New Sampling Strategies for Bayesian Optimization of Functions with Location-Dependent Sample Noise

Conor McGrory
Adviser: Jonathan Pillow

This thesis represents my own work in accordance with University regulations.

Acknowledgements

This project would not have been possible without the help of many people.

First and foremost, I would like to thank my parents for being supportive in every possible way over the past four years, and my brother, for being a constant source of insight about many things.

I would like to thank my professors, especially William Howarth, Doug Clark, Thomas Kelly, Ramon Van Handel, and Elad Hazan, for showing me new ways to think, and Jugal Kalita, for introducing me to the amazing field of machine learning.

I would like to thank Jeremy Cohen, for discussing many of the ideas contained in this paper with me a great many times, and Abby Van Soest, for helping me figure out how I was going to explain all of it.

I would also like to thank Barbara Engelhardt, for helping me do the background research for this project over the summer and for being my second reader.

Finally, I would like to thank Jonathan Pillow, for working on this project with me for the past year and helping me develop it from a vague idea into a full-blown research project.

1. Introduction

1.1. Black-box optimization

Global optimization — the task of finding the global maximum or minimum of a function on some domain — is a common problem in many different scientific and engineering disciplines. In its most basic form, this problem is very general, but in order to develop methods for solving optimization problems, it helps to consider specific classes of objective functions. For convex optimization, we assume that the objective function and the domain we are optimizing it over are both convex. For many numerical optimization methods, we assume that the objective function has a defined gradient and Hessian that we can evaluate anywhere on the domain. These approaches work for many families of objective functions, but there are also some functions we would like to optimize that don't meet the criteria for these more standard methods. These functions don't have closed mathematical forms, and we can't assume that they satisfy properties like linearity or convexity. Aside from some very minimal assumptions — one being the universal assumption that any function we are trying to optimize *has* an optimum on the given domain — we don't know anything about these functions besides their output values corresponding to different input values. Such functions are referred to in the literature as "black-box" functions.

Phenomena commonly treated as black-box functions include functions that are the result of some numerical procedure, like Monte Carlo integrals, and measurements of unknown physical quantities, like the relative amount of oil that can be obtained by drilling at a certain point on the Earth's surface. In both of these cases, there are two additional complications that are common to many phenomena modeled as black-box functions. The first is that we often can't observe the value of the function directly — it is distorted by some level of random noise. The second is that observing the value of the function at some specific input — *sampling* it, in technical parlance — is very expensive in terms of time, money, or computational resources. In this case, we want to find a good estimate for the optimum using as few observations as possible.

For the rest of this paper, for the sake of simplicity, we will consider the global optimization problem in terms of maximization. Although the convention is usually to use minimization, much of the current literature on black-box optimization uses the maximization setting, so we will remain consistent with this work.

1.2. Bayesian optimization

Because our knowledge about black-box functions is extremely limited, we can't apply traditional optimization techniques, like gradient descent or Newton methods, to find their maximizing values. Also, because the functions are often very expensive to evaluate, brute-force methods don't work well either. Instead, we need to find some way to estimate the maximizing value of a black-box function by observing its values at different input points. One solution to this problem, first proposed in the late 1970s[11], is to model the objective function as a random variable. Using a set of samples of the function, we can infer a probability distribution over its possible values at different points on the domain, and then use this distribution to select additional points to sample at, until we have enough data to construct an estimate of the function's maximizing value. Intuitively, we use this distribution to quantify our beliefs about the objective function, given our current samples, so that we can be smarter about which points we choose to sample at next. This technique is known as

Bayesian optimization, and it has proven to be effective for optimizing many different types of black-box functions.

One of the great things about approaching optimization probabilistically is that it makes it much easier to optimize functions which we can only observe through noisy samples. The vast majority of the Bayesian optimization literature handles noisy samples by assuming that the level of the noise is constant for all points on the domain. This is reasonable to assume in many cases, but there are also other cases, specifically in the fields of machine learning and data modeling, where the noise level is drastically different depending on where on the domain the function is sampled.

1.3. Overview of project

The purpose of this project was to develop modifications for the standard Bayesian optimization method that improve its performance in cases where the level of the sample noise varies as a known function of the point where the function is being sampled. We created two new criteria for selecting sample points — these are called *acquisition functions* in the literature — that make use of this extra information and tested their performance on synthetic black-box optimization problems. In our experiments, Bayesian optimization using these two new acquisition functions to select points consistently outperformed Bayesian optimization using more standard methods.

This paper is structured as follows. First, we discuss relevant background information pertaining to machine learning (Section 2) and Bayesian optimization (Section 3). Then, we give a formal definition of the specific problem we are trying to solve (Section4) and explain our new acquisition functions (Section5). Finally, we explain our experiments and discuss their results (Section 6).

2. Machine Learning Background

2.1. Regression: Learning a real-valued function from samples

All supervised learning problems can be mathematically formulated as the task of producing estimates of the values of an unknown function for different inputs, given a training set of known input-output pairs. If the range of the function is a finite set of values, this task is called *classification*, and if the range is instead the real line, it is called *regression*. For our purposes in this paper, we will assume the regression model, where the unknown function, denoted by f, is a real-valued function on a closed and bounded domain $\mathcal{X} \in \mathbb{R}^d$. As input, the learning algorithm is given a set $D = \{(x_1, y_1), (x_2, y_2), ...(x_N, y_N)\}$ of *samples* of the function, where, for each pair $(x_i, y_i) \in D$, x_i is a point in $\mathcal{X}$ (called the *sample point*), and y_i is a real value (called the *sample value*) that is related to the unknown function value $f(x_i)$ either directly or through the addition of random noise.

In the most basic case, the samples in D are direct samples of the function; for each x_i, we have $y_i = f(x_i)$. Learning in this case is commonly referred to as *interpolation*. In many real-world applications, however, the function f cannot be sampled directly. Instead, we assume that the sample value y_i is equal to the function value $f(x_i)$ plus some level of mean-zero Gaussian noise. This makes y_i a random variable with distribution equal to

$$y_i \sim \mathcal{N}(f(x_i), \sigma_i^2)$$

The variance σ_i^2 of y_i is called the *sample noise variance*. We commonly assume that the noise values for each sample are independent random variables – this makes the y_i independent and identically distributed. Many models assume that σ_i^2 is the same for all i; in this case, it is simply denoted σ^2. Because the sample noise variance here is constant over the domain, we refer to this as the *constant sample noise* assumption. Another approach is to assume that σ_i^2 depends on the sample point x_i; here it is written $\sigma^2(x_i)$ and treated as a function of x_i. Because the sample point x_i is also sometimes called the sample *location*, we refer to this as the *location-dependent sample noise* assumption. In either case, as long as we know the value of σ_i^2 for each x_i, we can perform regression on the set of sample points and corresponding observations to get an estimate of the function.

2.2. The classical approach to learning

In the classical formulation of supervised learning, a learning algorithm uses the sample data D to construct a function h_D, called the *hypothesis*, which predicts the values of f for all inputs on the domain. By treating h_D as an estimator of f, we can prove bounds on its expected error as a function of the number of samples in the dataset. In order to encode our prior knowledge about the function we are trying to learn — which is essential to the success of the learning algorithm — we assume that the hidden function f is in a specifc set $\mathcal{H}$ of functions on the domain, which we call the hypothesis class. Then, the learning algorithm only returns hypotheses from $\mathcal{H}$. By setting $\mathcal{H}$ to be a family of functions parametrized by a vector of real-valued parameters, we reduce the problem of learning f to the problem of learning the correct value of this vector, which can often be cast as an optimization problem.

Take, for example, the case of simple linear regression. Here, we assume that f is a function of the form $f(x) = w^T x + w_0$, which is equivalent to defining the hypothesis class as:

$$\mathcal{H} = \{h : \mathcal{X} \to \mathbb{R} | h(x) = w^T x + w_0, w \in \mathbb{R}^d, w_0 \in \mathbb{R}\}$$

For simplicity, we will denote the parameter vector (w, w_0) by θ. Because any value of θ specifies a function in $\mathcal{H}$, choosing a hypothesis $h_D \in \mathcal{H}$ based on the data is equivalent to choosing an estimate $\hat{\theta}_D \in \mathbb{R}^{d+1}$ of the parameter vector.

How, then, do we use D to choose $\hat{\theta}_D$? One common approach in the classical learning setting is to choose the value of θ that maximizes the probability of the observed data under the model. This is called a *maximum likelihood* (ML) estimate. Formally, we can define this as

$$\hat{\theta}_D = \arg\max_{\theta \in \mathbb{R}^{d+1}} p(D|\theta)$$

For the linear regression case, the data points (x_i, y_i) are assumed to be independently drawn with Gaussian noise, so the likelihood of set of samples $D = \{(x_1, y_1), (x_2, y_2), ...(x_N, y_N)$ for a particular value of θ is

$$p(D|\theta) = \prod_{i=1}^{N} \mathcal{N}(y_i | w^T x_i + w_0, \sigma^2)$$

To compute the maximizing point of $p(D|\theta)$, we can solve the equivalent problem of minimizing the negative log likelihood function, defined as:

$$NLL(\theta) = -\log p(D|\theta)$$

Expanding the likelihood function gives us a formula for the *NLL* function:

$$\begin{aligned} NLL(\theta) &= -\log \Big[\prod_{i=1}^{N} \mathcal{N}(y_i | w^T x_i + w_0, \sigma^2) \Big] \\ &= -\frac{1}{2\sigma^2} \sum_{i=1}^{N} (y_i - w^T x_i - w_0)^2 - \frac{N}{2} \log(2\pi\sigma^2) \end{aligned}$$

Because N and σ^2 are constants, minimizing the negative log likelihood is equivalent to minimizing the objective function J, defined as:

$$J(\theta) = \sum_{i=1}^{N} (y_i - w^T x_i - w_0)^2$$

Therefore, we can express $\hat{\theta}_D$ as

$$\hat{\theta}_D = \arg\min_{\theta \in \mathbb{R}^{d+1}} J(\theta)$$

Because $J(\theta)$ is the sum of the square differences between the observed y_i values and their corresponding predicted values $w^T x + w_0$ under the model, this method is commonly referred to as *least squares* regression. In this case, there also exists an analytic solution for $\hat{\theta}_D$. If the data is expressed in matrix form, where $D = (X, y)$, then this solution is

$$\hat{w}_D = (X^T X)^{-1} X^T y$$

However, in most cases of maximum likelihood estimation, an analytic solution for the minimizing point of the objective function doesn't exist, so $\hat{\theta}_D$ has to be found using a numerical method. [12]

Once we have a value for $\hat{\theta}_D$, we can use it to estimate the value of the function for a new point x_{N+1}:

$$\hat{f}(x_{N+1}) = \hat{w}^T x_{N+1} + \hat{w}_0$$

where $(\hat{w}, \hat{w}_0) = \hat{\theta}_D$. Therefore, by restricting our set of possible functions to a parametrized set and estimating the values of the parameters, we have created an estimate of the function.

One disadvantage of this approach is that it only allows us to encode "hard" assumptions about f into our model. By choosing a particular hypothesis class $\mathcal{H}$, we are encoding the assumption that f is *certainly* a function in $\mathcal{H}$, and not any other type of function. There is no way to encode the assumption that f is *more likely* one type of function than another. For the linear regression example, by picking $\mathcal{H}$ to be the set of all affine functions, we encode the "hard" assumption "*f is an affine function*" into our model. However, there is no hypothesis class that allows us to encode the "soft" assumption "*f is an affine function, and the likelihood of f having a particular coefficient vector w is equal to $p(w)$*". In the case where some of our prior knowledge about f takes the form of probabilistic assumptions, the classical framework doesn't allow us to incorporate all that we know into our model.

2.3. The Bayesian approach to learning

Bayesian learning is a formulation of supervised learning which, among other things, allows us to encode probabilistic prior assumptions about f into our model. In the Bayesian formulation, the task of machine learning is split into the two subtasks of *inference* and *prediction*. First, we use the samples in D and prior assumptions about f to infer a distribution over possible f. Then, we use this distribution to make predictions about the values of f at new locations. The key difference here is that, unlike the classical formulation, the Bayesian forumlation treats the unknown function f as a random variable. Bayesian learning takes its name from Bayes' Rule, the law of probabilistic inference named after the Reverend Thomas Bayes, one of the key figures in the development of probability theory. In its most general formulation, Bayes' rule relates the marginal and conditional probabilities of two events A and B:

$$p(A|B) = \frac{p(B|A)p(A)}{p(B)}$$

The contribution of Bayesian statistics is not Bayes' rule itself – in our modern axiomatic theory of probability, this formula is a just direct consequence of the definition of marginal and conditional probability – but its application as a means of using evidence to update prior beliefs. Specifically, if H is a hypothesis, and D is a set of observations, the rule can be used to express the conditional probability of the hypothesis H given our observations D:

$$p(H|D) = \frac{p(D|H)p(H)}{p(D)}$$

Here, $p(H)$ is our assumption about the probability of H before we see any data. This is called the *prior* distribution because it represents our prior beliefs. $p(D|H)$ is our assumption about the probability of observing data D if the hypothesis H is true. This is called the *likelihood*. $p(D)$ is the marginal probability of observing D, so it is called the *marginal likelihood*. Finally, $p(H|D)$ is the probability that H is true given that we observed D. This is called the *posterior* distribution, because

it represents our beliefs about H after observing D. Although we illustrated Bayes' rule here with discrete probability, the same relationship holds in the case of continuous distributions, where the probability mass functions $p(D)$, $p(H)$, $p(D|H)$, and $p(H|D)$ are replaced with probability density functions. In order to carry out Bayesian inference, the only distributions we need to specify are the prior $p(H)$ and the likelihood $p(D|H)$; the marginal likelihood can be obtained by summing (or, in the continuous case, integrating) over possible values of H. Given these quantities, we can use Bayes' rule to update our prior beliefs using the observed data.

As an example of how learning works in the Bayesian framework, consider our earlier example of linear regression. Say we want to encode into our model the assumption that each coefficient w_i is independently drawn from a Gaussian distribution with mean zero and variance τ^2:

$$p(w_i) = \mathcal{N}(w_i|0,\tau^2)$$

Using the independence assumption, this gives us the following joint prior density for the coefficient vector w:

$$p(w) = \prod_{i=1}^{N} \mathcal{N}(w_i|0,\tau^2)$$

If we assume nothing about the value of the intercept w_0, we can say that $p(\theta) = p(w)$. Using Bayes' rule, we can combine this prior distribution with the likelihood of the data to get a posterior distribution over possible values of θ:

$$\begin{aligned} p(\theta|D) &\propto p(D|\theta)p(\theta) \\ &= p(D|\theta)p(w) \\ &= \prod_{i=1}^{N} \mathcal{N}(y_i|w^T x_i + w_0, \sigma^2) \cdot \prod_{j=1}^{d} \mathcal{N}(w_i|0,\tau^2) \end{aligned}$$

In the Bayesian framework, the common way to produce an estimate $\hat{\theta}_D$ of the parameters of a function is to find the value of θ that maximizes the density function of the posterior distribution – in other words, the most likely value of θ, given the observed data. This is called the *maximum a posteriori* (MAP) estimate. For this case, we can write it as:

$$\hat{\theta}_D = \arg\max_{\theta \in \mathbb{R}^{d+1}} p(\theta|D)$$

In a similar way as for the ML estimate in least-squares regression, we can show that, with the likelihood function above, the MAP estimate can be found by minimizing an objective function $J(\theta)$:

$$\hat{\theta}_D = \arg\min_{\theta \in \mathbb{R}^{d+1}} J(\theta)$$

In the case of linear regression with a Gaussian prior on the weight vector w, this function is

$$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} (y_i - w^T x_i - w_0)^2 + \lambda w^T w$$

where $\lambda = \frac{\sigma^2}{\tau^2}$.[12] Notice that this function is just the objective function for least-squares regression with the extra term $\lambda w^T w$ added. This term, a constant multiple of the squared ℓ_2-norm of w,

penalizes large values of w, which helps to prevent the model from overfitting to the data. In the classical framework, this is called *regularization*, but in the Bayesian framework, it is just a consequence of the prior distribution – by assuming a Gaussian prior on the coefficients, we are assuming that large values of w are less likely than those close to zero, and this is reflected in the posterior. The minimum of this objective function also happens to have an analytic solution, which is

$$\hat{w}_D = (X^T X + \lambda I)^{-1} X^T y$$

Another difference between the classical and the Bayesian frameworks is their approach to estimation of the function values. In the classical framework, the estimated value of the function for an new input x_{N+1} is obtained by simply plugging the estimated parameters $\hat{\theta}_D$ into the general equation assumed for the hypothesis class and returning the value of the resulting function h_D for input x_{N+1}. Because all the classical approach gives us is an estimate of the hidden function, we simply evaluate it at any points we want to estimate. In the Bayesian approach, however, we have more than just a single estimator for the function – we have a probability distribution over possible parameters, and therefore, over possible functions. To estimate $f(x_{N+1})$, we compute the *expected* value of $f(x_{N+1})$ under the posterior. [12]

2.4. Nonparametric Bayesian learning

In the above example of Bayesian learning, there are two prior assumptions encoded into our model. The first of these is a "soft" assumption — the prior distribution over the parameter vector θ. The second, however, is a "hard" one — the assumption that the hidden function f is in the hypothesis class $\mathcal{H}$ parametrized by θ. In order for us to make prior assumptions about θ, we first have to assume that f belongs to a family of functions parametrized by θ. This observation raises the question of whether or not it would be possible to perform Bayesian learning *without* assuming a parametrized hypothesis class.

As it turns out, there are methods – called *nonparametric* methods – for performing Bayesian inference without explicitly defining a parametrized hypothesis class. The general idea behind this approach is that, instead of implicitly inferring a distribution over possible values of f by parametrizing $\mathcal{H}$ and inferring a distribution over possible values of θ, we *explicitly* infer a posterior distribution over possible f, given a prior and likelihood function. The application of Bayes rule in this case is

$$p(f|D) = \frac{p(D|f)p(f)}{p(D)}$$

How, though, do we define a probability distibution over the set of $\mathcal{X} \to \mathbb{R}$ functions? If $\mathcal{X}$ is finite, then this is just the distribution of a vector-valued random variable with dimension $|\mathcal{X}|$. However, if $\mathcal{X}$ is a subset of $\mathbb{R}^d$, this notion becomes harder to realize, both from a mathematical standpoint and a computational one. Thankfully, there exists a distribution over the set of $\mathbb{R}^d \to \mathbb{R}$ functions, called a *Gaussian process*, which has many ideal properties for use in Bayesian machine learning.

2.4.1. Stochastic processes

In order to understand what a Gaussian process is, it helps to first consider the more general concept of a stochastic process. A *stochastic process* is a way of mathematically representing a probability distribution over functions. Formally, it is defined as a set of random variables, indexed

by elements of some other set, which we call the *index set*. This is a very general definition, and practical examples of stochastic processes all involve additional assumptions about the relationship between the random variables and the index set. One common example of a stochastic process is a discrete-time random walk, which is essentially a sequence of random variables that represent the position values of an object at different times. In the simplest case, the range of any random variable X_t is the real line, and the index set, which contains all of the values of t is the set of natural numbers. For discrete-time random walks, one common assumption to make about the relationship between the random variables $\{X_t\}$ and the index set $\{t\}$ is the first-order Markov assumption – we assume that X_{t+1} is conditionally independent of all variables X_{t-k}, given the value of X_t. This assumption allows us to prove meaningful properties about the process.

This concept of a stochastic process is a natural way to describe systems that evolve probabilistically over time, but how does it relate to our notion of a distribution over functions? This is actually just a matter of interpretation. Each realization of a stochastic process represents a function from the index set to the range set of the random variables. Using the distributions of the random variables, and any assumptions we make about the relationship between the random variables and the index set, we can compute the probability — or, in continuous cases, the probability density — of any realization, and therefore, any function from the index set to the range set. Using our example from above, this means that we could think of a discrete-time random walk as a distribution over the set of possible paths that an object could take through space — that is, the set of functions from time values to position values.

2.4.2. Gaussian processes

A Gaussian process (GP) is a stochastic process where any finite subset of the random variables in the process have a joint Gaussian distribution.[1] If we denote random variables in a GP as $f(x)$, where x is a point in the index set, then for any finite vector $(x_1, x_2, ...x_k)$ of index points, the corresponding vector of random variables $(f(x_1), f(x_2), ...f(x_k))$ has a multivariate Gaussian distribution. More intuitively, we can think of a Gaussian process as a generalization of the multivariate Gaussian distribution to "vectors" of infinite length. In the same way that a multivariate Gaussian distribution is completely specified by its mean vector and covariance matrix, a GP is completely specified by its mean function μ and covariance function k (also called the *kernel*), which we can define as:[13]

$$\begin{aligned} \mu(x) &= \mathbb{E}[f(x)] \\ k(x,x') &= \mathbb{E}[(f(x)-\mu(x))(f(x')-\mu(x'))] \end{aligned}$$

Because these two functions completely specify the GP, we can write a GP the same way we write a Gaussian distribution. If a random function f has a GP distribution with mean function μ and covariance function k, we write:

$$f(x) \sim \mathcal{GP}(\mu(x), k(x,x'))$$

The definition of a GP is clearly very general. For our purposes, we will make the additional assumption that the index set $\mathcal{X}$ is a subset of $\mathbb{R}^d$. From the definition, we can see that if $f(x) \sim \mathcal{GP}(\mu(x), k(x,x'))$, then the GP induces the following Gaussian distribution on any set of points

[1] This section and the next both draw heavily on the introduction to Gaussian processes provided in [13]. For a more detailed expostition, the reader can look to Ch. 2 of this book.

$(x_1, x_2, \ldots x_n) \in \mathcal{X}^n$:

$$\begin{bmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_n) \end{bmatrix} \sim \mathcal{GP}\left(\begin{bmatrix} \mu(x_1) \\ \mu(x_2) \\ \vdots \\ \mu(x_n) \end{bmatrix}, \begin{bmatrix} k(x_1,x_1) & k(x_1,x_2) & \ldots & k(x_1,x_n) \\ k(x_2,x_1) & k(x_2,x_2) & \ldots & k(x_2,x_n) \\ \vdots & \vdots & \ddots & \vdots \\ k(x_n,x_1) & k(x_n,x_2) & \ldots & k(x_n,x_n) \end{bmatrix} \right)$$

One important question to ask about GP models is how they relate to more traditional parametric models. It turns out that, if the covariance function k can be expressed as an inner product in some feature space,[2] then performing GP regression is equivalent to performing linear regression on the features of this feature space. These two regression problems are actually dual representations of each other — the parametric one is called the "weight space" view of regression, and the nonparametric one is called the "function space" view. [13] Mercer's theorem, a powerful result in mathematical analysis, tells us that any continuous function that is also a valid covariance function (nonnegative definite and symmetric) can be expressed as an inner product in some (possibly infinite-dimensional) feature space, so this duality between the weight space and function space views of regression holds regardless of which covariance function we choose.[9]

For our use of GP models in this paper, we will assume that the covariance function k is of the form

$$k(x,x') = \rho^2 \exp(\frac{||x-x'||^2}{2\ell^2})$$

where ρ and ℓ are free parameters. This function is called the *squared exponential kernel*, and it is commonly used in GP regression. Because this function is assumed as part of the prior distribution over f, the values of ρ and ℓ, which encode our assumptions about the magnitude and smoothness of the function, are referred to as *hyperparameters*.

2.4.3. Gaussian process regression

When we compute the Gaussian distribution that a particular GP induces on a finite set of points in $\mathcal{X}$, we are effectively computing the marginal distribution of those points under the GP. The fact that this can be computed directly from the μ and k is commonly called the *marginalization property* of GPs, which mirrors the marginalization property of Gaussian distributions.[13] This property is very convenient for performing regression with GPs. Say we make the following model assumptions about f: [3]

$$f \sim \mathcal{GP}(0,k)$$
$$y \sim \mathcal{N}(f(X), \Lambda_X)$$

[2] A *feature space* is a vector space, usually with a very high or even infinite number of dimensions, where each dimension represents a different function on $\mathcal{X}$ (called a "feature"). They are used to give linear regression methods more flexibility — instead of assuming that the hidden function is linear in x itself, we assume it is linear in a number of *functions* of x. For example, if we are trying to fit a degree-five polynomial to a set of data points, we perform linear regression with a feature space containing all monomials with degree less than five. This is commonly called polynomial regression.

[3] The prior assumption that $\mu(x) = 0$ is very common when performing GP regression, because in practice, there is rarely a good reason for us to assume anything about the mean of the function we are trying to learn. However, it is completely possible to perform GP regression with prior assumptions about the mean function. [13]

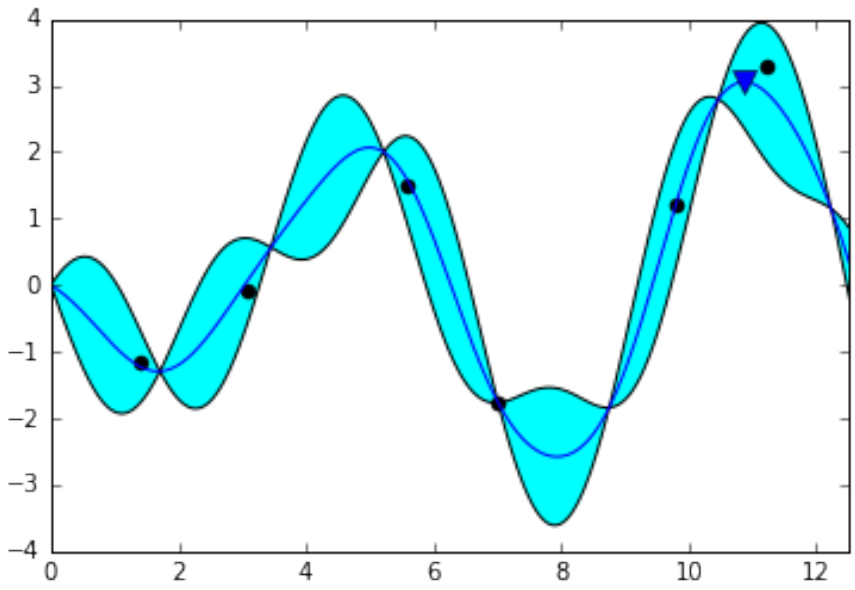

Figure 1: A Gaussian process fit to a set of sample points. The blue line is the posterior mean $\mu(x|D)$, and the shaded interval is a confidence interval of $\pm\sqrt{v(x|D)}$.

Here, X is a matrix where each row $X_{i,:}$ is the i^{th} sample point x_i, $f(X)$ is the vector of corresponding function values $f(x_i)$, y is the vector of sample values y_i, and Λ_D is the *sample noise matrix*, which is defined as [4]

$$\Lambda_D = \begin{bmatrix} \sigma^2(x_1) & 0 & \dots & 0 \\ 0 & \sigma^2(x_2) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sigma^2(x_n) \end{bmatrix}$$

For our purposes, the two quantities we want to be able to find are the *marginal mean* and the *marginal variance* of the GP at any single point $x^* \in \mathcal{X}$. Because marginalizing a GP to a finite vector of sample points results in a multivariate Gaussian distribution over the possible values of the function at those points, it follows that marginalizing a GP to a single point x^* results in a univariate Gaussian distribution over $f(x^*)$. Formally, we have

$$f(x^*) \sim \mathcal{N}(\mu(x^*), k(x^*, x^*))$$

In this way, the marginal mean and marginal variance can be thought of as functions on $\mathcal{X}$ that depend on the GP. Because the marginal mean is equal to the mean function of the GP, it is just written as $\mu(x^*)$. For the marginal variance, however, we define the function v as:

$$v(x^*) = k(x^*, x^*)$$

It is important to note that knowing the values of μ and v at every point x^* does *not* tell us everything about the GP. Specifically, these two functions tell us nothing about the covariance between the values of the function at different points on the domain, so they do not completely specify the GP.

[4] Note that by writing the likelihood of the samples y as a diagonal multivariate Gaussian distribution, we capture the assumption that the n observations are all independent. For the sake of generality, we have also assumed location-dependent sample noise. In the case where the sample noise is constant, the sample noise matrix would be equal to $\sigma^2 I$, which would make the likelihood a spherical Gaussian distribution.

The goal of GP regression in our case is to find the marginal mean and marginal variance of the posterior GP $p(f|D)$ at any point $x^* \in \mathcal{X}$. When performing regression in other contexts, we normally use Bayes' rule for this, but in the case of GP regression, this is unnecessary – a GP induces a Gaussian over any finite set of points, so we can use our GP prior to compute the joint distribution of the vector $(y, f(x^*))$, and then condition this on our observed values of y to get $p(f(x^*)|D)$. Note that this posterior distribution depends on the values of X and y, and the kernel function k. Because, in this paper, we assume the squared exponential form for k, its value is completely determined by the values we choose for the hyperparameters ρ and ℓ. [5] Because closed-form equations exist for conditioning a mutivariate Gaussian distribution, all of this can be done analytically.[13]

From our model assumptions and the definition of a GP, we can derive the following form for the joint distribution of our sample values y and the value of f at x^*, given the prior covariance function k:

$$\begin{bmatrix} y \\ f(x^*) \end{bmatrix} \sim \left(\begin{bmatrix} K(X,X) + \Lambda_D & K(X,x^*) \\ K(X,x^*)^T & k(x^*,x^*) \end{bmatrix} \right)$$

Here, we use $K(X,X')$ to denote the cross-covariance matrix of the two matrices X and X':

$$K(X,X')_{ij} = k(X_{i,:}, X_{j,:})$$

Using the conditioning formula for a multivariate Gaussian distribution[13], we can then compute the posterior distribution over $f(x^*)$: [6]

$$f(x^*)|D \sim \mathcal{N}(\mu(x^*|D), v(x^*|D)), \text{ where}$$

$$\mu(x^*|D) = K(X,x^*)^T \left[K(X,X) + \Lambda_D\right]^{-1} y \tag{1}$$

$$v(x^*|D) = k(x^*,x^*) - K(X,x^*)^T \left[K(X,X) + \Lambda_D\right]^{-1} K(X,x^*) \tag{2}$$

Because the marginal distribution at any point x is Gaussian, the MAP estimate of $f(x)$ is equal to $\mu(x|D)$. The marginal variance $v(x)$ gives us a measure of how confident we are about this estimate.

The advantage of GP regression is that it allows us to perform regression in a computationally straightforward way without making strong assumptions about the nature of the hidden function. The covariance function also provides a very easy way to encode assumptions about the hidden function's smoothness and relative magnitude. This is ideal for modeling black-box functions.

2.5. Active Learning

In most supervised learning settings, we usually assume that the learning algorithm has no control over the input data. The algorithm is given a set of sample points and their corresponding values, and returns an estimate of the function based on this data. In some scenarios, however, the

[5] When performing GP regression, there are a number of different ways to set the values of the kernel hyperparameters based on the observed data. One option is simply to compute a ML estimate of the hyperparameters based on the data, and then use the estimated hyperparameter values to fit the GP. Another possiblity, used by [14], is to use a fully Bayesian treatment, where we assume some prior distribution over the hyperparameters and then fit the GP by integrating over possible values of the hyperparameters

[6] Slightly more general equations also exist that allow us to compute the posterior distribution induced by the prior and data over a set $(x_1^*, x_2^*, ...x_k^*)$ of unobserved points. Because this is a multivariate Gaussian, instead of computing the marginal mean and variance, we compute the marginal mean vector and covariance matrix. These equations can be found in [13]

learning algorithm can choose the points to sample the function at. This is called *active learning*, or, alternatively, *experimental design*. The term experimental design comes from the fact that one natural application of active learning methods is the design of scientific experiments, where the experimenter actively selects which measurements to take, with the goal of obtaining an accurate estimate of the phenomena being studied (the function).

There are many different ways of posing the active learning problem, but for our purposes, we will focus on *sequential* case, where the learner selects sample points one-by-one. This case is especially interesting in the Bayesian framework because we can use the posterior distribution inferred from our previous samples to determine the location of our next sample.[7]

One strategy for choosing sample points in Bayesian experimental design, put forward by MacKay in [10], is to choose the point that, under expectation, will 'tell us the most' about the hidden function. Assuming that we are equally interested in knowing about all values of the function — i.e. that we aren't trying to learn about any specific region — a good metric for much we 'know' about the function is the differential entropy of the posterior distribution. [8] If we let D be the set of data points that we have already sampled, and let (x,y) be a potential new sample point and its potential observed value, then we can write the potential posterior distribution after taking this potential sample as $p(f|D \cup (x,y))$. MacKay then defines the *expected information gain* $G(x)$ at the point x as:

$$G(x) = \mathbb{E}_{p(f|D)}\left[H(p(f|D \cup (x,y))) - H(p(f|D))\right]$$

where $H(p)$ is the differential entropy of the distribution p. [9] Note that this expectation is taken over the current distribution $p(f|D)$ — we are using our current knowledge about f to predict how our knowledge about f might change from future samples. To select our next sample point x_{n+1}, we find the value in $\mathcal{X}$ that maximizes G:

$$x_{n+1} = \arg\max_{x \in \mathcal{X}} G(x)$$

MacKay then shows that, under a Gaussian approximation for the posterior distribution in the parameter space, the following holds:

$$\arg\max_{x \in \mathcal{X}} G(x) = \arg\max_{x \in \mathcal{X}} \frac{v(x|D)}{\sigma^2(x)}$$

Therefore, in order to pick a next sample point that maximizes expected information gain, all we need to do is find the x value that maximizes $\frac{v(x|D)}{\sigma^2(x)}$. A function that is maximized to determine the next sample point in a sequential active learning algorithm is called a *selection criterion*. We will refer to this specific selection criterion as the *MacKay criterion* and denote it as $a(x|D)$:

$$a(x|D) = \frac{v(x|D)}{\sigma^2(x)}$$

[7] An overview of different methods for Bayesian active learning is given in [2].

[8] The differential entropy of a continous probability distribution $p(x)$ is defined as $H(p) = -\int_{\mathcal{X}} p(x) \log p(x) dx$. This is a natural generalization of the Shannon entropy of a discrete distribution, which is defined as $H(p) = -\sum_{x_i \in \mathcal{X}} p(x_i) \log p(x_i)$. However, it should be noted that the differential entropy does not preserve all of the desirable properties of the discrete entropy.

[9] In MacKay's paper, he writes this definition and the subsequent proofs in terms of a parameterized hypothesis class, where the posterior distribution is over the parameter vector instead of the function itself. However, because of the correspondence between weight-space and function-space regression, his results also hold for GP models.

Intuitively, the MacKay criterion makes sense. To gain the most information about the hidden function, we should preferably sample at points with high posterior variance (these are the points where we know the least about the value of the function) and low sample variance (which makes our observed values more accurate).

3. Bayesian Optimization

3.1. Definition

Bayesian optimization is a technique that applies Bayesian learning to the problem of optimizing expensive-to-evaluate, black-box functions. The key idea behind Bayesian optimization is that we can treat black-box optimization like an active learning problem, where the goal is to use samples of the objective function to learn about the location of its maximizing point. The optimization algorithm alternates between sampling the objective function at a point in the domain and performing Gaussian process regression on the current set of samples to obtain a posterior distribution over possible objective functions, which is then used to choose the next sample point. After a pre-set number of iterations, it uses its current posterior distribution to construct an estimate of the maximizing value of the objective function.

Because the distribution over possible objective functions is computed by performing GP regression on the set of samples, we are effectively making the following assumptions about the black-box objective function f and the output y_i that we obtain for an input x_i:

$$f \sim \mathcal{GP}(0, k)$$
$$y_i \sim \mathcal{N}(f(x_i), \sigma_i^2)$$

Here k is a covariance function, which encodes our prior assumptions about the smoothness of f. Just like with regression, we refer to the case where σ_i^2 is constant for all i as the *constant sample noise* scenario, and the case where σ_i^2 is a function of x_i as the *location-dependent sample noise* scenario.

In order to choose sample points based on the posterior distribution $p(f|D)$, we define a function α, called the *acquisition function*, on the domain $\mathcal{X}$, which is parametrized by the GP distribution $p(f|D)$. By 'parametrized', we mean that $\alpha(x)$ is defined completely in terms of its argument x and values such as $\mu(x|D)$ and $v(x|D)$ that can be computed from the GP. Intuitively, the value $\alpha(x)$ tells us how "valuable" the point x is, given the current posterior distribution. To choose the next sample point, we find the point that maximizes α. Because we often have a mathematical expression for α, this can be done with standard numerical methods.

The full Bayesian optimization algorithm, expressed in pseudocode, is:

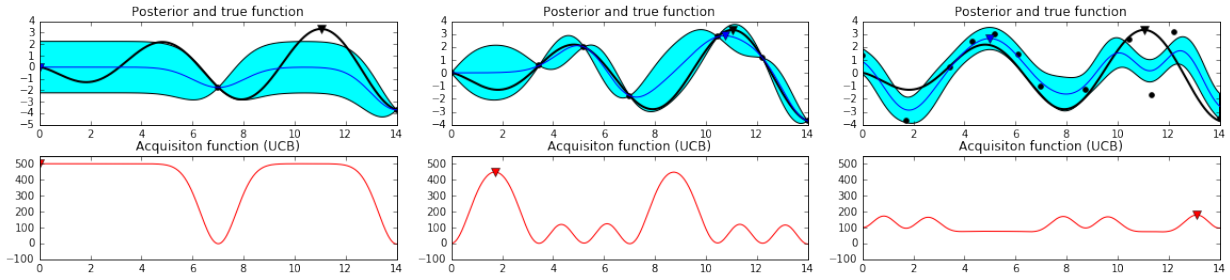

Figure 2: An example run of the Bayesian optimization algorithm after 2, 7, and 11 iterations.

Algorithm 1: Bayesian optimization algorithm

Input : Objective function `BlackBox()`, acquistion function α, number of iterations N
Output : Estimate $\hat{x}_{max}$

$x_1 \leftarrow$ `Random()`
$y_1 \leftarrow$ `BlackBox(`x_1`)`
$D_1 \leftarrow (x_1, y_1)$
$P_1 \leftarrow$ `GPRegression(`D_1`)`
for $n \leftarrow 2$ **to** N **do**
    $x_n \leftarrow \arg\max_{x \in \mathcal{X}} \alpha(x|P_{n-1})$
    $y_n \leftarrow$ `BlackBox(`x_n`)`
    $D_n \leftarrow D_{n-1} \cup \{(x_n, y_n)\}$
    $P_n \leftarrow$ `GPRegression(`D_n`)`
end
$\hat{x}_{max} \leftarrow \arg\max_{x \in \mathcal{X}}$ `PosteriorMean(`P_N`)`
return $\hat{x}_{max}$

3.2. Acquisition functions

One of the key factors in determining the success or failure of the Bayesian optimization method is our choice of acquisition function. There is no clear-cut way to use a posterior GP over possible objective functions to determine the best point to sample at next, so a number of different acquisition functions have been proposed.

One naive approach would be to use an active learning criterion, like the MacKay criterion from Section 2.5, as our acquisition function. This would select sample points in a way that would maximize our information gain about the entire objective function f. By learning about the values of f at all points on the domain, we would also eventually develop an accurate estimate of its maximizing point. However, this would be seriously inefficient in terms of the number of samples, because we would spend many of our samples learning precise estimates for values of f that are clearly not the maximum.

Good acquisition functions are effectively heuristics that allow us to focus our sampling efforts on parts of the domain that are more likely to contain the maximizing value. Once we are have sampled enough in a part of the domain to determine with a relative degree of confidence that the maximizing value does not lie there, we want to stop sampling there, and return only if our belief changes.

3.2.1. Expected Improvement

One of the most commonly used acquisition functions for Bayesian optimiztion is the *expected improvement* function (EI). The intuition behind this function, as implied by the name, is that $EI(x)$ should be equal to the amount that, given the current posterior distribution specified by $\mu(x|D)$ and $v(x|D)$, we expect y to *improve* over the maximum sample value observed so far (which we denote by y^+). Defining 'improvement' in this case is harder than it looks at first. If we were to define the improvement $I(y)$ for a given sample value as

$$I(y) = y - y^+$$

then, the expectation of this would evaluate to

$$\begin{aligned}\mathbb{E}[I(y)] &= \mathbb{E}[y - y^+] \\ &= \mathbb{E}[y] - \mathbb{E}[y^+] \\ &= \mu(x|D) - y^+\end{aligned}$$

and the point chosen by maximizing $\mathbb{E}[I(y)]$ would just be the maximizer for the posterior mean $\mu(x|D)$. To obtain a more useful quantity, we define the *improvement* $I(y)$ for a sample value y as:

$$I(y) = \max[y - y^+, 0]$$

which is equal to the difference between y and y^+ if $y \geq y^+$ and zero otherwise. Using this as the improvement metric, we define the EI aquisition function for a dataset D as: [10]

$$EI(x) = \mathbb{E}_{p(y|x,D)}[I(y)]$$

Using the properties of the Gaussian distribution, it can be shown that this integral has an analytic form.[7] If we define the dummy variable u as

$$u = \frac{\mu(x|D) - y^+}{\sqrt{v(x|D)}}$$

then the EI function can be expressed as

$$EI(x) = \sqrt{v(x|D)}\Big[u\Phi(u) + \phi(u)\Big]$$

where ϕ is the standard Gaussian PDF and Φ is the standard Gaussian CDF.

Since its introduction in a paper by Mockus et al. in 1978[11], EI has enjoyed widespread practical use. There are two major reasons for this. First, EI has done well against other acquisition functions in empirical studies of accuracy and efficiency [7]. Second, unlike the UCB acquisition function (described below), which has also performed well in these studies, EI has no parameters that require tuning, which makes it much easier to use. However, EI also demonstrates some suboptimal behavior, especially in the case of noisy observations. Specifically, it has a tendency to get the Bayesian optimization algorithm "stuck" at local optima, causing the algorithm to sample repeatedly at a non-globally optimal point, and fail to explore the rest of the domain. Also, because it computes the expected improvement of a sample over the maximum sample value observed so far, it can be seriously thrown off by one sample that is unusually high due to noise. This problem is sometimes addressed by replacing the maximum sample value y^+ with the posterior maximum μ^+, which is more robust to noise.

Another relevant fact about EI, specifically for the location-dependent sample noise scenario, is that it does not take the sample noise of a point into account. The value of EI at any given point x on the domain is determined completely by the posterior mean $\mu(x|D)$, the posterior variance $v(x|D)$, and whatever value we are trying to improve over (either y^+ or μ^+). Although the sample noise at points we have *already* sampled is relevant — this determines the GP — whether $\sigma^2(x)$ is large or small has no effect on the value of the EI function at x.

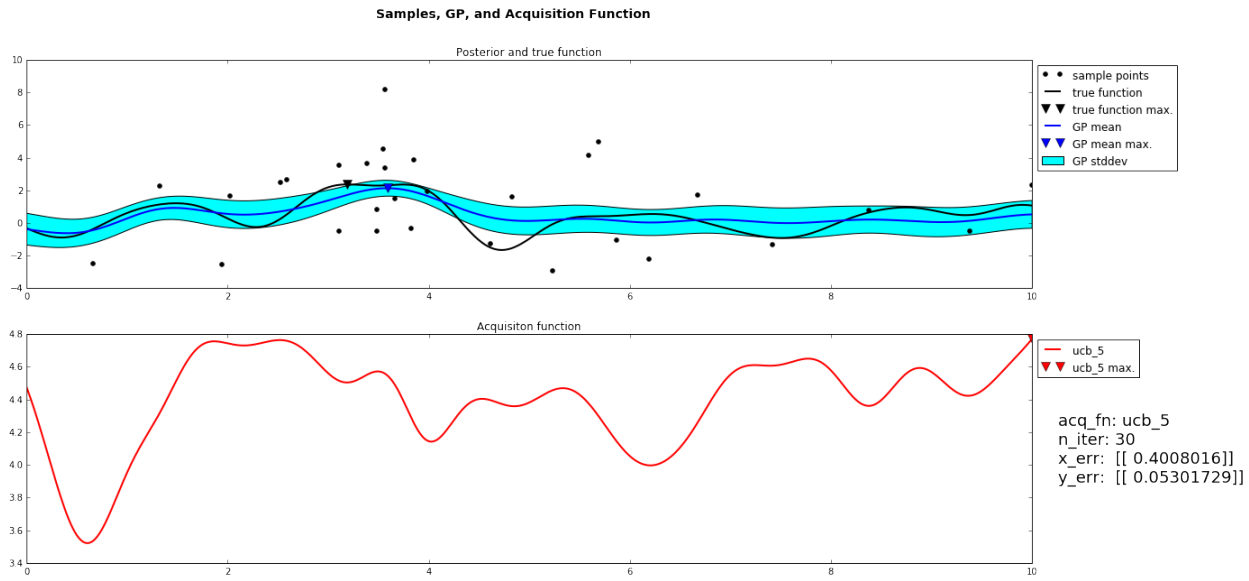

Figure 3: The Bayesian optimization algorithm using UCB ($\kappa = 5$) after 30 iterations

3.2.2. Upper Confidence Bound

Another acquisition function used for Bayesian optimization is the Upper Confidence Bound (UCB) function. This approach was first introduced in 1997 by Cox and John [3] in a non-Bayesian experimental design context, but it has since become a common acquisition function for Bayesian optimziation methods. The idea behind UCB is that, in order to find the next point to sample at, we maximize a weighted sum of the posterior mean and posterior variance — an upper confidence bound on the value of the function:

$$UCB(x) = \mu(x|D) + \kappa\sqrt{v(x|D)}$$

where κ is a parameter set by the user.

There are two ways to interpret what this function is doing. One is that, by using what is effectively an "optimistic" estimate of the function value at each point to guide our selection, we ensure that the points we neglect to sample at have a very low probability of being the maximizing point. Another is that we are using κ as a tradeoff parameter to balance "exploring" the values of the function in different parts of the domain and "exploiting" known maxima. [11] Srinivas et al.[15] [6] were able to prove theoretical bounds on the regret of Bayesian optimization with UCB under specific circumstances. Like the *EI* function, *UCB* is not sensitive to the sample noise at the point x — the only quanities it considers are the posterior mean and posterior variance.

3.2.3. Entropy Search and Predictive Entropy Search

One of the problems with both *EI* and *UCB* is that neither function has a principled mathematical justification. The "improvement" metric that *EI* maximizes in expectation has no meaning outside of the context of defining *EI*, and neither interpretation of *UCB* gives us anything more than intuition

[10] It can be show that another equivalent way to interpret the *EI* function is as a conditional expectation of the difference $y - y^+$ given that $y \geq y^+$.

[11] The tradeoff between "exploration" and "exploitation" is a common concept in optimization, most notable in the multi-armed bandit setting.

for why the function should work. Unfortunately, when defining acquisition functions, intuition can be a poor indicator what will actually work. Using the probability of improvement as an acquisition function, for example, makes a lot of intuitive sense, but can do very poorly in practice.[8] A more principled way to go about the problem, as proposed by [4] and [5] is to use the posterior GP fit to our previous samples to define a distribution $p_{\min}$ over $\mathcal{X}$ that specifies the probability that any given point x^*in $\mathcal{X}$ is the maximizing point of the function. This can be written as:

$$p_{\min}(x^*|D) = \int_{f:\mathcal{X}\to\mathbb{R}} \mathbf{1}_{\{x^*=\arg\max_{x'\in\mathcal{X}} f(x')\}} p(f|D)df$$

where $\mathbf{1}_{\{x^*=\arg\max_{x'\in\mathcal{X}} f(x')\}}$ is the indicator function for the event that x^* is the maximum of f. For our next sample point, we can pick the point that maximizes the expected decrease in differential entropy H between the current posterior distribution over possible minima and the posterior distribution that results from sampling at a point x: [12]

$$\Delta H(x) = H(p_{\min}(x^*|D) - \mathbb{E}_{p(y|x,D)}\left[H(p_{\min}(x^*|D\cup(x,y))\right]$$

By maximizing ΔH, we find the point x on the domain that will give us the most information about the location of the maximizing value of the hidden function.

The obvious problem with this approach is tractability – the distribution $p_{\min}$ is written as an integral over an infinite-dimensional space of functions, which has no closed form and is even difficult to compute by sampling methods. The Entropy Search (ES) [4] and Predictive Entropy Search (PES) [5] acquisition functions are two different schemes for approximating ΔH, both of which require a large amount of computational overhead. *ES* has been shown to outperform *EI* and *UCB* on synthetic test functions in terms of regret [4], and *PES* has been shown to outperform *EI*, *UCB*, and *ES* on synthetic test functions, as well as outperforming *UCB* and *ES* on benchmark test functions using the same metric. For the case of sample-dependent noise variance, *PES* and *ES* do take the sample noise value at a potential sample point x into account, because this affects the posterior distribution that would result from sampling at x.

[12]This approach of sampling at the point that maximizes expected decrease in differential entropy of the posterior is the exact same approach that MacKay uses for active learning in [10]. In using this as an acquisition function, we are effectively performing active learning on the distribution $p_{\min}$.

4. Problem description

4.1. Motivation

In all of the papers we have seen on Bayesian optimization and its application to real world problems, the sample noise is assumed to be constant over the entire domain. Although there are many scenarios where this assumption is valid, there are also some where it is a significant oversimplification. One example of such a scenario comes from the use of black-box optimization for tuning the parameters of machine learning algorithms, as in [14]. Imagine that our goal is to find the number of hidden units in a deep neural network that maximizes cross-validated performance on some validation set. Here, the domain is that set of possible numbers of hidden layers, and the black-box function we are trying to maximize is the cross-validated performance. To "sample" the cross-validated performance for a given number of hidden units, we train the neural net on our training data and then evaluate its performance on the validation set using cross-validation. Now, assume that we are using the backpropogation algorithm to train the neural net. If we are able to run backpropogation until convergence for all of our samples, then the constant-noise assumption makes sense here. However, if instead, due to restrictions in time or computational resources, we can only run the algorithm for, say, 10,000 iterations for each sample, then the "noise" in our sample values — the variance in the performance of the network due to the randomly initialized weight values — is not constant for the entire domain. The backpropogation algorithm converges more slowly for networks with large numbers of hidden units, so we can expect the noise to be much larger for samples of the performance with many hidden units. Therefore, the noise level is some increasing function of the number of hidden units we are using.

No work that we have seen addresses the problem of performing Bayesian optimization in scenarios like this one, where the level of the sample noise is location-dependent. One major question here is whether or not knowing the value of the sample noise variance function $\sigma^2(x)$ gives us any valuable information about where it is best to sample the hidden function during optimization. Intuitively, it seems like this additional information would be useful — all other things being equal, we would like to sample the function in places where the is less noise — but because the current literature doesn't address the location-dependent noise case, there are currently no proposals for how $\sigma^2(x)$ could be worked in to an acquisition function, and no empirical results showing that an acquisition function sensitive to $\sigma^2(x)$ would significantly outperform existing acquisition functions in scenarios with location-dependent noise.

4.2. Definition

Formally, we define the problem of black-box optimization with location-dependent sample noise as follows. Just like in the constant noise case, the optimizer is given the ability to generate a noisy sample y of the objective function f at any points x on the domain $\mathcal{X}$. However, we also assume that the optimizer has access to a function $\sigma^2(x)$, which returns the exact value of the sample noise at any point x. While sampling $f(x)$ is assumed to be very expensive, determining $\sigma^2(x)$ is not. The objective, just as in the constant-noise case, is to produce an estimate $\hat{x}_{max}$ of the true maximizing point x_{max} of f on $\mathcal{X}$, using some predetermined number of samples n.

This is just a generalization of the usual formulation of black-box optimization where, instead of being given a fixed value for the sample noise variance, the optimizer is given a function. If this function happens to be a constant value, then the two cases are identical.

4.3. Project Goals

For this project, we wanted to explore different ways of extending Bayesian optimization to scenarios with location-dependent sample noise. Our specific goals were:

1. To develop new acquisition functions that incorporate the sample noise variance, and
2. To test them against the most commonly used existing acquisition functions in scenarios where the sample noise variance at any point is a known function of the point's location

5. Noise-sensitive acquisition functions

5.1. Overview

We were able to develop two noise-sensitive acquisition functions that perform effectively in experimental settings. The basic idea behind both of these functions is the same: noise-sensitive acquisition functions should cause the Bayesian optimization algorithm to preferentially sample at points with lower sample noise, because these points give us more information about the objective function. However, the two approaches differ in how they attempt to strike a balance between the different factors that contribute to the overall value of a particular sample point for finding the maximum of f.

5.2. The UCB2 acquisition function

The first approach we considered for creating a noise-dependent acquisition function was to develop a rough measure of the "exploration" value of a point that takes the sample noise into account, and then create a variant of the UCB function that where the old exploration term — the variance of the posterior distribution — is replaced with this new term. One natural idea for this is to use the difference between the current posterior variance at a potential sample point x and the posterior variance of the GP that would result from sampling at x. This is a better measure of exploration for the location-dependent noise case because, instead of just quantifying the amount of uncertainty that exists about the function's value at x, it quantifies the amount of this uncertainty that would be eliminated by sampling at x.

One great thing about this quantity is that, when performing GP regression, the posterior variance of the GP depends only on the sample points and their sample noise levels, not on the sample values. This can be seen in the GP regression equation for posterior variance, which does not depend on the vector of observed values. This means that the posterior variance $v(x|D \cup \{(x^*,y^*)\})$ of our GP after obtaining some sample (x^*,y^*) does not depend on the value of y^*, and can be written as $v(x|D \cup \{x^*\})$. This allows us to compute how sampling at a point x^* will affect the posterior variance without needing to actually sample there.

The UCB function (Section 3.2.2) is defined as:

$$UCB(x) = \mu(x|D) + \kappa\sqrt{v(x|D)}$$

By replacing the noise-agnostic exploration term $v(x|D)$ with the noise-sensitive exploration term $v(x|D) - v(x|D \cup x)$, we can define our modified version of UCB, which we will call UCB2:

$$UCB2(x) = \mu(x|D) + \kappa\sqrt{v(x|D) - v(x|D \cup x)} \tag{3}$$

5.3. Shrinking factor equations

In order to find a closed form for the UCB2 function, we need to find a closed form for our noise-sensitive exploration term $v(x|D) - v(x|D \cup x)$. For this purpose, we used the GP regression equations to derive the following two equations, which show the effect that sampling at a point x and observing a value y has on the posterior mean and variance of the GP at x:

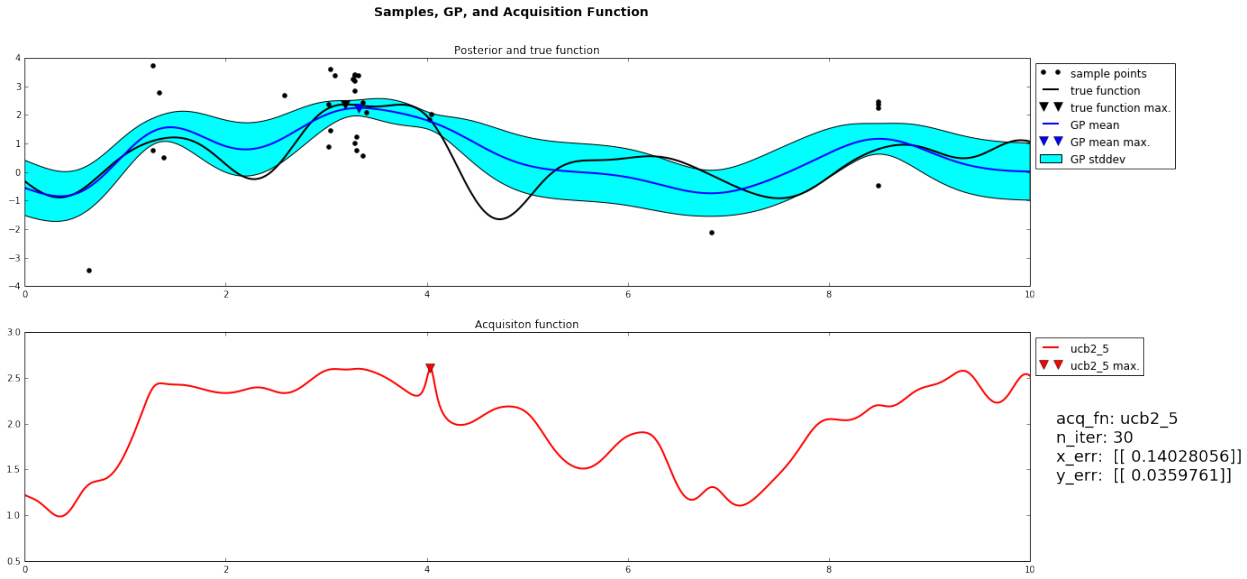

Figure 4: The Bayesian optimization algorithm using UCB2 ($\kappa = 5$) after 30 iterations

$$\mu(x|D \cup \{(x,y)\}) = \mu(x|D) + \frac{v(x|D)}{v(x|D) + \sigma^2(x)} \left[y - \mu(x|D) \right] \tag{4}$$

$$v(x|D \cup \{(x,y)\}) = v(x|D) - \frac{v(x|D)}{v(x|D) + \sigma^2(x)} v(x|D) \tag{5}$$

where $c(x|D)$, which we call the *shrinking factor*, is defined as:

$$c(x|D) = \frac{v(x|D)}{v(x|D) + \sigma^2(x)} \tag{6}$$

Intuitively, these makes sense. For an observation point x with a high level of noise, $c(x|D)$ is small, so sampling at x moves the posterior mean only slightly closer to y, and only slightly reduces the posterior variance. However, if the noise level is low, sampling moves the posterior mean very close to y and reduces the posterior variance substantially. If the noise level is zero (noiseless interpolation), the posterior mean at x is set to y and the posterior variance is set to zero, just as we would expect. The derivation of both of these equations from the GP regression equations can be found in Appendix A.

5.4. A formula for UCB2

Using (5), we can write a closed form for the noise-sensitive exploration term $v(x|D) - v(x|D \cup x)$:

$$\begin{aligned} v(x|D) - v(x|D \cup x) &= v(x|D) - \left[v(x|D) - c(x|D) v(x|D) \right] \\ &= c(x|D) v(x|D) \\ &= \frac{v^2(x|D)}{v(x|D) + \sigma^2(x)} \end{aligned}$$

Plugging this in to our definition of UCB2 in (3), we get

$$\begin{aligned} UCB2(x) &= \mu(x|D) + \kappa\sqrt{\frac{v^2(x|D)}{v(x|D)+\sigma^2(x)}} \\ &= \mu(x|D) + \kappa\frac{v(x|D)}{\sqrt{v(x|D)+\sigma^2(x)}} \end{aligned}$$

Therefore, the formula for UCB2 is:

$$UCB2(x) = \mu(x|D) + \kappa\frac{v(x|D)}{\sqrt{v(x|D)+\sigma^2(x)}} \tag{7}$$

5.5. Another approach: Active learning on upper quantile of function

One thing we observed when studying the performance of different acquisition functions is that the best acquisition functions tend to sample most heavily in parts of the domain where the values of f are relatively large. This observation leads to the idea that if, at the end of the Bayesian optimization routine, the sample points and their corresponding values are plotted on the same graph as the hidden function, the samples generated by an effective acquisition function should be dense in the regions near the modes of the function and sparse in the regions with lower values.

One way to accomplish this desired behavior is to write an acquisition function that tries to approxmiate active learning on some upper quantile of the function. Obviously, we can't know which parts of the domain have objective function values in, say, the top tenth of the interval containing all objective function values — if we knew this, we could just disregard all other parts of the domain – and even trying to determine the probability that the value of the function any given point on the domain is in this upper quantile would require wrangling with a distribution just as intractable as the distribution used by the ES and PES functions from Section 3.2.3. However, by combining an active learning criterion with our knowledge from the posterior GP, we can create a heuristic that approximates this ideal behavior.

5.6. The Expected Gain acquisition function

The Expected Gain (EG) function is defined similarly to the EI function. First, we define the *gain* $G(x)$ for a point $x \in \mathcal{X}$ as:

$$G(x|D) = \begin{cases} a(x|D) & \text{if } f(x) \geq \mu^+(D) \\ 0 & \text{otherwise} \end{cases}$$

where $a(x|D) = \frac{v(x|D)}{\sigma^2(x)}$ is the MacKay criterion from Section 2.5 and $\mu^+(D)$ is the maximum value of the posterior mean function on $\mathcal{X}$. We then define the EG function as:

$$EG(x) = \mathbb{E}_{p(f|D)}\left[G(x)\right] \tag{8}$$

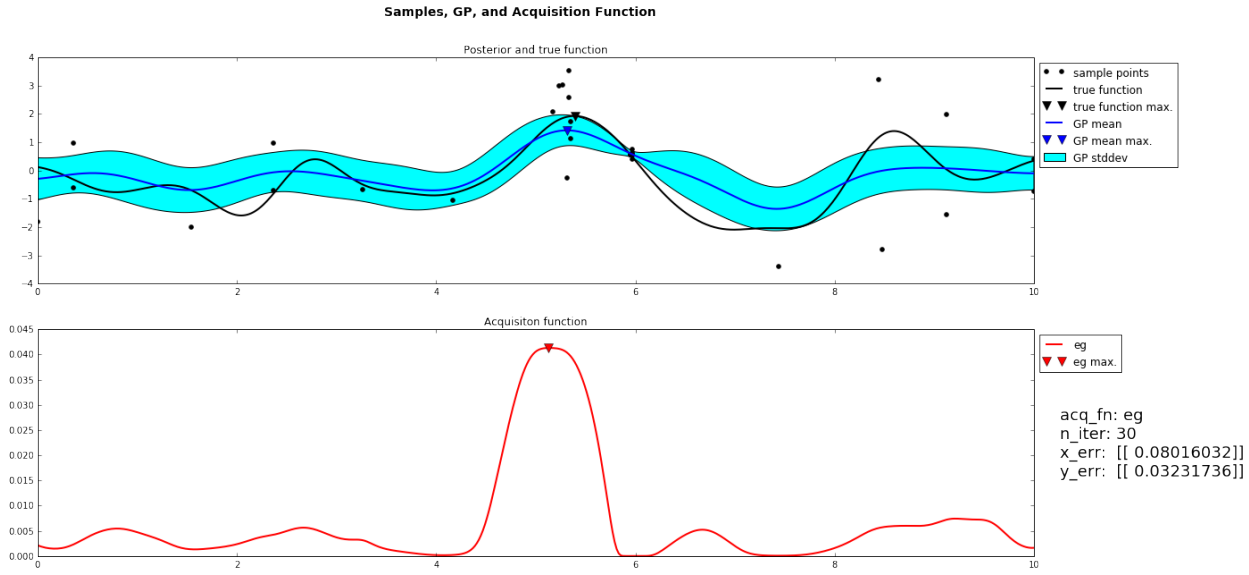

Figure 5: The Bayesian optimization algorithm using EG after 30 iterations

From this definition, we can derive a formula for EG:

$$\begin{aligned} EG(x) &= \mathbb{E}_{p(f|D)}\left[G(x)\right] \\ &= \mathbb{E}_{p(f|D)}\left[G(x)|f(x) \geq \mu^+(D)\right] \cdot p(f(x) \geq \mu^+(D)|D) \\ &\quad + \mathbb{E}_{p(f|D)}\left[G(x)|f(x) < \mu^+(D)\right] \cdot p(f(x) < \mu^+(D)|D) \\ &= a(x|D) \cdot p(f(x) \geq \mu^+(D)|D) + 0 \cdot p(f(x) < \mu^+(D)|D) \\ &= a(x|D) \cdot p(f(x) \geq \mu^+(D)|D) \end{aligned}$$

Becuase $f(x|D)$ has a Gaussian distribution with mean $\mu(x|D)$ and variance $v(x|D)$, we have

$$p(f(x) \geq \mu^+(D)|D) = \Phi(\frac{\mu(x|D) - \mu^+(D)}{v(x|D)})$$

Plugging this in to the equation above, we get the formula:

$$EG(x) = a(x|D) \cdot \Phi(\frac{\mu(x|D) - \mu^+(D)}{v(x|D)}) \tag{9}$$

where Φ is the CDF of a standard Gaussian distribution. Just like the EI function, EG has no parameters that need to be tuned by the user, which is very useful in practice.

6. Evaluation

6.1. Software

In order to empirically test our new noise-sensitive acquisition functions and compare them against the existing acquisition functions, we first needed to develop a codebase for running the experiments. Although several open-source packages exist for performing Bayesian optimization, none of the ones that we were able to find had a built-in way to define noise-sensitive acquisition functions or were structured in a way that would allow us to make that modification without significant effort. Because of this, we ultimately decided to create a codebase from scratch that lets us run the Bayesian optimization algorithm on synthetic objective functions with location-dependent sample noise using different acquisition functions. This library also has functions for computing performance metrics for different runs of the algorithm and plotting their results. The code was written in Python (using the NumPy library for numerical computations), and its general structure was based on the source code from the open-source package Spearmint, developed by the authors of [14]. Developing and testing this codebase was a significant part of this project.

6.2. Experimental setup

For our experiments, we mostly followed the setup used by [5] to test their PES acquisition function on synthetic objective functions. Because the point of our experiments was to compare the performance of UCB2 and EG against noise-agnostic acquistion functions, we also created synthetic noise variance functions. All components of the experiment — creating synthetic objective and noise functions, maximizing the acquisition function and posterior mean during the Bayesian optimization algorithm, and choosing sample points — were performed on a 500-point grid of evenly-spaced points on the real interval $[0, 10]$, which we refer to as the *experiment grid*.

The objective functions were generated from a GP on the experiment grid.[13] This GP had a mean function equal to zero at all points, used the squared exponential kernel function, and had hyperparameter values $\rho = 1$ and $\ell = 0.5$. We generated 1000 objective functions for our test dataset.

For each objective function, we created four different noise variance functions. The first of these functions was just a constant-valued function, where the noise variance is equal to 0.3 at every point on the domain. The next three noise functions were created in the following way. First, we generated a function from a GP on the experiment grid. Then, we shifted this function so that its minimum value was at some pre-specified value (this made sure the function was always positive). For all three functions, the GP used a squared exponential kernel with $\ell = 0.25$. The two parameters that we varied were the ρ hyperparameter in the kernel of the GP and the minimum value. The first location-dependent noise function was generated with $\rho = 1.0$ and $\texttt{min\_val} = 0.1$; the second was generated with $\rho = 2.0$ and $\texttt{min\_val} = 0.2$, and the third was generated with $\rho = 3.0$ and $\texttt{min\_val} = 0.2$. The purpose of varying these parameters was to change the overall level of distortion present in our samples of the objective function. These four noise functions allowed us to create four test sets: the first consisted of the objective functions paired with constant noise functions, and the next three consisted of the objective functions paired with each of the different

[13] In this context, where everything is being done on a grid, 'generating a function from a GP' means sampling a vector from the multivariate Gaussian distribution that the GP induces on the grid.

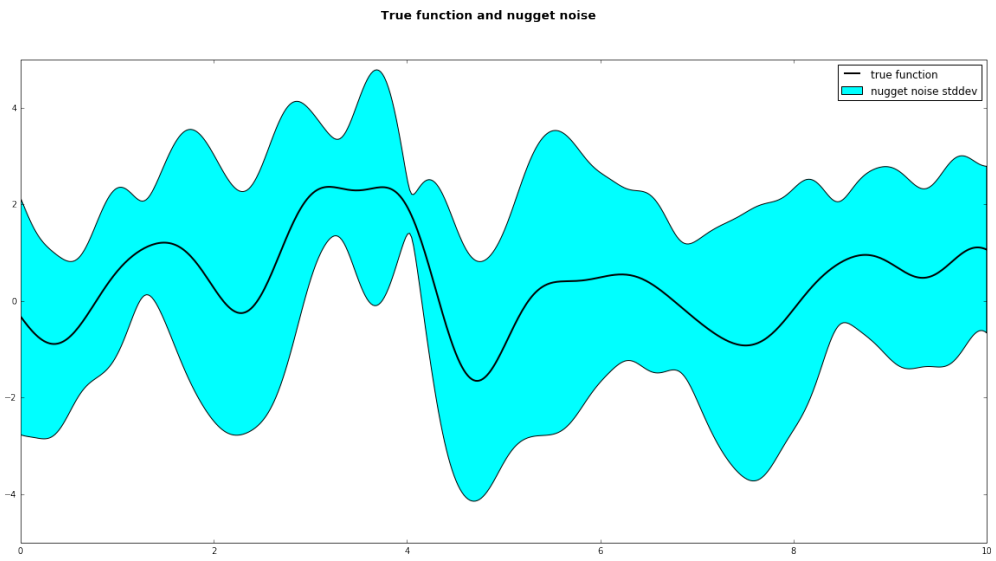

Figure 6: An example of one of the synthetic objective functions generated for the experiment, along with its corresponding noise variance function

classes of location-dependent noise functions. Each of these test sets contained the same 1000 objective functions.

In order to quantify performance, we used the metric of Immediate Regret (IR) also used by [5]. If the Bayesian optimization algorithm is run for n iterations, then the IR at iteration n is defined as:

$$IR_n = |f(x_{max}) - f(\hat{x}_{max})|$$

where $x_{max} = \arg\max_{x \in \mathcal{X}} f(x)$ is the true maximizing point of the function, and $\hat{x}_{max} = \arg\max_{x \in \mathcal{X}} \mu(x|D_n)$ is the estimate of this maximizing point returned by the algorithm after iteration n.

The acquisition functions that we tested were: the MacKay active learning criterion (as a control), UCB (with $\kappa = 5$), EI, Modified EI (which computes expected improvement over the posterior mean maximum, not the maximum sample value), UCB2 (with $\kappa = 5$), and EG. We ran the Bayesian optimization algorithm, using each of these functions, on each of the four test sets for 50 iterations, recording the IR values after every iteration. When performing GP regression, the algorithm used the "true" distribution — the GP that was used to generate the objective functions — as its prior. This part of the experiment is not very realistic — when optimizing a real black-box function, we generally do not know the best values to use for the hyperparameters — but because the purpose of this experiment was to compare two classes of acquisition functions, not to test the efficacy of the Bayesian optimization algorithm itself, we thought it was best to remove this potentially complicating factor from the experiments.

6.3. Results

The four plots in Figure 7 show, for each acquisition function, the base-10 logarithm of the median IR value across all trials, at each iteration. Although plots of the mean IR look similar, the median IR was chosen by [5] because the distribution of IR values across trials tends to be heavy-tailed, and the median better represents the center of the data.

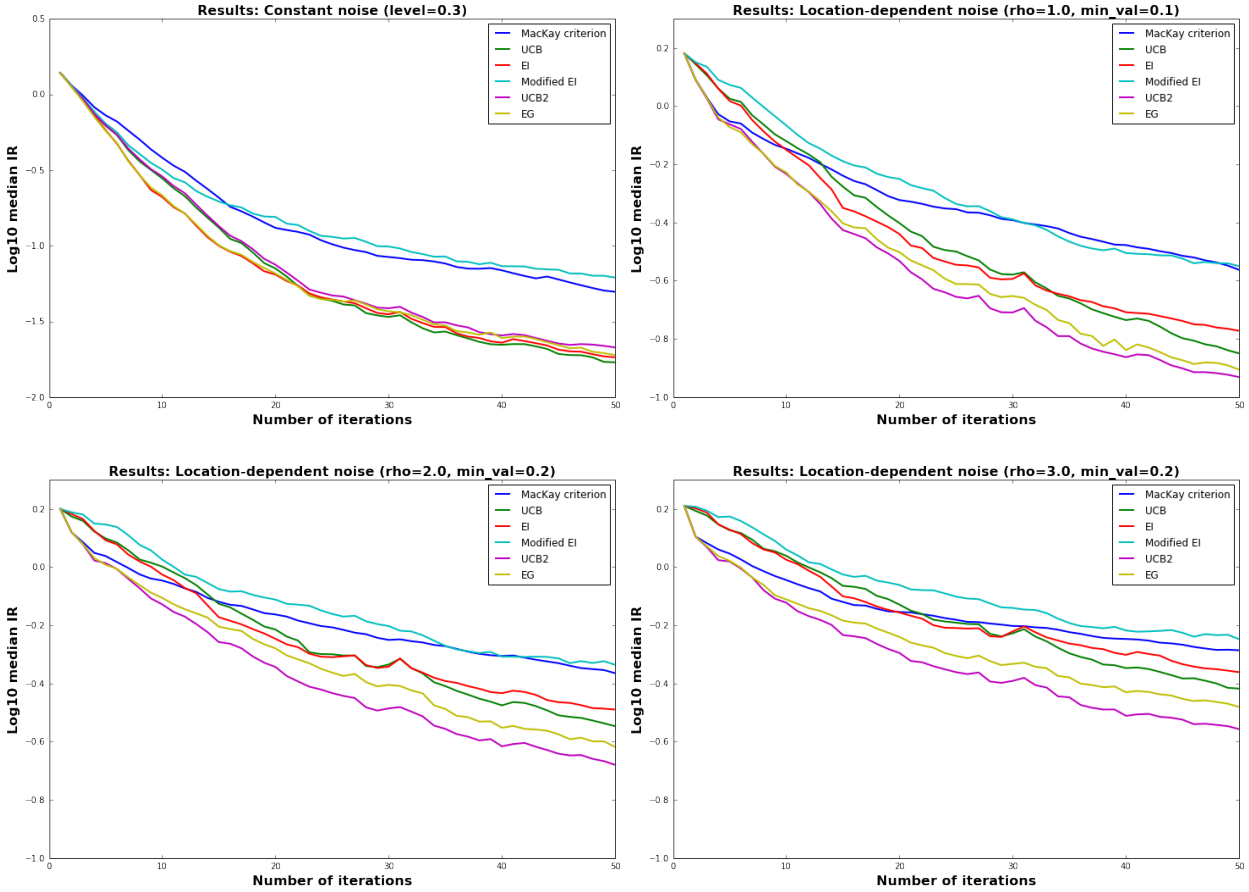

Figure 7: Log-median IR of acquisition functions on test datasets

For the test case with constant noise, the UCB, EI, UCB2, and EG acquisition functions perform equally well, with EI and EG having a small advantage for iterations 5-20. These functions all perform well for objective functions on our domain, in synthetic, low observation variance experiments. The MacKay criterion and the modified EI function perform significantly worse. Most of this is what we would expect: because the sample noise is not location-dependent, there is no reason why the noise-sensitive functions should perform better than the noise-agnostic ones, and all acquisition functions should generally perform better than the MacKay criterion. One thing, however, stands out: the modified EI function, which we expected to be more noise-resistant than EI, performs significantly worse than EI, and slightly worse than the MacKay criterion. This turns out to be the case for all four tests, regardless of the noise function used.

One reason for modified EI's underperformance could be the fact that we are giving the Bayesian optimization algorithm the "true" prior GP from which the objective functions are drawn. In practice, we often don't have access to this information, and the kernel hyperparameters for the prior have to be estimated during the regression stage of the Bayesian optimization algorithm, as explained in Section 2.4.3. If unusually extreme sample values cause our estimates of these hyperparameters to be way off, then the EI function could get "stuck" on local minima more easily — for example, if the prior underestimates the level of uncertainty about the function values, then this could discourage the EI function from exploring the domain. In this case, modified EI might perform better relative

to standard EI.

For all three test cases with location-dependent sample noise, the noise-sensitive acquisition functions UCB2 and EG have smaller median IR than the noise-agnostic functions UCB and EI for every single iteration after iteration 5. UCB2 also has slightly lower median IR than EG for the vast majority of iterations. Although one might expect the noise-sensitive functions to outperform the noise-agnostic ones by a larger amount for the cases with larger noise levels, the difference in median IR remains relatively the same for all three of the test cases with location-dependent noise.

7. Conclusion

The experiments we performed suggest that, under circumstances where the sample noise varies as a known function of the sample point, our noise-sensitive acquisition functions UCB2 and EG consistently outperform the commonly used UCB and EI functions. Although these results are very encouraging, they are also somewhat preliminary — there are still many more situations in which we would like to test the effectiveness of UCB2 and EG. For one, we have yet to perform any experiments with benchmark objective functions, like the Branin-Hoo function, or real-world black-box optimization problems, like those examined in [14]. We also haven't tested UCB2 and EG on domains with dimension larger than one, or in situations where gridding is not computationally feasible. These are all good directions for future work. Utimately, the only way that a heuristic like an acquisition function can be shown to be effective is through empirical testing and practical use. However, the very stark nature of our results gives us reason to believe that UCB2 and EG are very promising solutions for the location-dependent case of black-box optimization, and should be pursued further.

References

[1] D. Bernstein, *Matrix Mathematics*. Princeton University Press, 2005, p. 44.

[2] K. Chaloner and I. Verdinelli, "Bayesian experimental design: A review," *Statistical Science*, pp. 273–304, August 1995.

[3] D. D. Cox and S. John, "Sdo: A statistical method for global optimization," in *Multidisciplinary design optimization: state of the art*, 1997, pp. 315–329.

[4] P. Hennig and C. J. Schuler, "Entropy search for information-efficient global optimization," *The Journal of Machine Learning Research*, vol. 13, no. 1, pp. 1809–1837, 2012.

[5] J. M. Hernández-Lobato, M. W. Hoffman, and Z. Ghahramani, "Predictive entropy search for efficient global optimization of black-box functions," *Advances in Neural Information Processing Systems*, pp. 918–926, 2014.

[6] *Gaussian process optimization in the bandit setting: No regret and experimental design*. Haifa, Israel: International Conference on Machine Learning, 2010.

[7] D. R. Jones, "A taxonomy of global optimization methods based on response surfaces," *Journal of global optimization*, vol. 21, no. 4, pp. 345–383, 2001.

[8] D. R. Jones, M. Schonlau, and W. J. Welch, "Efficient global optimization of expensive black-box functions," *Journal of Global optimization*, vol. 13, no. 4, pp. 455–492, 1998.

[9] V. Korotkov. Mercer theorem (encyclopedia of mathematics).

[10] D. J. C. MacKay, "Information-based objective functions for active data selection," *Neural Computation*, vol. 4, pp. 590–604, 1992.

[11] J. Mockus, V. Tiesis, and A. Zilinskas, "The application of bayesian methods for seeking the extremum," *Towards global optimization*, vol. 2, no. 117-129, p. 2, 1978.

[12] K. P. Murphy, *Machine Learning*. MIT Press, 2012.

[13] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[14] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," *Advances in neural information processing systems*, pp. 2951–2959, 2012.

[15] N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger, "Gaussian process optimization in the bandit setting: No regret and experimental design," arXiv preprint arXiv:0912.3995, 2009.

A. Derivation of shrinking factor equations

The shrinking factor equations referenced in (???) are:

$$\mu(x|D\cup\{(x,y)\}) = \mu(x|D) + \frac{v(x|D)}{v(x|D)+\sigma^2(x)}\left[y-\mu(x|D)\right] \tag{10}$$

$$v(x|D\cup\{(x,y)\}) = v(x|D) - \frac{v(x|D)}{v(x|D)+\sigma^2(x)}v(x|D) \tag{11}$$

We will derive these equations directly from the GP regression equations, using a formula for computing the blockwise inverse of a matrix.

A.1. GP regression equations

For standard GP regression, we make the following model assumptions about f:

$$f \sim \mathcal{GP}(0,k)$$
$$y(x) \sim \mathcal{N}(f(x),\sigma^2(x))$$

From this, it is possible to derive the following equations for the predictive mean and variance functions:

$$\mu(x|D) = k_D(x)^T\left[K_D+\Lambda_D\right]^{-1}D_y \tag{12}$$

$$v(x|D) = k(x,x) - k_D(x)^T\left[K_D+\Lambda_D\right]^{-1}k_D(x) \tag{13}$$

A.2. Blockwise Matrix Inversion

In general, the inverse of a block matrix can be computed from the inverses of the individual blocks using the following formula:

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} = \begin{bmatrix} A^{-1}+A^{-1}B(D-CA^{-1}B)^{-1}CA^{-1} & -A^{-1}B(D-CA^{-1}B)^{-1} \\ -(D-CA^{-1}B)^{-1}CA^{-1} & (D-CA^{-1}B)^{-1} \end{bmatrix} \tag{14}$$

which holds if and only if the matrices A and $(D-CA^{-1}B)$ are invertible. [1]

For our purposes, we will only need to use a special case of this formula where B is a column vector (which we will write as b), C is equal to b^T, and D is a 1x1 matrix (which we will write as c). Performing these substitutions on the above formula gives us:

$$\begin{bmatrix} A & b \\ b^T & c \end{bmatrix}^{-1} = \begin{bmatrix} A^{-1}+\frac{1}{z}A^{-1}bb^TA^{-1} & -\frac{1}{z}A^{-1}b \\ -\frac{1}{z}b^TA^{-1} & \frac{1}{z} \end{bmatrix} \tag{15}$$

where $z = c - b^TAb$.

A.3. Derivation of Equation 10

First, let $D' = D\cup\{(x,y)\}$ be the set of data points used to fit the GP after sampling at x. We can write the GP regression equation for the predictive mean function, using this dataset in place of the pre-sampling dataset D:

$$\mu(x|D') = k_{D'}(x)^T\left[K_{D'}+\Lambda_{D'}\right]^{-1}D'_y \tag{16}$$

Because D' is just D with a single point added, we have:

$$k_{D'}(x) = \begin{bmatrix} k_D(x) \\ k(x,x) \end{bmatrix} \tag{17}$$

$$K_{D'} = \begin{bmatrix} K_D & k_D(x) \\ k_D(x)^T & k(x,x) \end{bmatrix} \tag{18}$$

$$\Lambda_{D'} = \begin{bmatrix} \Lambda_D & 0 \\ 0 & \sigma^2(x) \end{bmatrix} \tag{19}$$

$$D'_y = \begin{bmatrix} D_y \\ y \end{bmatrix} \tag{20}$$

The first part of the mean equation that we can compute is $\left[K_{D'} + \Lambda_{D'}\right]^{-1}$. Using equations 18 and 19, we have

$$K_{D'} + \Lambda_{D'} = \begin{bmatrix} K_D + \Lambda_D & k_D(x) \\ k_D(x)^T & k(x,x) + \sigma^2(x) \end{bmatrix} \tag{21}$$

We can use equation 15 to compute the inverse of this matrix. Assigning the block variables from the formula to the blocks of $K_{D'} + \Lambda_{D'}$, we have:

$$\begin{aligned} A &= K_D + \Lambda_D \\ b &= k_D(x) \\ c &= k(x,x) + \sigma^2(x) \end{aligned}$$

We can now compute the inverse matrix block-by-block, according to 15:

$$\begin{aligned} \left[K_{D'} + \Lambda_{D'}\right]^{-1}_{(1,1)} &= A^{-1} + \frac{1}{z} A^{-1} b b^T A^{-1} \\ &= \left[K_D + \Lambda_D\right]^{-1} + \frac{1}{z}\left[K_D + \Lambda_D\right]^{-1} k_D(x) k_D(x)^T \left[K_D + \Lambda_D\right]^{-1} \end{aligned} \tag{22}$$

$$\begin{aligned} \left[K_{D'} + \Lambda_{D'}\right]^{-1}_{(1,2)} &= -\frac{1}{z} A^{-1} b \\ &= -\frac{1}{z}\left[K_D + \Lambda_D\right]^{-1} k_D(x) \end{aligned} \tag{23}$$

$$\begin{aligned} \left[K_{D'} + \Lambda_{D'}\right]^{-1}_{(2,1)} &= -\frac{1}{z} b^T A^{-1} \\ &= -\frac{1}{z} k_D(x)^T \left[K_D + \Lambda_D\right]^{-1} \end{aligned} \tag{24}$$

$$\left[K_{D'} + \Lambda_{D'}\right]^{-1}_{(2,2)} = \frac{1}{z} \tag{25}$$

For the dummy variable z, we have

$$\begin{aligned} z &= c - b^T A b \\ &= \sigma^2(x) + k(x,x) - k_D(x)^T \left[K_D + \Lambda_D\right]^{-1} k_D(x) \end{aligned} \tag{26}$$

Substituing equation 13 (the GP regression equation for predictive variance) into this equation gives us

$$z = \sigma^2(x) + v(x|D) \tag{27}$$

Our original equation for $\mu(x|D')$ now looks like

$$\mu(x|D') = \begin{bmatrix} k_D(x) \\ k(x,x) \end{bmatrix}^T \begin{bmatrix} \left[K_{D'} + \Lambda_{D'}\right]^{-1}_{(1,1)} & \left[K_{D'} + \Lambda_{D'}\right]^{-1}_{(1,2)} \\ \left[K_{D'} + \Lambda_{D'}\right]^{-1}_{(2,1)} & \left[K_{D'} + \Lambda_{D'}\right]^{-1}_{(2,2)} \end{bmatrix} \begin{bmatrix} D_y \\ y \end{bmatrix} \tag{28}$$

Multiplying this out by blocks, we get

$$\begin{aligned} \mu(x|D') = & k_D(x)^T \left[K_{D'} + \Lambda_{D'}\right]^{-1}_{(1,1)} D_y \\ & + k_D(x)^T \left[K_{D'} + \Lambda_{D'}\right]^{-1}_{(1,2)} y \\ & + k(x,x) \left[K_{D'} + \Lambda_{D'}\right]^{-1}_{(2,1)} D_y \\ & + k(x,x) \left[K_{D'} + \Lambda_{D'}\right]^{-1}_{(2,2)} y \end{aligned} \tag{29}$$

We can compute this sum term-by-term. For the first term,

$$\begin{aligned} & k_D(x)^T \left[K_{D'} + \Lambda_{D'}\right]^{-1}_{(1,1)} D_y \\ & \quad = k_D(x)^T \left[\left[K_D + \Lambda_D\right]^{-1} + \frac{1}{z} \left[K_D + \Lambda_D\right]^{-1} k_D(x) k_D(x)^T \left[K_D + \Lambda_D\right]^{-1} \right] D_y \\ & \quad = k_D(x)^T \left[K_D + \Lambda_D\right]^{-1} D_y \\ & \qquad\qquad + \frac{1}{z} \left[k_D(x)^T \left[K_D + \Lambda_D\right]^{-1} k_D(x) \right] \left[k_D(x)^T \left[K_D + \Lambda_D\right]^{-1} D_y \right] \\ & \quad = \mu(x|D) + \frac{1}{z} \left[k(x,x) - v(x|D)\right] \mu(x|D) \end{aligned} \tag{30}$$

where the last step above comes from substituing in the GP regression equations 12 and 13. We can also use these substitutions for the next three terms:

$$\begin{aligned} k_D(x)^T \left[K_{D'} + \Lambda_{D'}\right]^{-1}_{(1,2)} y &= k_D(x)^T \left[-\frac{1}{z} \left[K_D + \Lambda_D\right]^{-1} k_D(x) \right] y \\ &= -\frac{1}{z} \left[k_D(x)^T \left[K_D + \Lambda_D\right]^{-1} k_D(x) \right] y \\ &= -\frac{y}{z} \left[k(x,x) - v(x|D)\right] \end{aligned} \tag{31}$$

$$\begin{aligned} k(x,x) \left[K_{D'} + \Lambda_{D'}\right]^{-1}_{(2,1)} D_y &= k(x,x) \left[-\frac{1}{z} k_D(x)^T \left[K_D + \Lambda_D\right]^{-1} \right] D_y \\ &= -\frac{k(x,x)}{z} \left[k_D(x)^T \left[K_D + \Lambda_D\right]^{-1} D_y \right] \\ &= -\frac{k(x,x)}{z} \mu(x|D) \end{aligned} \tag{32}$$

$$\begin{aligned} k(x,x) \left[K_{D'} + \Lambda_{D'}\right]^{-1}_{(2,2)} y &= k(x,x) \left[\frac{1}{z}\right] y \\ &= \frac{k(x,x)}{z} y \end{aligned} \tag{33}$$

Plugging all of these terms into 29, we have:

$$\begin{aligned}
\mu(x|D') &= \mu(x|D) + \frac{1}{z}\left[k(x,x) - v(x|D)\right]\mu(x|D) \\
&\quad - \frac{y}{z}\left[k(x,x) - v(x|D)\right] \\
&\quad - \frac{k(x,x)}{z}\mu(x|D) \\
&\quad + \frac{k(x,x)}{z}y \\
&= \mu(x|D) + \frac{k(x,x)}{z}\mu(x|D) - \frac{v(x|D)}{z}\mu(x|D) \\
&\quad - \frac{k(x,x)}{z}y + \frac{v(x|D)}{z}y - \frac{k(x,x)}{z}\mu(x|D) + \frac{k(x,x)}{z}y \\
&= \mu(x|D) + \frac{v(x|D)}{z}y - \frac{v(x|D)}{z}\mu(x|D) \\
&\quad + \frac{k(x,x)}{z}\mu(x|D) - \frac{k(x,x)}{z}\mu(x|D) + \frac{k(x,x)}{z}y - \frac{k(x,x)}{z}y \\
&= \mu(x|D) + \frac{v(x|D)}{z}y - \frac{v(x|D)}{z}\mu(x|D) \\
&= \mu(x|D) + \frac{v(x|D)}{z}(y - \mu(x|D))
\end{aligned} \tag{34}$$

Because $z = \sigma^2(x) + v(x|D)$ (equation 27), this can be written as

$$\mu(x|D') = \mu(x|D) + \frac{v(x|D)}{\sigma^2(x) + v(x|D)}(y - \mu(x|D))$$

which gives us equation 10. □

A.4. Derivation of Equation 11

This derivation follows the derivation of 10. First, we can write the GP regression equation for the predictive variance function, using D' in place of D:

$$v(x|D) = k(x,x) - k_{D'}^T\left[K_{D'} + \Lambda_{D'}\right]^{-1}k_{D'} \tag{35}$$

From equations 22 - 26, we know that this equation can be written as:

$$v(x|D) = k(x,x) - \begin{bmatrix} k_D(x) \\ k(x,x) \end{bmatrix}^T \begin{bmatrix} \left[K_{D'} + \Lambda_{D'}\right]^{-1}_{(1,1)} & \left[K_{D'} + \Lambda_{D'}\right]^{-1}_{(1,2)} \\ \left[K_{D'} + \Lambda_{D'}\right]^{-1}_{(2,1)} & \left[K_{D'} + \Lambda_{D'}\right]^{-1}_{(2,2)} \end{bmatrix} \begin{bmatrix} k_D(x) \\ k(x,x) \end{bmatrix} \tag{36}$$

where

$$
\begin{aligned}
\left[K_{D'}+\Lambda_{D'}\right]^{-1}_{(1,1)} &= \left[K_D+\Lambda_D\right]^{-1} + \frac{1}{z}\left[K_D+\Lambda_D\right]^{-1} k_D(x) k_D(x)^T \left[K_D+\Lambda_D\right]^{-1} \\
\left[K_{D'}+\Lambda_{D'}\right]^{-1}_{(1,2)} &= -\frac{1}{z}\left[K_D+\Lambda_D\right]^{-1} k_D(x) \\
\left[K_{D'}+\Lambda_{D'}\right]^{-1}_{(2,1)} &= -\frac{1}{z} k_D(x)^T \left[K_D+\Lambda_D\right]^{-1} \\
\left[K_{D'}+\Lambda_{D'}\right]^{-1}_{(2,2)} &= \frac{1}{z} \\
z &= \sigma^2(x) + v(x|D)
\end{aligned}
$$

Just like in 29, we can compute this matrix product block-wise:

$$
\begin{aligned}
v(x|D') = k(x,x) &- k_D(x)^T \left[K_{D'}+\Lambda_{D'}\right]^{-1}_{(1,1)} k_D(x) \\
&- k_D(x)^T \left[K_{D'}+\Lambda_{D'}\right]^{-1}_{(1,2)} k(x,x) \\
&- k(x,x) \left[K_{D'}+\Lambda_{D'}\right]^{-1}_{(2,1)} k_D(x) \\
&- k(x,x) \left[K_{D'}+\Lambda_{D'}\right]^{-1}_{(2,2)} k(x,x)
\end{aligned}
\tag{37}
$$

and the sum can be computed term-by-term, using the GP regression equations 12 and 13 For the first term:

$$
\begin{aligned}
& k_D(x)^T \left[K_{D'}+\Lambda_{D'}\right]^{-1}_{(1,1)} k_D(x) \\
&\quad = k_D(x)^T \left[\left[K_D+\Lambda_D\right]^{-1} + \frac{1}{z}\left[K_D+\Lambda_D\right]^{-1} k_D(x) k_D(x)^T \left[K_D+\Lambda_D\right]^{-1} \right] k_D(x) \\
&\quad = k_D(x)^T \left[K_D+\Lambda_D\right]^{-1} k_D(x) \\
&\qquad\quad + \frac{1}{z}\left[k_D(x)^T \left[K_D+\Lambda_D\right]^{-1} k_D(x) \right] \left[k_D(x)^T \left[K_D+\Lambda_D\right]^{-1} k_D(x) \right] \\
&\quad = k(x,x) - v(x|D) + \frac{1}{z}\left[k(x,x) - v(x|D)\right]\left[k(x,x) - v(x|D)\right] \\
&\quad = k(x,x) - v(x|D) + \frac{1}{z}\left[k^2(x,x) - 2k(x,x)v(x|D) + v^2(x|D)\right]
\end{aligned}
\tag{38}
$$

For the second:

$$
\begin{aligned}
k_D(x)^T \left[K_{D'}+\Lambda_{D'}\right]^{-1}_{(1,2)} k(x,x) &= k_D(x)^T \left[-\frac{1}{z}\left[K_D+\Lambda_D\right]^{-1} k_D(x) \right] k(x,x) \\
&= -\frac{1}{z}\left[k_D(x)^T \left[K_D+\Lambda_D\right]^{-1} k_D(x) \right] k(x,x) \\
&= -\frac{1}{z}\left[k(x,x) - v(x|D)\right] k(x,x) \\
&= -\frac{1}{z}\left[k^2(x,x) - k(x,x)v(x|D)\right]
\end{aligned}
\tag{39}
$$

For the third:

$$\begin{aligned} k(x,x)\left[K_{D'}+\Lambda_{D'}\right]^{-1}_{(2,1)}k_D(x) &= k(x,x)\Big[-\frac{1}{z}k_D(x)^T\left[K_D+\Lambda_D\right]^{-1}\Big]k_D(x) \\ &= -\frac{k(x,x)}{z}\Big[k_D(x)^T\left[K_D+\Lambda_D\right]^{-1}k_D(x)\Big] \\ &= -\frac{k(x,x)}{z}\left[k(x,x)-v(x|D)\right] \\ &= -\frac{1}{z}\left[k^2(x,x)-k(x,x)v(x|D)\right] \end{aligned} \tag{40}$$

For the fourth:

$$\begin{aligned} k(x,x)\left[K_{D'}+\Lambda_{D'}\right]^{-1}_{(2,2)}k(x,x) &= k(x,x)\left[\frac{1}{z}\right]k(x,x) \\ &= \frac{k^2(x,x)}{z} \end{aligned} \tag{41}$$

Plugging these into 37, we get:

$$\begin{aligned} v(x|D') &= k(x,x)-k(x,x)+v(x|D)-\frac{1}{z}\left[k^2(x,x)-2k(x,x)v(x|D)+v^2(x|D)\right] \\ &\quad +\frac{1}{z}\left[k^2(x,x)-k(x,x)v(x|D)\right] \\ &\quad +\frac{1}{z}\left[k^2(x,x)-k(x,x)v(x|D)\right] \\ &\quad -\frac{k^2(x,x)}{z} \\ &= v(x|D)-\frac{k^2(x,x)}{z}+\frac{2k(x,x)v(x|D)}{z}-\frac{v^2(x|D)}{z}+\frac{k^2(x,x)}{z} \\ &\quad -\frac{k(x,x)v(x|D)}{z}+\frac{k^2(x,x)}{z}-\frac{k(x,x)v(x|D)}{z}-\frac{k^2(x,x)}{z} \\ &= v(x|D)+\frac{k^2(x,x)}{z}+\frac{k^2(x,x)}{z}-\frac{k^2(x,x)}{z}-\frac{k^2(x,x)}{z} \\ &\quad +\frac{2k(x,x)v(x|D)}{z}-\frac{k(x,x)v(x|D)}{z}-\frac{k(x,x)v(x|D)}{z}-\frac{v^2(x|D)}{z} \\ &= v(x|D)-\frac{v^2(x|D)}{z} \end{aligned} \tag{42}$$

Because $z=\sigma^2(x)+v(x|D)$ (equation 27), this can be written as

$$v(x|D')=v(x|D)-\frac{v(x|D)}{\sigma^2(x)+v(x|D)}v(x|D)$$

which gives us equation 11 $\square$