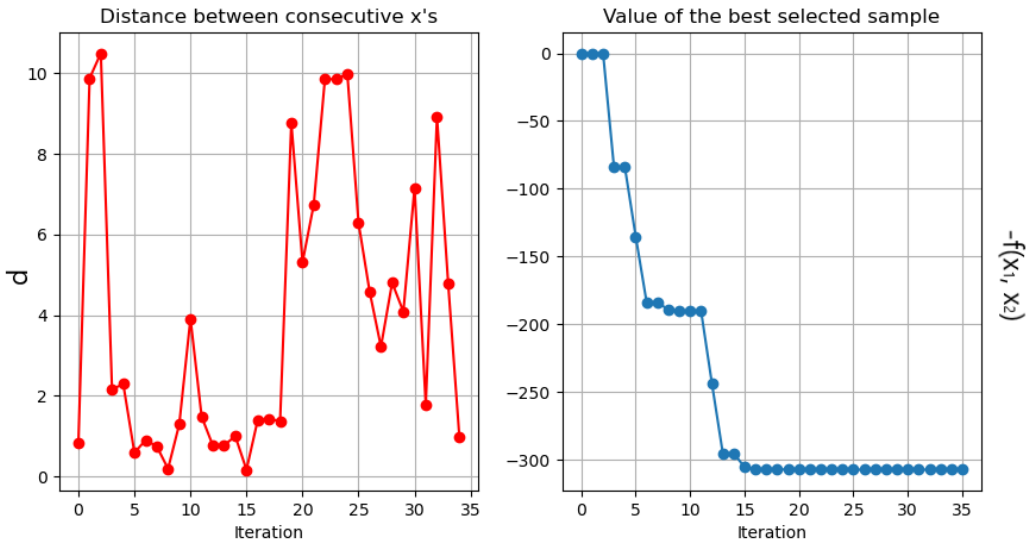

Figure 23: Convergence of the Bayesian optimisation algorithm using expected improvement: on the left - the distance between the last two evaluation locations (where d is calculated as in eq. A.1), on the right - the largest negative value of $-f(x_1, x_2)$ obtained at each iteration.

Similarly as when using expected improvement in one-dimensional problem, both plots of posterior mean (Figure 21) and posterior standard deviation (Figure 22 on the left) indicate that locations of each iteration are fairly spread throughout the whole region of interest, suggesting a healthy balance between exploring areas of high uncertainty (where standard deviation have large values) and exploiting the local areas where global maximum is believed to be most likely located (left and right top corners of the plots). In addition to that, the plot on the left in Figure 23 provide further evidence that during optimisation there was a balanced trade-off between exploitation (where the distance between the last two evaluation locations is very small, i.e. throughout iterations 16 to 18) and exploration (where the distance between the last two evaluation locations is large, i.e. throughout iterations 29 to 32). One should also notice the intelligence acquisition function (Figure 22 on the right) possesses just after the end of the optimisation process: since the majority of the region of interest ($x_1, x_2 \in [0, 10]$) was sufficiently explored and the evaluations at the top right corner of that region had much larger negative values of $-f(x_1, x_2)$, the expected improvement acquisition function tells us that we may already disregard most of the region, as there is certainty that the global maximum (or largest negative value of $-f(x_1, x_2)$) is located at the top right corner - that is why we observe the dominant dark violet colour in the acquisition function plot. What is more, looking at the right graph of Figure 23 we notice that global maximum was found just after 15 iterations, suggesting that when using expected improvement as the acquisition function for this two-dimensional problem we were not only able to find global maximum, but also do it quite efficiently.

Now let us compare how optimisation process differ if this time we are to use an upper confidence bound acquisition function (eq. 3.5) instead. To begin with, we need to choose the value of parameter ν which will be used for upper confidence bound function to determine the balance between exploration and exploitation. We will start with a value of $\nu = 2$ since that is the default value within the GPyOpt package in Python. If we run the optimisation (keeping in mind the same settings as before - a 35 iteration budget, $x_1, x_2 \in [0, 10]$ region of interest and starting point of $f(0, 0) = 0$) we obtain initial output displayed in Figure 24. Clearly, the obtained maximum value $f(1.6, 9.82) = 190.27$ is **not** the global maximum. Examining the posterior mean (in Figure 25), we can immediately notice that majority of evaluations of each iteration were clustered at the top left corner of the domain, suggesting a poor balance between exploration and exploitation (with the latter being too significant). This is also well reflected in posterior standard deviation plot (Figure

```
========================================================================
Value of (x,y) that maximises the objective:[1.60197623 9.81645494]
Maximum value of the objective: 190.27020152469103
========================================================================
```

Figure 24: Location of the largest value of $f(x_1, x_2)$ obtained using Bayesian optimisation with upper confidence bound as acquisition function with default parameter value $\nu = 2$.

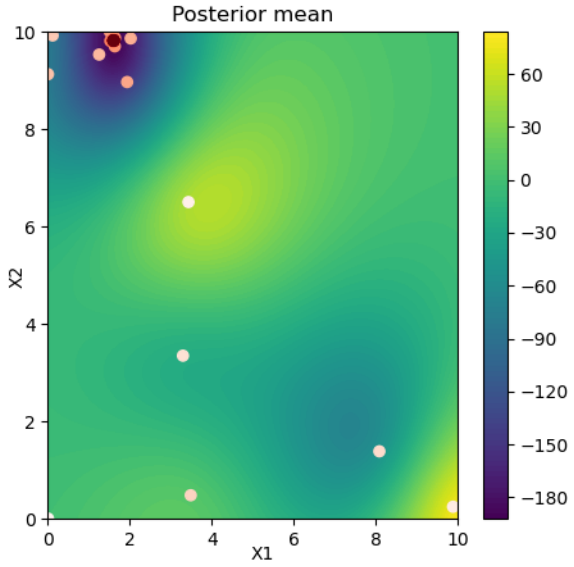

Figure 25: Two-dimensional posterior mean plot obtained after performing Bayesian optimisation using upper confidence bound with default parameter value $\nu = 2$ and 35 iteration budget. The round dots represent location of each iteration.

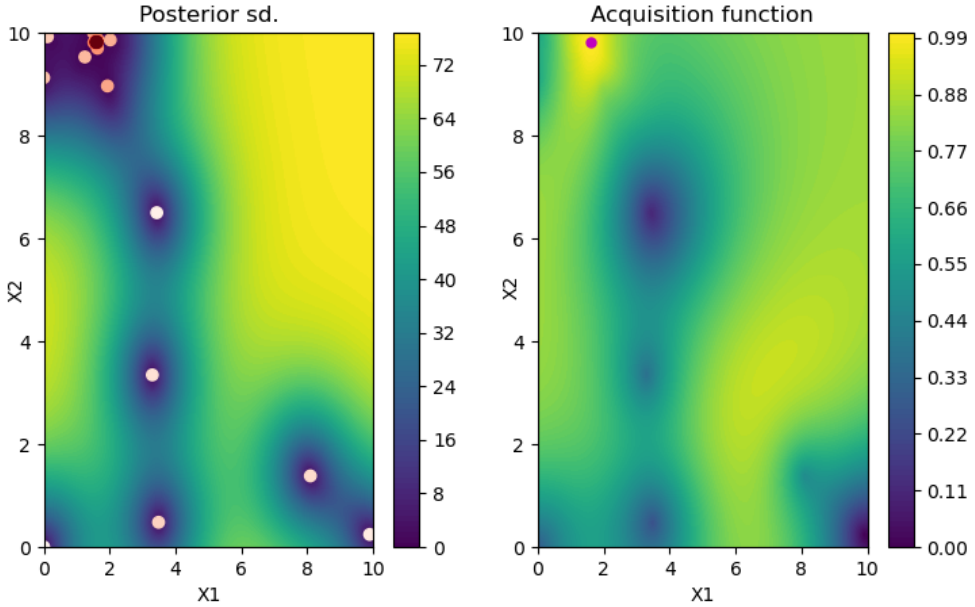

Figure 26: On the left - two-dimensional posterior standard deviation plot (where the round dots represent location of each iteration), on the right - the current state of upper confidence bound function (with default value of parameter $\nu = 2$) after 35 iterations (where the pink point represent location of the next evaluation if we were to perform 36th iteration).

26 on the left), which indicates a high uncertainty for the top right corner of the domain. Not only that, the dominant colours of higher end of the colour bar scale (yellow-green) across entire domain in the upper confidence bound function (Figure 26 on the right) indicate that there is a lack of certainty as to where exactly the global maximum could potentially be located, even after 35 iterations of the optimisation. Lastly, from convergence graphs on Figure 27 we can deduce that there was little to no exploration done after 13th iteration, as the distance between two evaluation locations beyond that iteration is negligible and there is practically no improvement in the search of the optimum.

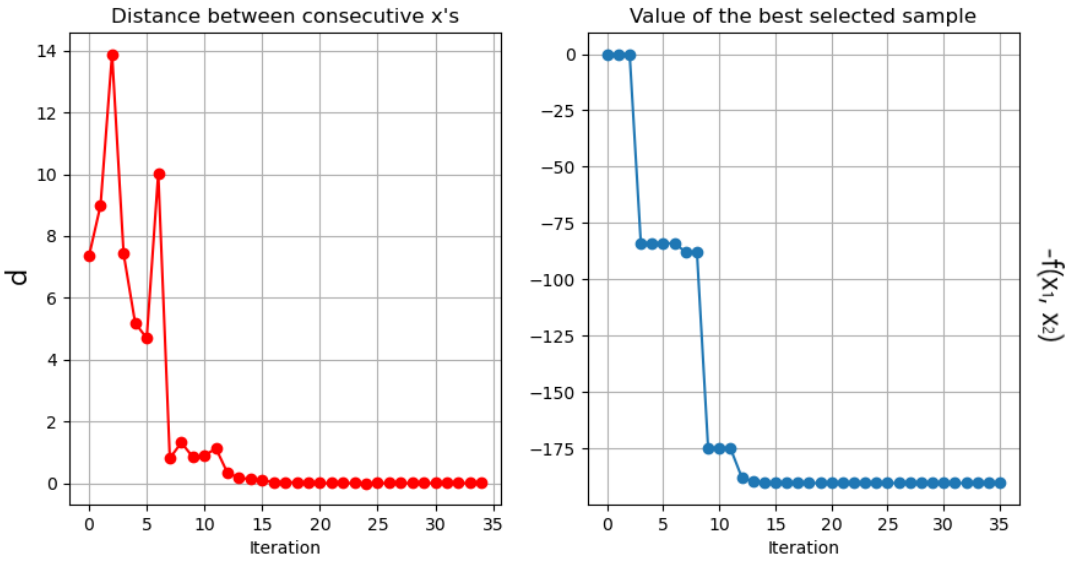

Figure 27: Convergence of the Bayesian optimisation algorithm using upper confidence bound with default parameter value $\nu = 2$: on the left - the distance between the last two observations (where d is calculated as in eq. A.1), on the right - the largest negative value of $-f(x_1, x_2)$ obtained relative to the previous iterations.

Hence, it may seem that an upper confidence bound acquisition function might not be suitable for this two-dimensional problem, but before we make this conclusion we aim to test if defining a better trade-off between exploration and exploitation can improve our optimisation results.

So in order to give more importance to the exploration of the uncertain areas, we increase our value of control parameter to $\nu = 3$ and perform Bayesian optimisation once more. Looking at initial output on Figure 28, we immediately notice that optimisation results have improved:

```
================================================================================
Value of (x,y) that maximises the objective:[7.93924713 9.66961688]
Maximum value of the objective: 307.2626142810332
================================================================================
```

Figure 28: Location of the global maximum of $f(x_1, x_2)$ obtained using Bayesian optimisation with upper confidence bound as acquisition function with parameter value $\nu = 3$.

In fact, the maximum value $f(7.939, 9.669) = 307.262$ is indeed a global maximum and is almost identical to the results we obtained when using expected improvement as our acquisition function (Figure 20). This resemblance is also visible in posterior mean (Figure 29) and standard deviation (Figure 30 on the left) plots, where locations of each iteration have a fairly even spread throughout the whole domain, indicating a fine balance between exploration and exploitation. Further evidence of that is also present on the left graph in Figure 31, where large distances between consecutive evaluation locations suggest that much more exploration of uncertain areas was done using parameter value $\nu = 3$ than using $\nu = 2$.

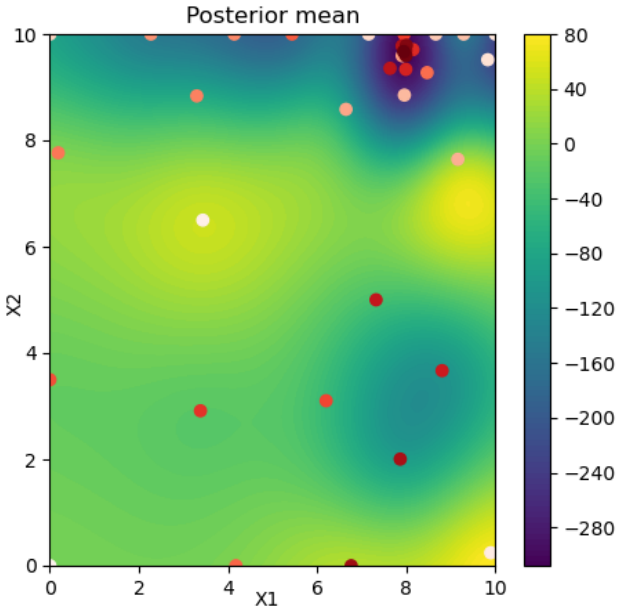

Figure 29: Two-dimensional posterior mean plot obtained after performing Bayesian Optimisation using upper confidence bound with parameter value $\nu = 3$ and 35 iteration budget. The round dots represent location of each iteration.

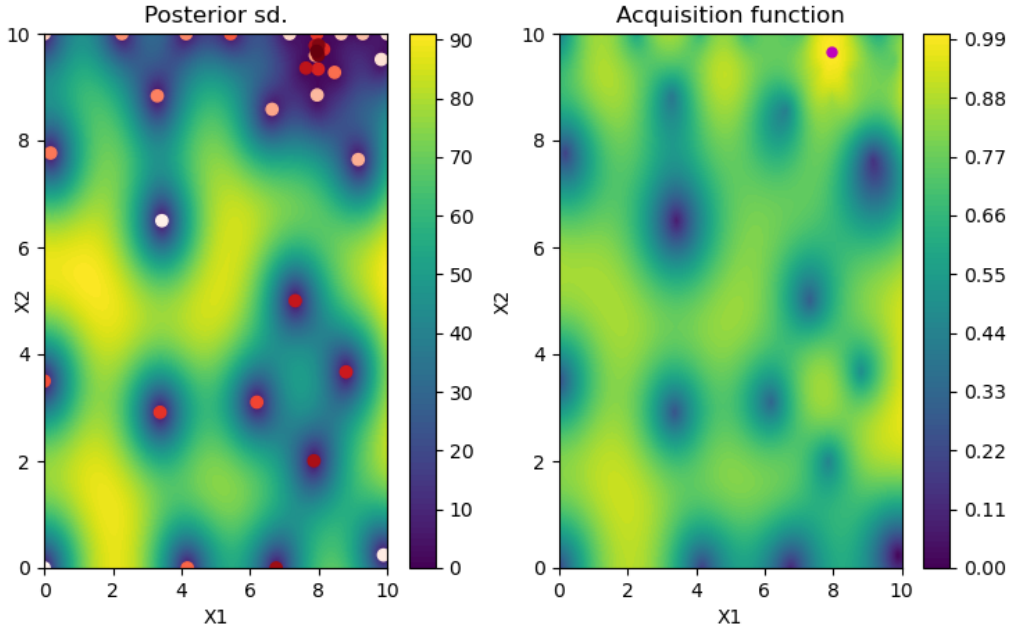

Figure 30: On the left - two-dimensional posterior standard deviation plot (where the dots represent location of each iteration), on the right - the current state of upper confidence bound function (with parameter $\nu = 3$) after 35 iterations (where the pink dot represent location of the next evaluation if we were to perform 36th iteration).

On the other hand, even though the upper confidence bound acquisition function with parameter value $\nu = 3$ (illustrated on the right in Figure 30) contains much more accurate information about the location of the global maximum after the optimisation budget was exhausted, there is still no absolute certainty of the location of the global maximum - there are many yellow/green areas which indicate potential locations of the optimum, most of which correspond to the regions of higher posterior standard deviation $\sigma(\mathbf{x})$. However, this is what we should expect if we recall that the

upper confidence bound function is defined using $\sigma(\mathbf{x})$ directly: GP-UCB$(\mathbf{x}) = \mu(\mathbf{x}) + \nu\sigma(\mathbf{x})$, hence regions of high uncertainty lead to higher values of GP-UCB$(\mathbf{x})$.

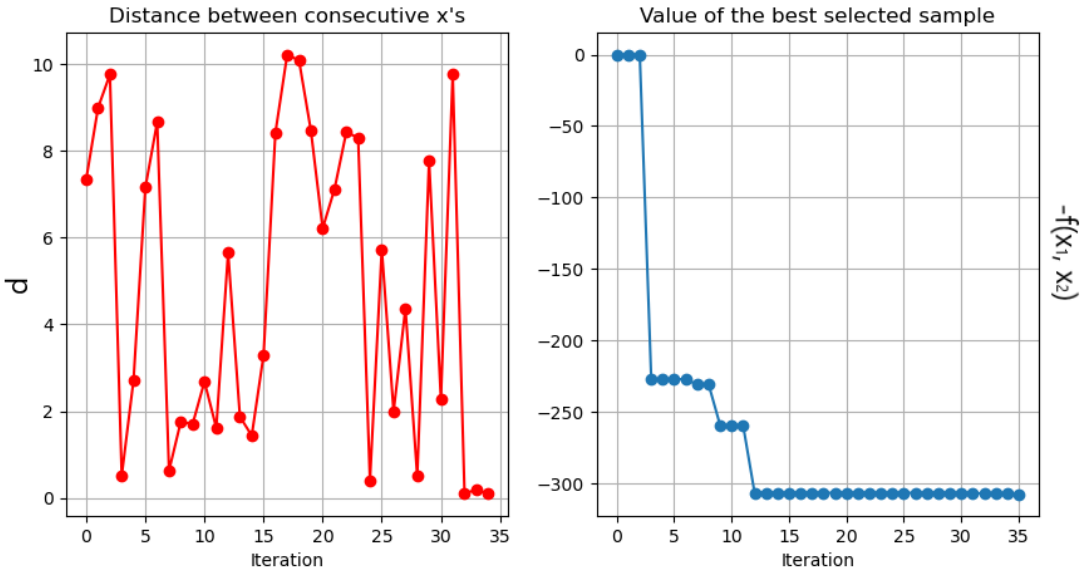

Figure 31: Convergence of the Bayesian optimisation algorithm using upper confidence bound with parameter value $\nu = 3$: on the left - the distance between the last two evaluation locations (where d is calculated as in eq. A.1), on the right - the largest negative value of $-f(x_1, x_2)$ obtained relative to the previous iterations.

Lastly, looking at the graph on the right in the Figure 31 we notice that the best optimum value obtained has not changed significantly since the 12th iteration, suggesting a great efficiency in detecting global maximum. Thus, we may conclude that defining a better trade-off between exploration and exploitation (increasing the parameter value ν in this case) can significantly improve the performance of the upper confidence bound function. One could also argue that this choice of acquisition function may be preferred over expected improvement for our two-dimensional optimisation problem as when using upper confidence bound function the global maximum was located (up to a slight difference of ± 0.5) slightly quicker - 12 iterations for UCB compared to 16 for expected improvement[7] (when comparing the graphs on the right in Figures 31 and 23).

4.2.1 Comparison with the *optim* function in R

Once again, we aim to compare Bayesian optimisation method with a more popular gradient-based general-purpose optimisation tool *optim* in R. Similarly as in the previous section, we continue with our two-dimensional problem set up, where we aim to find the maximum of $f(x_1, x_2) = (x_1^2 + x_2^2)(\sin^2(x_1) - \cos(x_2))$ with a 35 evaluation budget within the region of interest $x_1, x_2 \in [0, 10]$. Since choosing a starting point of $x_1 = x_2 = 0$ does not yield any results (optimisation stops after few iterations falsely concluding that our objective function $f(x_1, x_2)$ is just a flat surface, similarly as it happened in one-dimensional case), we choose to perform the optimisation at an arbitrary starting point $x_1 = x_2 = 1$. Initiating the *optim* function using "L-BFGS" method [19] with a predefined region of interest we obtain results displayed in Figure 32.

[7]Refer to Figure 37 in the appendix for precise evaluations at each iteration using both acquisition functions

Iteration	Starting point	1	2	3	4	5	6	7	8	9	10	11
x_1	1	1.001	0.999	1	1	3.1541	3.1551	3.1531	3.1541	3.1541	9.8938	9.8948
x_2	1	1	1	1.001	0.999	3.0185	3.0185	3.0185	3.0195	3.0175	10	10
$f(x_1, x_2)$	0.3355	0.3377	0.3334	0.3376	0.3335	18.9186	18.9253	18.9119	18.9269	18.9102	206.4661	206.6464

Iteration	12	13	14	15	16	17	18	19	20	21	22	23
x_1	9.8928	9.8938	9.8938	10	10	9.999	10	10	9.9155	9.9165	9.9145	9.9155
x_2	10	10	9.999	0	0	0	0.001	0	7.9550	7.9550	7.9550	7.9560
$f(x_1, x_2)$	206.2860	206.4661	206.5528	-70.4041	-70.4041	-70.4813	-70.4041	-70.4041	52.1810	52.3219	52.0404	52.3469

Iteration	24	25	26	27	28	29	30	31	32	33	34	35
x_1	9.9155	9.8982	9.8992	9.8972	9.8982	9.8982	10	10	9.999	10	10	10
x_2	7.9540	9.5806	9.5806	9.5806	9.5816	9.5796	9.5485	9.5485	9.5485	9.5495	9.5475	9.5538
$f(x_1, x_2)$	52.0151	226.9129	227.0908	226.7354	226.9063	226.9194	246.2922	246.2922	246.0920	246.2931	246.2911	246.2948

Extra iterations	36	37	38	39	40	41	42	43	44
x_1	10	9.999	10	10	10	10	9.999	10	10
x_2	9.55379	9.5538	9.5548	9.5528	9.5538	9.5538	9.5538	9.5548	9.5528
$f(x_1, x_2)$	246.2948	246.0946	246.2947	246.2947	246.2948	246.2948	246.0946	246.2947	246.2947

Figure 32: Two-dimensional function optimisation results using *optim* function with "L-BFGS-B" method in R. The best obtained value using 35 iteration budget is highlighted in red. Few extra iterations are provided for further analysis. [24]

Clearly, the obtained maximum value (highlighted in red) $f(x_1 = 10, x_2 = 9.5538) = 246.2948$ is not the global maximum, but instead correspond to one of the local peaks. If we were to increase our evaluation budget (visualised as "extra iterations" in Figure 32), our results would not improve, since the distance between consecutive observations become negligible and the optimisation process is terminated after the 44th evaluation. We further investigate the effectiveness of the *optim* function by running the optimisation at 10 different starting values x_1, x_2, which are all randomly sampled (using *runif* [20] function in R) from the region of interested $[0, 10]$. Unfortunately, this does not improve the search of the global maximum:

x_1	1.1370	6.0927	8.6092	0.0950	6.6608	6.9359	2.8273	2.9232	2.8622	1.8672
x_2	6.2230	6.2338	6.4031	2.3255	5.1425	5.4497	9.2343	8.3730	2.6682	2.3223
Largest value of $f(x_1, x_2)$ obtained during optimisation	246.2948	0	190.2702	246.2948	147.5914	147.5914	190.2702	190.2702	30.973	30.973

Figure 33: Optimisation results for two-dimensional problem using *optim* function at 10 different randomly sampled starting locations x_1, x_2. [24]

As observed in Figure 33, none of the values obtained during 10 optimisation processes with different starting values corresponded to the global maximum of $f(x_1 = 7.95, x_2 = 9.64) = 307.243$. Thus, it seems that a two-dimensional problem, where we have one distinctive global maximum as well as few other local peaks, is too complex for the gradient-based optimisation tool *optim*. In contrast, using Bayesian optimisation, we have not only managed to detect the global maximum, but also do it using a starting point of $x_1 = x_2 = 0$, which was not possible when using the *optim* function in R. In addition to that, when using Bayesian optimisation we had the flexibility to freely specify the balance between exploration and exploitation, hence tailor our approach to the optimisation based on the complexity of the unknown function (since $f(x_1, x_2)$ had not only one distinctive global maximum, but also few local peaks, using upper confidence bound acquisition function we were able to increase explorative power and improve efficiency in detecting the optimum value). Hence, Bayesian optimisation method once again proves to be better fitted for challenging optimisation problems, and especially in those dealing with two-dimensional functions.

5 Conclusions and further extensions

We have shown how, using Bayesian optimisation, one has a unique and efficient way of obtaining the global optimum of objective function without even knowing the explicit form it takes. Not only that, but we have also introduced the concept of acquisition functions, which drive the search in obtaining the optimum and give us the flexibility of how we want to approach the problem - whether we have a large evaluation budget, possibly expect just a single distinctive global optimum within the region of interest and hence should focus on exploitation, or we expect the function to be more complex and should feel the need to give more importance to the exploration - different acquisition functions enable us to adapt Bayesian optimisation algorithm so it would be best fitted for solving a problem at hand. Throughout this paper, by analysing one and two-dimensional problems, we have compared three acquisition functions: probability of improvement, expected improvement and upper confidence bound. We have found out that the expected improvement function provides us with a well balanced trade-off between exploration and exploitation, hence being well suited for efficiently optimising functions which contain many local extrema, but only a single global extremum. In contrast, when using probability of improvement, we observed much more exploitation of the local areas of the domain after only little exploration was done, indicating that probability of improvement acquisition function is better suited when we have a larger evaluation budget or expect an objective function with only one distinctive extremum. Lastly, we have shown that by adjusting the control parameter value ν of the upper confidence bound function we were able to define the trade-off between exploration and exploration ourselves and hence tailor this acquisition function to the user needs (which may differ based on the evaluation budget or any prior knowledge about the objective function available). It is also important to mention that rather than treating parameter value ν in the upper confidence bound function as a scalar and trying to find an appropriate value for it, in practice one could define an increasing function $\nu(i)$ instead, where i corresponds to the i-th iteration. This way defined control parameter ν gets bigger with every iteration (hence increasing exploration) and guaranteeing to continue exploring without falling into a local optimum and stopping the optimisation process too early [12].

One should note that in this paper we have only focused on optimising one and two-dimensional functions, but in fact, it is possible to implement Bayesian optimisation method for even higher dimensional problems[8] (in most problems it is most efficient for up to 10 dimensions [8]) and explore many other unknown objective functions. However, since Bayesian optimisation relies on Gaussian process regression, this optimisation method may be limited to only those objective functions, which can be modelled by Gaussian process regression (for instance, step functions are not suitable for Gaussian process regression [1]). Alternatively, one could explore if it is possible to improve the performance of Bayesian optimisation by using a surrogate model different than Gaussian processes, i.e., random forest regression or Parzen Tree estimator [14]. Lastly, the analysis discussed in this paper could be easily extended by comparing how different acquisition functions perform at different starting locations (different from $f(x = 0)$) using evaluation budgets of various sizes.

Overall, the flexible nature of the Bayesian Optimisation method suggests a wide range of potential areas for future applications: it seems that Bayesian optimisation is particularly well-positioned to offer impact in chemistry, chemical engineering, automation, materials design, and drug discovery, where practitioners undertake design efforts involving repeated physical experiments consuming years of effort and substantial monetary expense [3, 8]. These scientific areas not only create demand for future development of the Bayesian optimisation method, but also encourages the discovery of new potential applications.

[8]One should keep in mind that for higher dimensional problems we still have to optimise acquisition function, which has the same dimensionality as the original unknown objective function. Hence even though acquisition functions are cheap to evaluate and the exact form of them is known, optimising a very high dimensional function can be still quite challenging

A Appendix

The distance between two points

$$\{x_{1|\text{at n-th iteration}}, x_{2|\text{at n-th iteration}}\} \text{ and } \{x_{1|\text{at (n-1)-th iteration}}, x_{2|\text{at (n-1)-th iteration}}\}$$

in the x_1x_2-plane is calculated as (for n-th iteration, where $n = 1, 2, 3...$):

$$d = \sqrt{(x_{1|n} - x_{1|(n-1)})^2 + (x_{2|n} - x_{2|(n-1)})^2} \tag{A.1}$$

Iteration	x	f(x)
Known data point	0	0
1	1.5998	2.46E-15
2	0.7977	1.39E-09
3	1.5999	3.38E-17
4	0.0020	3.80E-15
5	0.4475	0.0196
6	0.3551	0.0094
7	0.5290	0.1470
8	0.5460	0.0533
9	0.5239	0.1783
10	0.5189	0.2060
11	0.5130	0.2319
12	0.5063	0.2489
13	0.4974	0.2462
14	0.2351	0.0011
15	1.0797	0.8541
16	1.0909	1.1186
17	1.0960	1.1868
18	1.0992	1.2077
19	1.1036	1.2064
20	1.1569	0.0810
21	1.4222	0.0032
22	0.6817	0.3617
23	0.1257	0.0096
24	0.9188	0.6469
25	0.9657	0.0171
26	0.8769	0.5126
27	1.3116	1.5568
28	1.3063	1.6564
29	1.3037	1.6825
30	1.3006	1.6911
31	1.2741	0.9722
32	1.5077	2.1752
33	1.5040	2.2355
34	1.5015	2.2508
35	1.4977	2.2341

(a) Using probability of improvement

Iteration	x	f(x)
Known data point	0	0
1	1.4732	1.2569
2	1.3300	0.8861
3	1.6000	0.0000
4	1.4250	0.0064
5	1.4794	1.5922
6	1.4874	1.9644
7	1.5039	2.2360
8	1.5266	1.3563
9	1.5002	2.2505
10	1.2602	0.4507
11	1.1490	0.1806
12	1.0293	0.0081
13	0.4353	0.0040
14	0.6418	0.0214
15	0.8029	0.0000
16	0.2395	0.0022
17	0.1206	0.0106
18	0.9164	0.6862
19	0.5395	0.0846
20	0.3351	0.0430
21	0.8681	0.3422
22	0.9657	0.0170
23	0.7221	0.3608
24	1.2083	0.0000
25	1.3686	0.0213
26	1.0927	1.1483
27	0.0605	0.0011
28	0.1804	0.0000
29	1.2982	1.6812
30	0.4880	0.2141
31	0.5909	0.0000
32	0.3839	0.0000
33	0.2872	0.0730
34	1.0672	0.4947
35	0.6857	0.4039

(b) Using expected improvement

Figure 34: Evaluation values for each iteration on one-dimensional problem (see section 4.1).

iteration	1	2	3	4	5	6	7	8	9
x	0	0.001	0	1.50E-14	0.001	0	3.17E-15	0.001	0
f(x)	0	1.50E-17	0	3.89E-104	1.50E-17	0	1.53E-109	1.50E-17	0

Figure 35: One-dimensional function optimisation results using the *optim* function in R with a starting point $x = 0$.

x	0	0.1	0.22	0.5	0.77	0.85	1	1.1	1.3	1.5
Largest value of f(x) obtained during the optimisation	0	0.811	0.251	2.251	0.251	1.211	0	1.211	1.691	2.251

Figure 36: One-dimensional function optimisation using the *optim* function in R with different starting values x. Values highlighted in yellow and red correspond to local and global maximum, respectively.

	Expected Improvement			UCB with v = 2			UCB with v = 3		
Iteration	x_1	x_2	$f(x_1, x_2)$	x_1	x_2	$f(x_1, x_2)$	x_1	x_2	$f(x_1, x_2)$
Known data point	0	0	0	0	0	0	0	0	0
1	0.8158	0.1953	-0.3171	3.4318	6.4997	-48.3389	3.4318	6.4997	-48.3389
2	9.3330	5.1942	-51.9065	9.8930	0.2439	-75.0876	9.8930	0.2439	-75.0876
3	0	10	83.9072	0	10	83.9072	10	10	227.0061
4	0	7.8444	-0.5902	3.2968	3.3453	22.1303	9.8259	9.5173	214.8532
5	0.7902	10	135.2287	8.0934	1.3820	50.9717	7.1536	10	215.2164
6	1.3785	10	183.6814	3.4868	0.4801	-9.5693	0	10	83.9072
7	2.2821	10	148.6496	0.1131	9.9210	87.8202	8.6599	10	230.7393
8	1.6139	9.6786	189.2960	0	9.1249	79.5469	9.2867	10	159.7985
9	1.6344	9.8691	190.0200	1.2497	9.5276	174.9886	7.9559	8.8521	259.2457
10	2.1586	8.6893	114.9558	2.0206	9.8591	174.0187	9.1549	7.6443	-19.5062
11	5.8457	10	136.6605	1.9340	8.9638	148.7897	6.6448	8.5844	93.3757
12	7.3260	10	243.6120	1.5101	10	187.7241	7.9060	9.5924	306.4546
13	8.0926	10	295.1034	1.6316	9.6975	189.4761	2.2662	10	150.1908
14	8.8652	10	200.1697	1.6361	9.8698	190.0049	4.1374	10	180.7473
15	7.9900	9.5188	305.3734	1.5230	9.8083	189.6629	3.2964	8.8365	76.1124
16	7.9894	9.6854	307.0785	1.6082	9.8049	190.2599	0.1868	7.7657	-3.2369
17	7.1955	8.5376	156.7236	1.5928	9.8342	190.2462	8.4649	9.2746	261.6881
18	8.1150	9.6241	303.2610	1.5912	9.7986	190.2423	4.1749	0	-4.5683
19	8.7365	8.3991	135.4084	1.60236	9.81782	190.27008	5.4341	10	181.6727
20	6.2052	0	-38.2699	1.60191	9.81668	190.27020	0	3.4983	11.4678
21	3.5073	4.5788	8.6860	1.60194	9.81660	190.27020	6.1992	3.1031	48.3620
22	7.9515	9.6432	307.2425	1.60195	9.81655	190.2702006	7.9484	10	298.6439
23	10	0	-70.4041	1.60195	9.81652	190.2702009	3.3788	2.9141	20.4944
24	7.9717	9.6422	307.1960	1.60197	9.81651	190.2702011	8.1467	9.7071	301.4628
25	0	3.6234	11.6344	1.60198	9.81649	190.2702013	7.9880	9.3363	298.6642
26	4.6174	7.8886	85.6924	1.60198	9.81651	190.2702010	8.8017	3.6688	109.5454
27	6.2646	3.6264	46.3763	1.60198	9.81650	190.2702012	7.3197	5.0013	35.8249
28	3.6419	1.7610	6.8599	1.60198	9.81648	190.2702013	7.6327	9.3557	284.2062
29	5.9424	5.9788	-59.8514	1.60197	9.81653	190.2702008	7.9029	9.7766	306.0174
30	7.8951	9.5704	305.9612	1.60197	9.81647	190.2702014	7.8626	2.0056	93.5755
31	8.2577	2.4209	118.2603	1.60199	9.81649	190.2702013	6.7622	0	-36.0145
32	10	2.7491	131.2096	1.60198	9.81647	190.2702014	7.9859	9.6905	307.0985
33	1.6827	5.9686	1.4075	1.60197	9.81648	190.2702014	7.9994	9.7876	305.8183
34	4.2640	10	195.1333	1.60196	9.81651	190.2702011	7.9901	9.5952	306.6894
35	3.9427	9.0809	142.8164	1.60197	9.81648	190.2702014	7.9392	9.6696	307.2626

Figure 37: Evaluation values for each iteration on two-dimensional problem using different acquisition functions (see section 4.2).

References

[1] Carl Edward Rasmussen and Christopher K. I. Williams, Gaussian Processes for Machine Learning, The MIT Press, 2006.

[2] Neil Lawrence: Introduction to Gaussian Processes, Summer School at Sheffield, talk given on September 14, 2015
https://www.youtube.com/watch?v=N4W8FOL3JU0&ab_channel=OpenDataScienceInitiative

[3] Prof. Peter I. Frazier, A Tutorial on Bayesian Optimization, Cornell University, July 10, 2018.

[4] Kevin P. Murphy, Machine Learning: A Probabilistic Perspective, The MIT Press Cambridge, Massachusetts, 2012.

[5] Nando de Freitas, Regression with Gaussian processes, from lecture series at the University of British Columbia, in 2013 (one can find course material posted by UBC on
https://www.cs.ubc.ca/ nando/540-2013/lectures.html
and the link to the video lecture uploaded by N. Freitas on:
https://www.youtube.com/watch?v=MfHKW5z-OOA&t=1s&ab_channel=NandodeFreitas).

[6] NEOS project, Types of Optimization Problems, University of Wisconsin - Madison, 2020
https://neos-guide.org/optimization-tree

[7] Prof. J. Mockus, Bayesian Approach to Global Optimization: Theory and Applications, 1989;
Prof. J. Mockus, G. Reklaitis and W. Eddy, Bayesian Heuristic Approach to Discrete and Global Optimization: Algorithms, Visualization, Software, and Applications, 1996;
Prof. Jonas Mockus, A Set of Examples of Global and Discrete Optimization: Applications of Bayesian Heuristic Approach, 2000.

[8] Javier Gonzalez, Principal Researcher at Microsoft Research Cambridge, An Introduction to Bayesian Optimisation, Gaussian Process Summer School, 2009
https://www.youtube.com/watch?v=EnXxO3BAgYk&ab_channel=GaussianProcessSummerSchool

[9] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams and Nando de Freitas, Taking the Human Out of the Loop: A Review of Bayesian Optimization, Harvard University's Library, 2015

[10] Eric Brochu, Vlad M. Cora and Nando de Freitas, A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning, December 14, 2010

[11] J. Mockus, V. Tiesis, and A. Zilinskas, Toward Global Optimization, volume 2, chapter The Application of Bayesian Methods for Seeking the Extremum, Elsevier, 1978.

[12] N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger, Gaussian process optimization in the bandit setting: No regret and experimental design, International Conference on Machine Learning, 2010.

[13] E. Brochu, M. Hoffman, and N. de Freitas. Hedging strategies for Bayesian optimization, eprint arXiv:1009.5419, arXiv.org, September 2010.

[14] Ian Dewancker, Michael McCourt, Scott Clark, Bayesian Optimization Primer, 2016
https://static.sigopt.com/b/20a144d208ef255d3b981ce419667ec25d8412e2/static/pdf/SigOpt_Bayesian_O

Software, Code & Packages

[15] A Gaussian Process Regression tutorial adapted from Jupyter notebooks by Wil Ward, Rich Wilkinson and Neil Lawrence, 2012. These notebooks are available on
https://nbviewer.jupyter.org/github/SheffieldML/notebook/blob/master/GPy/index.ipynb
Adjusted code used for this paper:
https://www.codepile.net/pile/9mxa4eKG

[16] Adapted from an example of implementation of Gaussian Process regression in programming language Python (Original source [5]). Link to the original code:
https://www.cs.ubc.ca/ñando/540-2013/lectures/gp.py

Adjusted code used for this paper:
https://www.codepile.net/pile/N1ObPpGo

[17] GPy is a Gaussian Process (GP) framework written in python, from the Sheffield machine learning group. Documentation and original source can be found here:
https://sheffieldml.github.io/GPy/

[18] GPyOpt is a Python open-source library for Bayesian Optimization developed by the Machine Learning group of the University of Sheffield. It is based on GPy [17], a Python framework for Gaussian process modeling. Documentation and original source can be found here:
https://github.com/SheffieldML/GPyOpt

[19] Documentation for general-purpose optimization tool *optim* in R:
https://www.rdocumentation.org/packages/stats/versions/3.6.2/topics/optim

[20] Documentation for generating random values from uniform distribution using runif function in R
https://www.rdocumentation.org/packages/stats/versions/3.6.2/topics/Uniform

[21] Link to the code used for one-dimensional Bayesian optimisation in section 4.1:
https://www.codepile.net/pile/bagM92eQ

[22] Link to the code used for two-dimensional Bayesian optimisation in section 4.2:
https://www.codepile.net/pile/1op95gWJ

[23] Link to the code used for one-dimensional problem optimisation with *optim* function in R on section 4.1.1:
https://www.codepile.net/pile/bglZVXW1

[24] Link to the code used for two-dimensional problem optimisation with *optim* function in R on section 4.2.1:
https://www.codepile.net/pile/4KL856KG