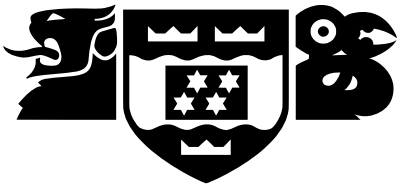
VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui



School of Engineering and Computer Science
Te Kura Mātai Pūkaha, Pūrorohiko

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Internet: office@ecs.vuw.ac.nz

Efficient Glocal Optimization for Expensive Functions

Michael Mudge

Supervisor: Marcus Frean

Submitted in partial fulfilment of the requirements for
Bachelor of Science with Honours in Computer Science.

Abstract

Optimization is an important process in many different fields, but it is often done using algorithms which take many more samples than they need to. This project concentrates on optimization algorithms which are efficient in the number of samples they take. Gaussian processes optimization (GPO) is an efficient algorithm, however it still shows areas of weakness which could be improved. GPO has two steps: build a predictive distribution which is a guess at the true function, and decide where to sample next using the predictive distribution and a measure of expected improvement. This project suggests improvements to both of these steps. The results show that these changes do improve the current algorithm when dealing with difficult functions, and do so without hindering it on simple functions.

Contents

Chapter 1

Introduction

Optimization is a fairly common task, however people often don't recognise tasks as optimization problems. A good example for an optimization problem is choosing the best driving speed to maximize fuel economy over a long drive. This is an optimization problem because we have driving speed, which is a parameter that we can change, and fuel economy which we want to optimize. There are already many ways in which we can find a driving speed that leads to an efficient fuel economy. The easiest approach is a brute force search: try each speed between 1km/h and 100km/h, and choose the one which gives the highest value. The value is the height of a continuous surface over the parameter search space. The value for the above example is the fuel economy because that is what we are trying to maximize. We can analyse optimization methods in terms of how *efficient* they are, that is, how many samples they need to take in finding the optimum. The brute force approach is at the bottom of this scale because it requires many samples before finding a good point. This gets much worse when we have more parameters because that leads to more dimensions and the curse of dimensionality (Bellman 1969) means it requires exponentially more samples.

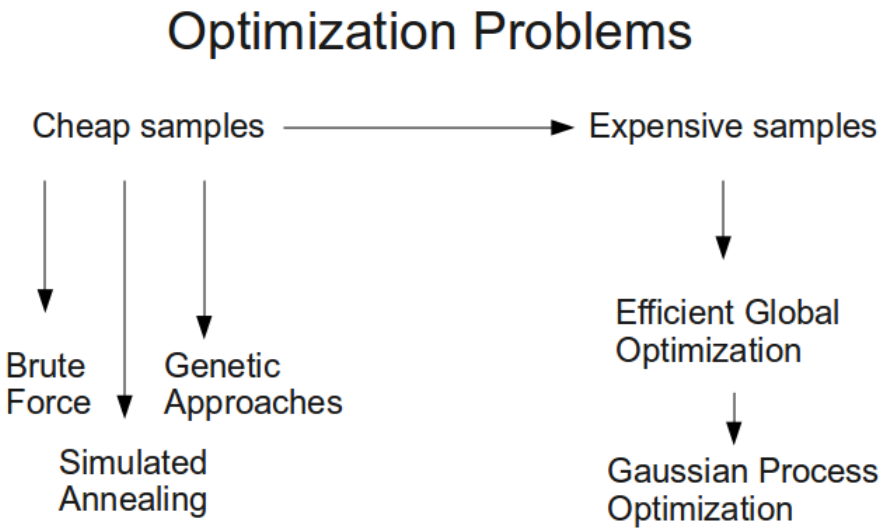


Figure 1.1: The current scale of how optimization algorithms and methods measure up in terms of efficiency.

Other optimization methods include Simulated Annealing (Kirkpatrick 1983) and Ge-

netic Programming (Koza 1992), which sample more frequently around the higher points seen so far. These approaches decrease the number of samples taken, and show sensible heuristic behaviours such as hill climbing. Figure 1.1 shows the optimization algorithms schematically, according to their efficiency.

This project is focusing on the right hand side of Figure 1.1: we want efficient global optimization (EGO) (Jones et. al. 1998, Osborne et. al. 2009, Frean and Boyle 2009) . EGO algorithms use a predictive distribution which makes use of the data already sampled to predict the value of anywhere else in the search space. They are usually more computationally intensive because they analyse heavily before deciding on where to select the next sample point in the search. Using a predictive distribution means we can get an prediction of the results for a sample without actually having to take a real sample which would be potentially expensive. The predictive distribution gives the probability for a value (y) given the point ($\mathbf{x}$) in the search space and the previously sampled data ($\mathcal{D}$), which is written as $P(y|\mathbf{x}, \mathcal{D})$. y is the value for the point $\mathbf{x}$, y is unknown because we haven't actually taken a sample at $\mathbf{x}$, we just need a variable representing the value for $\mathbf{x}$. $\mathcal{D}$ is the combination of all points we have sampled at X and the values corresponding to those points $\mathbf{y}$. $P(y|\mathbf{x}, \mathcal{D})$ is used to decide where to take our next sample, but there needs to be a measure which uses the predictive distribution and gives a utility proportional to how good a point it is for sampling at next. The next sample point is chosen by searching for the point with the highest utility. Figure 1.2 shows a graphical representation of how an iteration of an EGO algorithm would look.

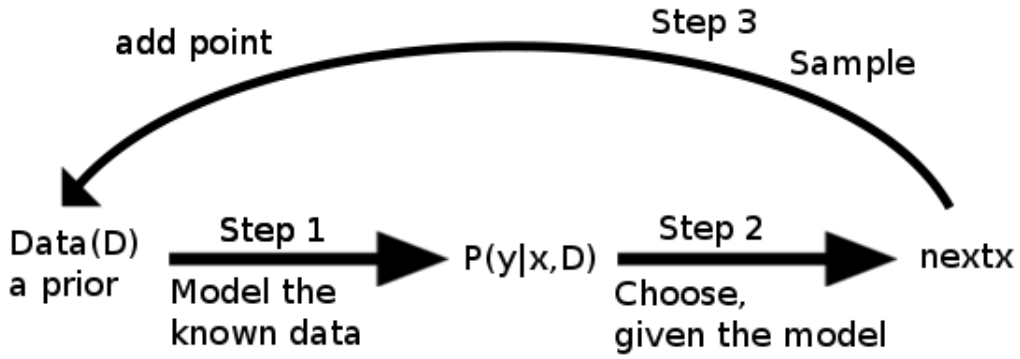


Figure 1.2: Graphical representation of EGO. In Step 1 we create a probability distribution $p(y|\mathbf{x}, \mathcal{D})$. In Step 2, we use this to find the point $\mathbf{x}$ with the highest utility. In Step 3 we add our new sample to the data, then we can use the updated data to repeat the process.

This project starts with Gaussian processes optimization (GPO) (Osborne et. al. 2009, Frean and Boyle 2009), which is used as the base algorithm for doing efficient global optimization. As the name suggests, GPO uses Gaussian processes to build its predictive distribution $p(y|\mathbf{x}, \mathcal{D})$. (The first step in Figure 1.2). Using this model means we get a Gaussian distribution for the height of any point in the search space. This algorithm uses the expected improvement (EI) over the best point as its measure for the utility of a sample point, the second step in Figure 1.2. (Sasena et. al. 2000, Osborne et. al. 2009, Frean and Boyle 2009). The EI calculates how much we expect to improve the best point by sampling at $\mathbf{x}$. A sample below the current best point will leave us with zero improvement.

GPO is already an efficient algorithm, but when analysed closely there are some problems with the first two steps. When building a predictive distribution for a function with only a small amount of data, the predictions $p(y|\mathbf{x}, \mathcal{D})$ are Gaussian with a mean of zero for large areas of the search space. This isn't a believable prediction when the sampled data is all above or all below zero for example. The EI measure which the current algorithm uses

has problems as well. It only considers the improvement over the best point so far, and so it will sample in areas which already have lots of samples. It will miss a global maximum which is surrounded by lower value points, because the algorithm will sample too often around a local maximum.

This report explains how the current GPO algorithm works, finer details of the problems the algorithm has, solutions to these problems, and how those solutions lead to improvements in the algorithm.

1.1 Contributions

My contributions to this project include three versions of the GPO algorithm which aim to improve it by...

- Using a background mean to improve the accuracy of the predictive distribution.
- Using the EI of the best mean instead of the best point.
- Using the EI of the area surrounding a point instead of just at a single point.
- The project also evaluates each of these, with a positive outcome in all cases.

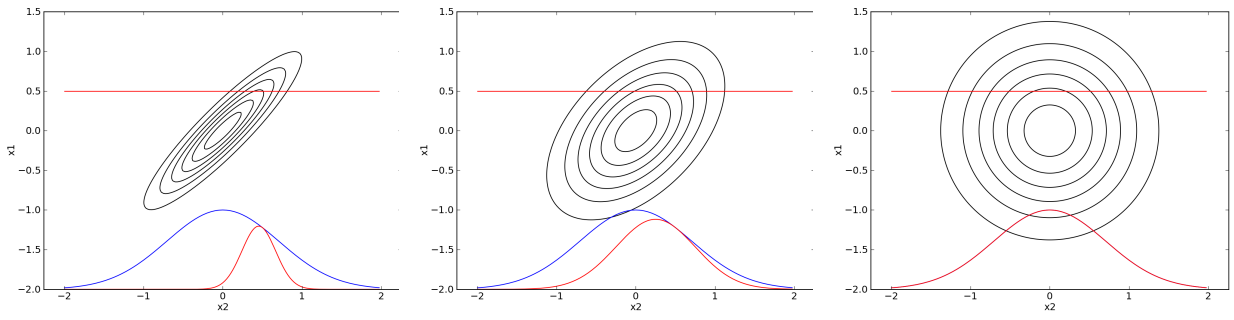
Chapter 2

Background

This chapter covers how the current Gaussian processes optimization (GPO) algorithm works. This is an EGO algorithm which uses Gaussian processes (GP) to generate its predictive distribution. The chapter explains what covariance is, and how Gaussian processes uses covariance to make a prediction about any point in the search space, the way in which we analyse how well a particular Gaussian process model fits the data, and how we choose the model which fits the data best. Lastly it shows how the currently favoured algorithms use a Gaussian process model to search for a point to sample at next.

2.1 Covariance

The covariance between two points is the amount of influence which a sample at one point has on the prediction for a sample at the other point. High covariance between two points implies that having a sample at one will heavily influence the prediction for the other. Low covariance will imply that there is a little bit of influence on the prediction when one is observed, and no covariance implies there is no influence on the prediction when one is observed.



(a) High covariance (b) Low covariance (c) No covariance

Figure 2.1: Three images showing different levels of covariance. The blue curve shows the distribution for x before y is known. the red curve shows the distribution for x once y is sampled along the red line.

Figure 2.1 shows three covariance distributions. The oval shape in the middle shows the probability distribution for x1 and x2, its highest in the middle. The larger curve at the bottom shows the probability distribution for just x2 when the height of x1 is unknown. The horizontal line through the oval represents a sample of 0.5 for the value at x1. The new

probability distribution for x2 is found by slicing the oval along x1=0.5, which is shown as the smaller curve along the bottom of the graph. Notice how the peak of the new probability for x2 is at the same point where the horizontal line crosses the highest point of the oval. When there is no covariance between x1 and x2 the curve does not change, Figure 2.1(c) seems to show only one curve because the two curves are exactly the same.

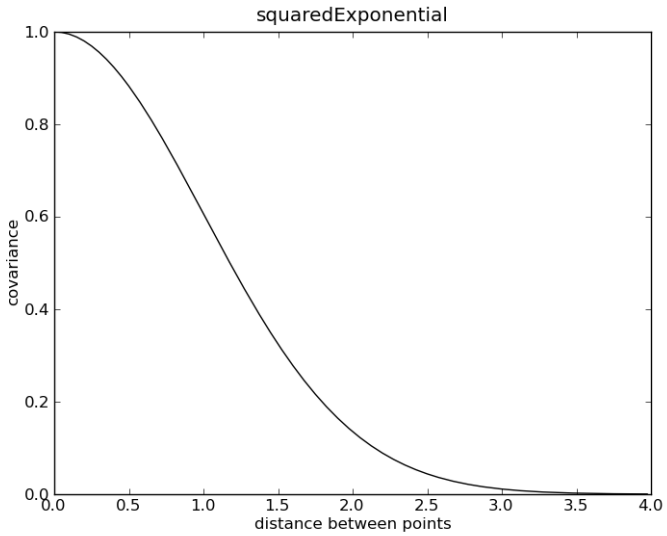


Figure 2.2: This graph shows the squared exponential covariance as a function of the distance between points. As the distance increases the covariance decreases.

The Gaussian process model is realised by parameterizing the covariance between two points as a function of the distance between them. For a continuous variables the most commonly used covariance function is a squared exponential model (MacKay 2003, Rasmussen and Williams 2006). Figure 2.2 shows this covariance between two points in a single dimension. The covariance is highest when there is no distance between the points, and drops off when the distance between the two points is larger. The function for the squared exponential covariance between two points $\mathbf{x}_i$ and $\mathbf{x}_j$ is

$$\mathrm{Cov}(\mathbf{x}_i, \mathbf{x}_j) = \alpha \exp\left[-\frac{1}{2}\sum_d \frac{(x_{i,d} - x_{j,d})^2}{2r_d}\right]. \tag{2.1}$$

Here α is the vertical scale and r_d is the length scale in the d-th dimension. These are parameters for the covariance function which are explained in more depth later.

2.2 Gaussian Processes

Gaussian process (GP) is a model which can be used to generate a predictive distribution over the underlying function using $\mathcal{D}$ which contains all points X and all sample heights $\mathbf{y}$. It is commonly used for the first step of EGO (Jones et. al. 1998, Osborne et. al. 2009, Frean and Boyle 2009). It is possible to generate a function which fits a given data set $\mathcal{D}$ of $(\mathbf{x}, y)$ pairs, but in reality there are an infinite number of functions which could fit the data. GP trys to model all the functions which could fit $\mathcal{D}$ by using a mean and standard deviation to capture the uncertainty. Figure 2.3 shows a Gaussian distribution for functions which fit the

plotted data. The black line is the mean and the green shaded area represents one standard deviation (error bars). It can be seen that predictions very close to the samples have a small standard deviation and the points further away from samples have a much larger standard deviation. This is due to the amount of covariance: close to a sample the covariance is very high and the model reflects a strong degree of belief that the value of a nearby sample will have a similar value.

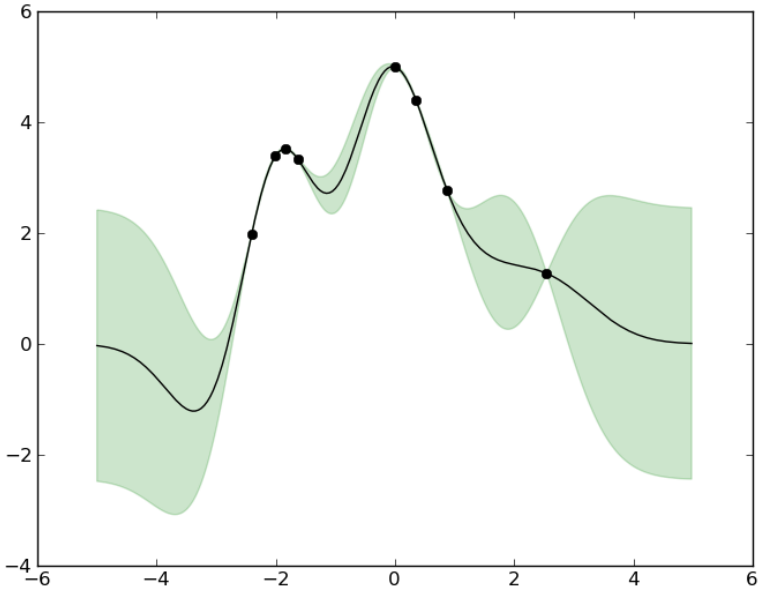


Figure 2.3: An example of a GP model predicting the functions which can fit some given data points. The black line is the mean and the green area represents one standard deviation from the mean.

To calculate the posterior mean and variance at a new point $\mathbf{x}_\star$ the GP model requires a covariance matrix. The matrix is filled with the covariance between each pair of points ($\mathbf{x} \in \mathcal{D}$) as follows:

$$\boldsymbol{C}_{ij} = \mathrm{Cov}(\mathbf{x}_i, \mathbf{x}_j) + \sigma^2_{\text{noise}}\delta_{ij} \tag{2.2}$$

$$\tag{2.3}$$

δ_{ij} is the kronecker delta function, which is 1 when i and j are equal, this means that the σ^2_{noise} is only added to the diagonal of the matrix. σ^2_{noise} is a parameter representing the assumed noise levels of the samples, and will be explained further in the next section. We can find the mean and variance using the following formulae (MacKay 2003, Rasmussen and Williams 2006)

$$\mu(\mathbf{x}_\star) = \mathbf{k}^T\boldsymbol{C}^{-1}\mathbf{y} \tag{2.4}$$

$$\sigma^2(\mathbf{x}_\star) = \kappa - \mathbf{k}^T\boldsymbol{C}^{-1}\mathbf{k} \tag{2.5}$$

where

$$\mathbf{k}_i = \mathrm{Cov}(\mathbf{x}_i, \mathbf{x}_\star) \tag{2.6}$$

$$\kappa = \mathrm{Cov}(\mathbf{x}_\star, \mathbf{x}_\star) \tag{2.7}$$

2.3 Choosing Hyperparameters

The covariance parameters are $\theta = \{r_1, .., r_d, \alpha, \sigma^2_{\text{noise}}\}$ (Rasmussen and Williams 2006). These are termed 'hyper-parameters' and can be learned from the data. The posterior is $p(\theta|\mathbf{y}, \mathbf{x})$, which is the product of the likelihood function $p(\mathbf{x}, \mathbf{y}|\theta)$ and a prior density for the hyper-parameters $p(\theta)$. The likelihood function $p(\mathbf{x}, \mathbf{y}|\theta)$ shows how well θ matches the data. $p(\theta)$ is how likely we think θ is without considering the data, this is known as the prior of θ. The posterior and its gradient can be calculated for any θ given the samples we have so far, and those are used to learn θ by searching for a maximum posterior likelihood. The conjugate gradient method is used to find the maximum posterior likelihood, multiple random initial start points are used and the best θ from all of these restarts is chosen and used for the final GP model.

Each iteration of the GPO algorithm can find a significantly different θ because each iteration will reset θ and relearn it from the available data($\mathcal{D}$).

2.4 Practical Issues with Optimizing Hyperparameters

To do the search for the maximum posterior likelihood the algorithm actually searches for the maximum posterior log likelihood. Taking the log of the likelihood does not change the position of the maximum but working in log space is more convenient for numerical routines. The actual likelihood is usually very small and problems can be encountered when storing small values in floating point numbers.

There is a problem which is encountered when we are searching for the maximum posterior log likelihood because it is not possible to invert every matrix. The algorithm's implementation uses an external library to search for a likely set of hyperparameters using the log likelihood and its gradient. When the search suggests values which correspond to a matrix with no inverse, it still requires a value for the log likelihood. Using the largest negative value possible for the log likelihood will mean those values for theta will be considered the most unlikely and therefore will never get chosen. The gradient of the log likelihood is also used by the external library to help it search so it requires a gradient even when the matrix has no inverse. Using a gradient of zero doesn't work because the algorithm believes it has found a maximum or minimum, it also can't use the gradient to head towards more likely values for θ. The search algorithm would occasionally stop the search early if it found the gradient to be zero. Instead the gradient can be set to the gradient of only the prior likelihood function for the values of θ, this means it doesn't consider the data in the likelihood gradient. It will however bring the θ values towards more likely values in the prior until it finds values which have an invertible matrix and it can use the true gradient again.

2.5 Expected Improvement

Now with a predictive distribution $p(y|\mathbf{x}, \mathcal{D})$, it is possible to search for a place to sample at next. The commonly used criterion for this is called expected improvement (EI) (Sasena et. al. 2000, Osborne et. al. 2009, Frean and Boyle 2009). Denote the value of the best sample so far by y_{best}, and the point we are looking at now $\mathbf{x}_\star$. EI calculates the improvement (I) to be expected over y_{best} given the predictive distribution $p(y|\mathbf{x}, \mathcal{D})$. If our new sample is above y_{best}, the improvement is the difference between the two points. If our new sample is below y_{best} then the improvement is zero. Knowing this, we can integrate only over the values for y which will improve y_{best} (Jones et. al. 1998, Osborne et. al. 2009, Frean and Boyle 2009).

$$I = y_\star - y_{best} \tag{2.8}$$

$$EI(\mathbf{x}_\star) = \int_{y_{best}}^{\infty} I \; p(y_\star|\mathcal{D}, \mathbf{x}_\star) \; dy_\star \tag{2.9}$$

The distribution $p(y^*|\mathbf{x}_\star, \mathcal{D})$ is Gaussian, therefore we can use the Gaussian probability distribution function ϕ and the Gaussian cumulative distribution function Φ. We need these functions to calculate the integral of a gaussian distribution, which we need to do to find the expected improvement of $\mathbf{x}_\star$.

$$\phi(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2} \tag{2.10}$$

$$\Phi(x) = \frac{1}{2}\left[1 + erf(\frac{x}{\sqrt{2}})\right] \tag{2.11}$$

These functions are only for standard normal Gaussian distributions, so we need to normalize first. Also the cumulative distribution function calculates the integral from $-\infty$ to x and we want it from x to ∞. However $p(y^*|\mathbf{x}_\star, \mathcal{D})$ is symmetric about μ so when normalized the integral from x to ∞ is the same as $-\infty$ to $-x$. We can use u, the negative standard normalized y_{best} to calculate $EI(\mathbf{x}_\star)$ as follows (Frean and Boyle 2009):

$$u = \frac{\mu - y_{best}}{\sigma(x)} \tag{2.12}$$

$$EI(\mathbf{x}_\star) = \sigma(u\Phi(u) + \phi(u)) \tag{2.13}$$

Figure 2.4 shows an illustrative example of EI for a Gaussian process predictive distribution in one dimension. Once we can calculate the expected improvement for any point in the search space by using the predictive distribution and our measure of expected improvement, we need to search for the point $\mathbf{x}_\star$ that *maximizes* this quantity. We can use any convenient search method to do this, because calculating the expected improvement occurs in the model, not the "real world" of potentially expensive samples. We assume that calculations are more time and cost efficient than taking samples from the actual function. If they aren't then we don't need to use an efficient optimization algorithm.

2.6 Problems with GPO

The current GPO algorithm is already an efficient optimization algorithm but there are some areas in which it doesn't work as efficiently as it could. There are problems with building the predictive distribution (Step 1 in Figure 1.2): for example when there aren't many samples, the predictions for large areas of the search space have a mean of zero, even when the few samples which are present suggest that the function being optimized is either lower or higher than zero. The expected improvement (Step 2 in Figure 1.2) has problems as well: it will often be high in an area which already has many samples. If there are many samples in an area it usually means that the area has been well explored and there isn't much chance of improving by sampling there. This behaviour means that GPO will miss higher peaks because it keeps sampling in an area which it already has multiple samples in.

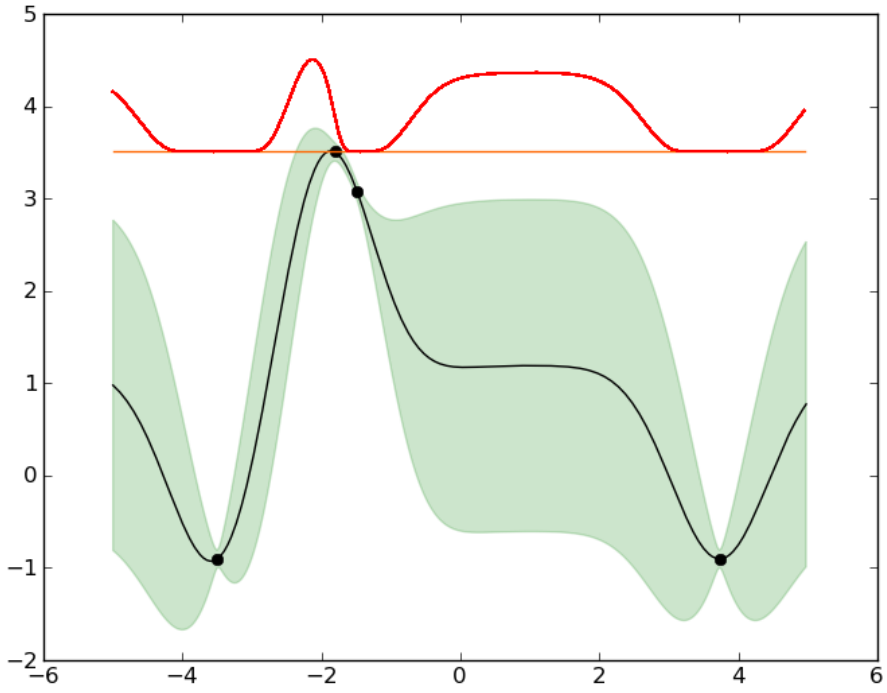


Figure 2.4: An example of EI under a GP model. The black curve is the GP mean and the green shaded area represents one standard deviation from the mean. The horizontal line is the best point so far. The EI is measured from that line, it is shown in the graph by the red curve at the top. You can see the the expected improvement is largest where the mean is high or the variance is large.

Chapter 3

Evaluation Method

There needs to be a way to compare two implementations of the GPO algorithm to see which one is better at finding the global maximum quickly. This chapter explains what criteria the implementations were measured on to do the comparison in the three chapters which follow, and the set of functions which the algorithms were evaluated on. It also examines what it is that makes a function difficult to optimize, because it is more important that EGO algorithms work efficiently on the difficult problems.

3.1 Criteria

In the context of this project, the obvious criterion for a search algorithm is its efficiency: it needs to find the global maximum with a minimal number of samples. The number of samples which the algorithm takes is used as the x-axis for the results graphs in chapters to follow.

The algorithms also need a measure of how well they have performed after each sample so they can be compared with other implementations. We can consider 3 approaches for the algorithms measure of performance:

a) use the highest point which the algorithm had discovered.

b) the highest point of the predictive distribution's mean.

c) use the highest predictive distribution's mean where a sample has been previously taken.

(a) has the problem that it would sometimes use a point which it doesn't believe it can reproduce. If the predictive distribution is low where the highest sample is, the predictive distribution doesn't believe another sample there will be high. (b) has the opposite problem: the point of the predictive distribution's highest mean can be somewhere with no samples near it. A high predictive distribution doesn't necessarily mean that the algorithm has found a high point of the function. (c) was chosen for the measure of highest utility because it used the highest predictive mean but only in places where samples had been taken.

There was a small change to (c) in that the height of the true underlying function was the actual value used for the utility measure. We use the sample where the highest predictive mean was as the point which the algorithm considers to be best. We then obtain the utility by sampling the true function at this point. This measure can only be used when the underlying function is known as it requires access to the true height. This does mean that for evaluation purposes (but not within the GPO algorithm itself), the utility measure is associated with the global maximum of the underlying function, which is what we are looking for. Using the

highest point, or the predictive distribution's highest mean, has the problem that it doesn't depend on the underlying function. It doesn't matter how good a point it has actually found but only on how good a point it thinks its found.

3.2 Test Functions

A range of functions where required to run tests on so that the different algorithms could be compared. After researching which functions are commonly used for testing global optimization algorithms (Molga and Smutnicki 2005, Sudhanshu 2006) the following functions were used as a base test suite. The changes implemented on the GPO algorithm will only show significant differences for a subset of possible functions, because the problems which the changes aim to fix are not present in all functions. Most of the following functions had local minima rather than local maxima (i.e. were designed for minimization routines in mind whereas we use maximization). In these cases the inverted function was used in place of the original function. The base test suite includes the following functions

- Dejong function
  - The Dejong function is a simple function where the height is proportional to the euclidean distance to 0. It has one global minimum
- Branin function
  - A function in two-dimensions which has three global minima.
- Six Hump Camel Back function
  - A function in two-dimensions which has six local minima, two of which are also global minima.
- Three Hump Camel Back function
  - A function in two-dimensions which has three local minima, one of which is the global minima.
- Bird function
  - A function in two-dimensions which had two global minima in the search space.
- Drop Wave function
  - A function in two-dimensions. It has a region of local minima and the global minimum is also a region. A region is an area of space which has the same value, this means the minima isn't just a single point of the function.

To make the comparisons as fair as possible, multiple runs were done and the average utility after each iteration was calculated. Each run started with two uniformly chosen points which all the algorithms would share. That is the algorithms were given the same initial samples so they had equal starting knowledge on each individual run. The initial samples are not counted in the number of samples taken by the algorithm, so the utility recorded for zero iterations in the results is the utility calculated using the two initial samples. The results shown were averaged over 100 runs. Ideally more runs would have been done but due to the number of functions which the algorithms were being run on and the computational time required to run the tests this wasn't feasible.

3.3 Functions that are Hard to Optimize

Something which became very obvious during evaluation was the effect of the function's complexity on the results. A function is considered to be easy to optimize if it has a single maximum, or if all its maxima are of a similar height. If all maxima are the same height the algorithm only needs to find one, and any simple hill climbing algorithm can achieve this quickly. All the functions chosen above, which are widely used for evaluation in global optimization, seemed to be in this category of "simple to optimize".

A difficult problem should involve some local maxima which are easier to find and a global maximum which is not so easy to discover. The global maximum should have a value which is noticeably larger than the local maxima. To represent this type of function a new test function was developed using a combination of Gaussian hills which were summed together. The function could be altered by changing the height, standard deviation and location of each Gaussian hill. The global maximum was created as a Gaussian hill with a *small* standard deviation so it was difficult to find, and it also had the largest height so it was definitely the global maximum. Multiple variations of these hard-to-optimize functions were created and used to evaluate the algorithms more thoroughly. Without any functions used in other global optimization evaluations, there should be more evaluation done in the area of hard-to-optimize functions. Figure 3.1 shows an example of a hard-to-optimize function in 2 dimensions, the height of the function at each point is shown as the height of the blue curve. This problem is a 2-dimensional problem because there is 2 parameters which can be changed,but it is shown in 3-dimensions because this indicates the value of the function at every possible combination of the two parameters.

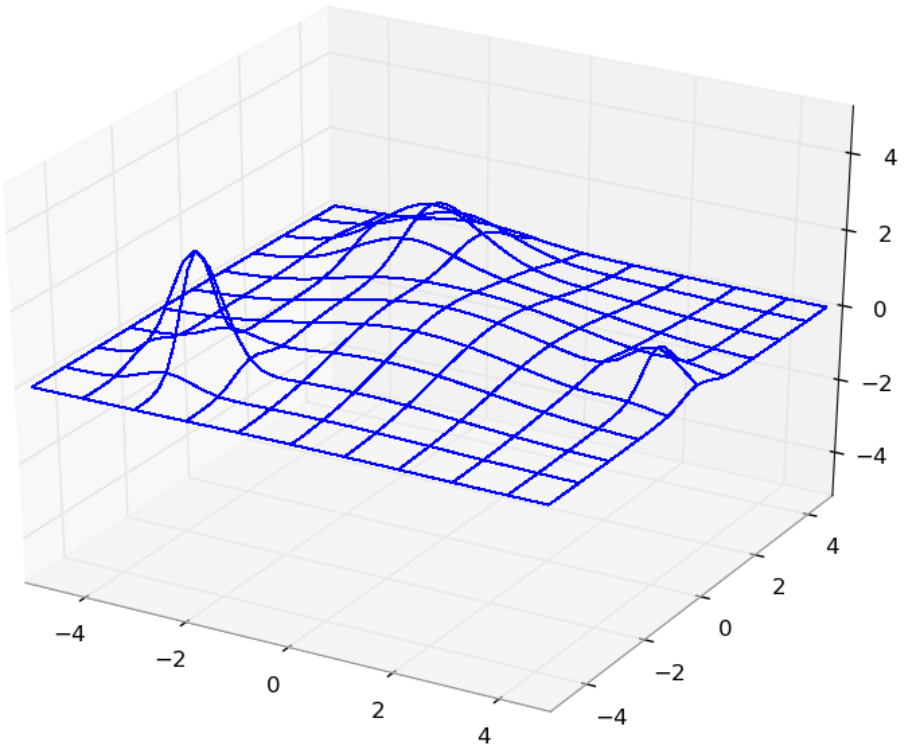


Figure 3.1: An example of a hard-to-optimize function. Notice it has multiple maxima, each with a different height and location.

Chapter 4

Setting The Background Mean

It is possible to improve GPO by improving the way in which the predictive distribution is built from the data(Step 1 in Figure 1.2) The current algorithm has as a default a background mean of zero. This reflects an assumption that the predictive distribution's mean is zero in the areas of search space which don't have any samples. This chapter proposes two alternatives to use for the background mean, and evaluates their effect on GPO.

4.1 The Problem

Gaussian Processes allow us to predict a y value for any $\mathbf{x}$ point within our search space, just by using some samples ($\mathcal{D}$). Each sample will have an influence on predictions else where, and this influence is determined by the covariance between the sample and the other point. When we need a prediction for an $\mathbf{x}$ which is close to the samples in X, Gaussian Processes relies heavily on these samples to calculate the prediction for y. However when the $\mathbf{x}$ is far away from all of the samples in X, Gaussian Processes don't rely alot on the samples, meaning the resulting predicted mean is very close to zero.

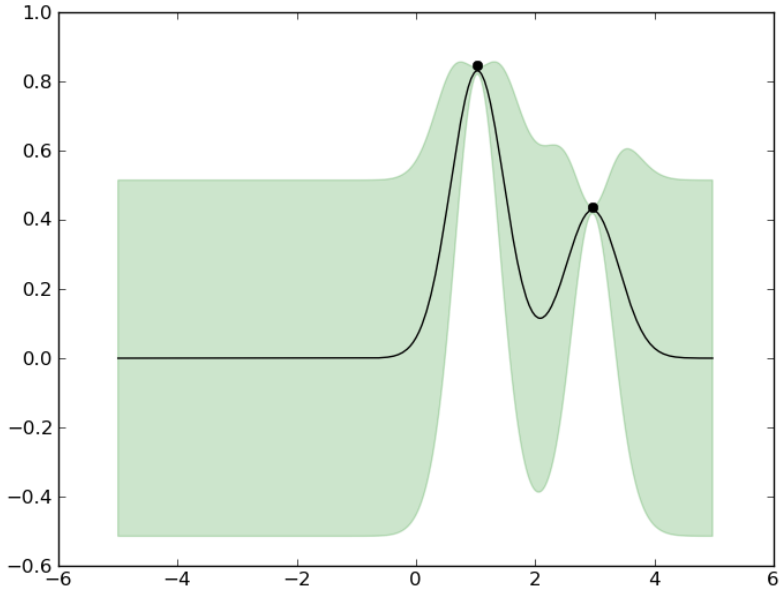


Figure 4.1: The mean drops back to zero on both sides of the data, even though we believe the mean is higher from the samples so far.

By way of example, Figure 4.1 shows a simple GP model which predicts a mean of zero for most of the search space, despite both samples being above zero. We do not usually believe that the mean of a function is zero, so we shouldn't make predictions which assume this mean. The current GPO algorithm will not perform well when the actual mean of the function is much lower, or much higher, than zero. When the mean is lower than zero, Gaussian Processes will make predictions which are higher than the actual function. When the mean is higher than zero, the predictions will be lower than the actual function. This problem goes away to some extent as more samples are added to our data, because the predictions are only bad when made about areas of the search space which don't have samples. However that is small consolation for optimization algorithms which by definition need to search novel areas.

4.2 Solution

The solution to this problem is to incorporate a background mean into the Gaussian Processes prediction. This background mean will be used when making predictions for any point, only it will become less important when the prediction is heavily influenced by the other samples.

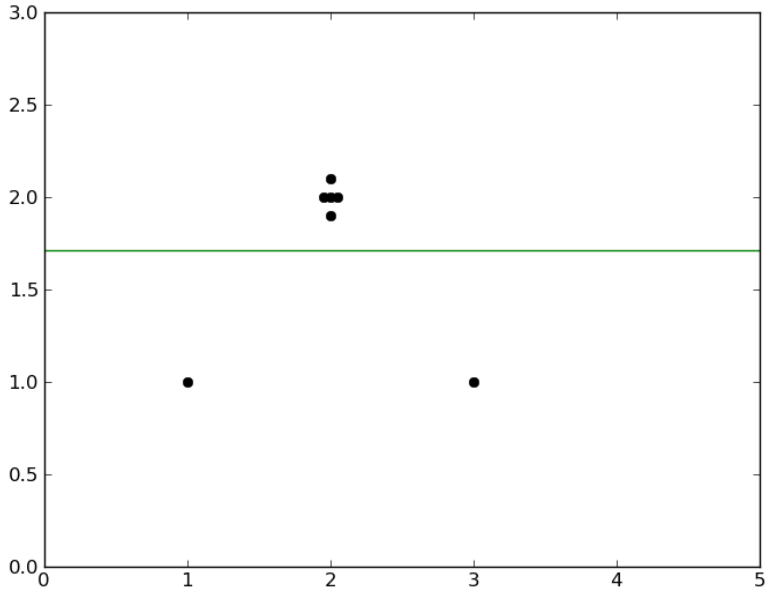


Figure 4.2: The mean of the samples(horizontal green line) is much higher than what we believe the mean of the function is. This is caused by the cluster of points.

The easiest answer is to use the mean of all of the samples we have taken: as they are all from the function which we wish to know the mean of, so they should give us an idea of the functions mean. However, the samples from the function have *not* been taken randomly and so will not accurately represent the mean of the function. The samples have been taken by an algorithm which is trying to optimize the function, so we believe that it will sample more often near the peaks of the function. The mean of the samples will therefore be higher than the mean of the function, and this will only get worse as our algorithm samples more frequently around the high points. Figure 4.2 shows a cluster of high points which causes

the sample mean to be higher than we believe the true mean of the function to be.

Instead of using the mean of the samples, we should use the value which gives the maximum likelihood for the Gaussian Processes distribution. We can use the likelihood $p(\theta|\mathcal{D})$ to find this value:

$$log\ p(\theta|\mathcal{D}) = (\mathbf{y} - \mu)\boldsymbol{C}^{-1}(\mathbf{y} - \mu) + \text{terms which do not include } \mu \tag{4.1}$$

This is a quadratic so we can easily find the maximum by differentiating, setting to zero and solving for μ, which yields[1]:

$$\mu = \frac{\sum_i (\boldsymbol{C}^{-1}.\mathbf{y})_i}{\sum_{ij}(\boldsymbol{C}^{-1}_{ij})} \tag{4.2}$$

This is the max likelihood value for the background mean. The formula above involves the covariances between each point: since each y is weighted by the inverse covariance. The y value is weighted by the covariances between the x where the sample was taken and all the other samples in X. Because the covariance matrix is inverted, points which are close together will have smaller values in the matrix than points which are far apart. If a point is close to many other samples, its weight will be smaller, as there will be more small values in the sum for that weight. When a point is far away from all the other samples, its weight will be larger, as all the values in the sum will be large. This means that the calculated mean of the background will be weighted more towards samples which do not have other samples near them and less towards samples which have many samples near.

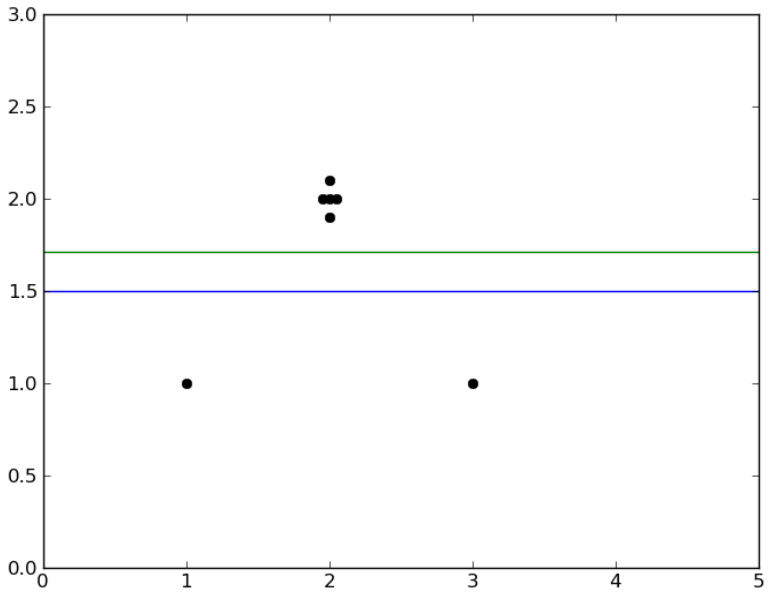


Figure 4.3: Here the calculated mean(horizontal blue line) is lower that the mean of the samples(horizontal green line) even when there is a cluster of high points. The calculated mean is much closer to where we would expect the mean of the function to be.

[1]This formula arose following some discussion during a weekly meeting with my superviser Marcus Frean.

I implemented three versions of the GPO algorithm: one used a mean of zero, one used the mean of the samples and the last used the calculated background mean. They all used the same basic algorithm, but each had a different way of calculating the background mean.

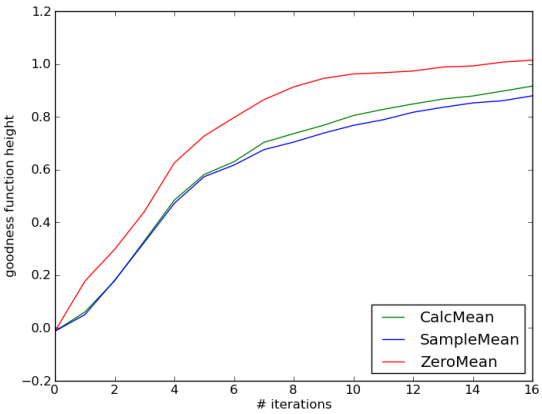
4.3 Results

The main expectation for the results was that the existing version of the algorithm with $\mu = 0$ would not perform very well when dealing with functions which weren't averaged on zero. The functions from the test suite were used and the results are displayed in Figure 4.4. The results for the test suite didn't show much difference between the algorithms. Figures 4.4(a) and 4.4(c) show that the zero mean version did slightly *better* on average. However when the algorithms were tested on a simple two dimension Gaussian curve which had been lowered so that its mean would be below zero, the differences are clear. Figure 4.5 shows the clear advantages which the calculated mean and sample mean have over the zero mean version of the algorithm on functions which are below zero.

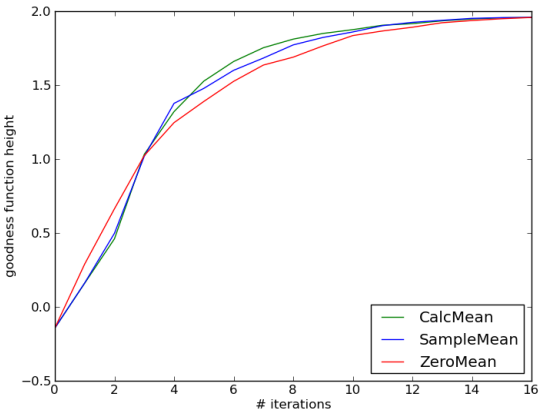
The results where the zero mean algorithm is doing better can be explained by the concept that when the true function mean is above zero, the algorithm will predict lower utilities in most areas. This low prediction encourages the algorithm to climb the first hill it finds, which means the algorithm does very well on functions which don't have any local maxima other than the global one. The same is not the case when the function does have some low local maxima because the zero mean algorithm will get stuck in these local maxima for longer than the other algorithms. Figure 4.6 shows the results for a hard-to-optimize function.

It may be acceptable for an implementation to under-perform when dealing with simple functions because of the much larger improvements gained when dealing with difficult functions and functions which are below zero.

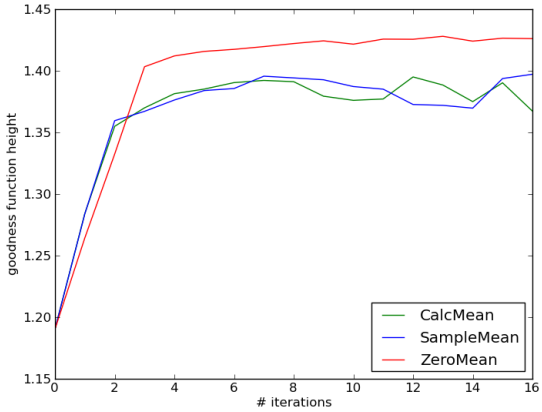
To summarize, a problem was identified with the predictive mean predicting zero for large areas of the search space. Two solutions which added different background means to the predictive distribution were proposed. Evaluation confirmed the solutions do improve the algorithm when optimizing functions which are below zero and difficult functions with multiple maxima, while the original algorithm still performs slightly better on simple function which don't contain local maxima.



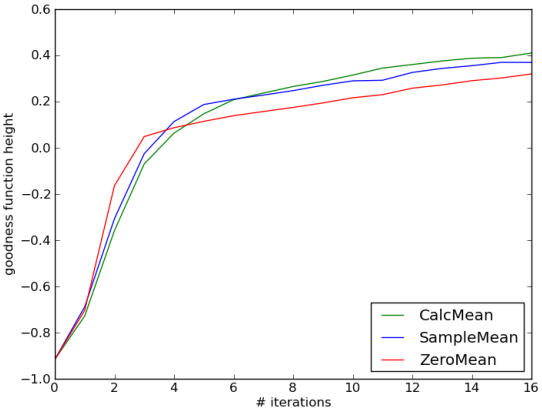
(a) Bird function results



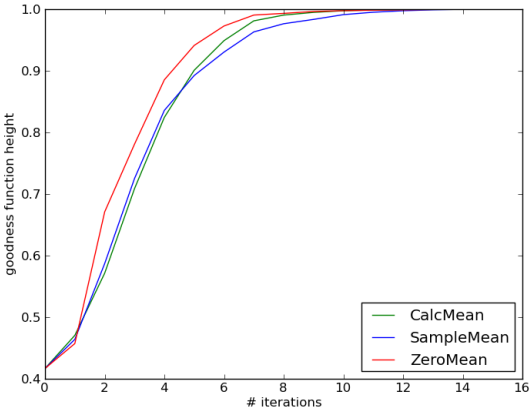
(b) Branin function results



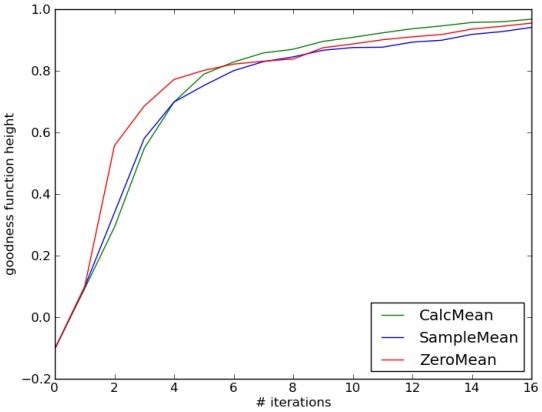
(c) Dropwave function results



(d) Six Hump Camel Back function results



(e) Dejongs function results



(f) Three Hump Camel Back function results

Figure 4.4: Results on standard problems for the different background means.

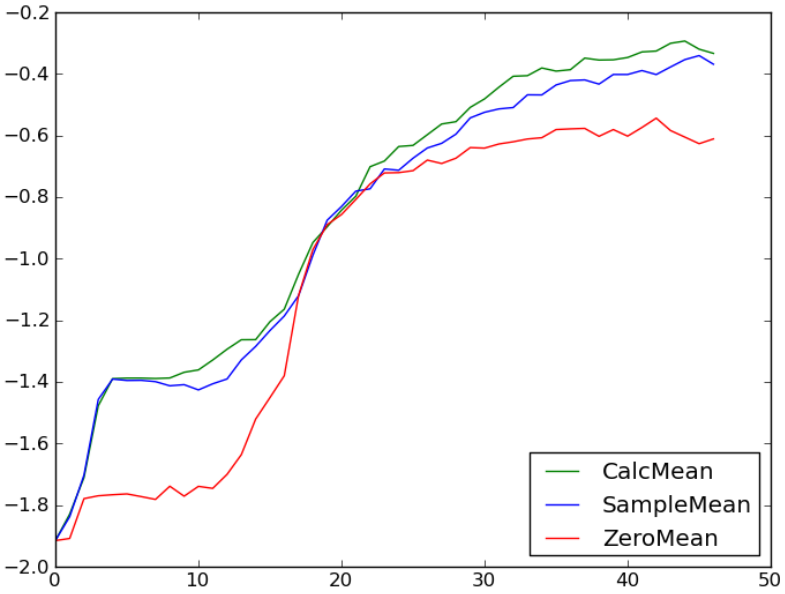


Figure 4.5: The results for a Gaussian hill in 2 dimensions, we can clearly see that the $\mu = 0$ algorithm is not doing as well during the beginning and end of the search.

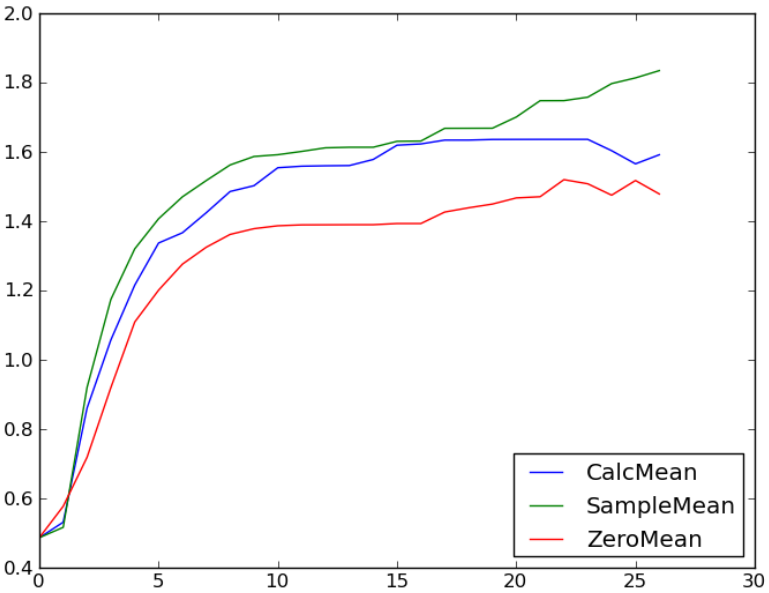


Figure 4.6: Results for a hard-to-optimize function with a mean above zero. The $\mu = 0$ version quickly finds a local hill but doesn't carry on to find the global maximum as quick as the new implementations.

Chapter 5

Expected Improvement Over The Best Predictive Mean

It is possible to improve GPO further by improving how it decides how valuable a point is to sample at for the next sample (Step 2 in Figure 1.2) The current algorithm considers a point valuable if it has a high EI over the best sample point so far. This doesn't consider that the point may already have samples around it which make it fairly clear that the point won't improve very much. This chapter suggests an alternative criterion which uses the EI over the best mean, and evalutes the effects on GPO.

5.1 Problem

The current measure for choosing where to sample next is the expected improvement, which calculates the improvement by considering the difference between our next sample and the previous best sample. If our next sample is lower than the current best then the improvement is zero, because we still have our previous best sample and there is no improvement. To find the EI, we need to multiply the probability of y by the improvement a sample of y gives us. We can find the EI by integrating over all the possible y values we could get for our next sample. This measure doesn't seem to take into account the places we have sampled before.

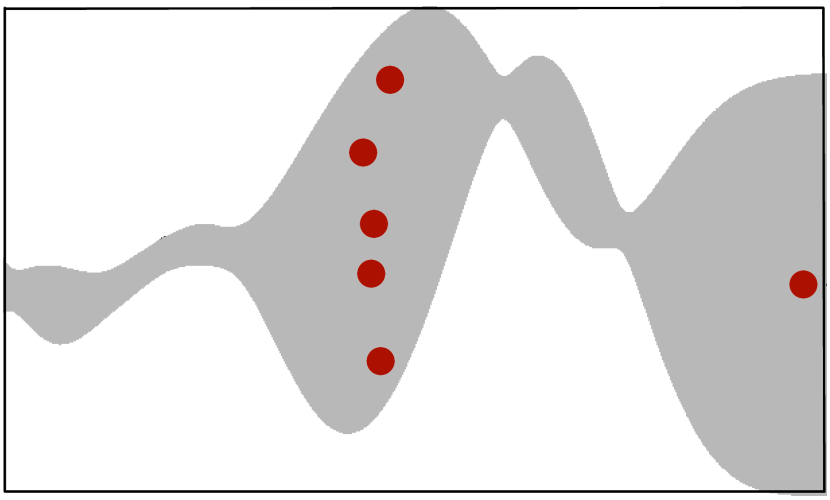


Figure 5.1: The current algorithm will be just as like to sample on the left as on the right.

We shouldn't aim to get our sample higher than the best *sample* so far, because that will lead to the algorithm sampling where it "got lucky" once (an anomalously high sample). Instead, we want our algorithm to sample where the GP's predictive mean is likely to be high.

5.2 Solution

Marcus Frean had already outlined a solution to the problem, which I chose to implement. The idea was to calculate the EI of the predictive distribution's mean, instead of the EI for the best sample. To find the EI of the mean we need to calculate the new mean after a sample at $\mathbf{x}_\star$ with a height of $y_\star$. We would need to add $\mathbf{x}_\star$ to the covariance matrix and re-invert it.

5.2.1 Improving The Mean

If we use the EI of the mean over the best mean we should get behaviour which makes the algorithm less likely to sample near other sample points. The mean will not move as much when there are other samples influencing its position. When there are no samples nearby, the mean will only be influenced by the new sample, meaning the algorithm acts the same as the current EI when looking away from other samples.

Matrix Block-wise Inversion

To calculate the new mean we need a covariance matrix which includes the new $\mathbf{x}_\star$. To recreate the covariance matrix and then invert it would be slow and inefficient. However, by using the partitioned inverse equations (MacKay 2003) we can show that

$$\boldsymbol{C}_{N+1}^{-1} = \begin{bmatrix} \boldsymbol{M} & \mathbf{m} \\ \mathbf{m}^T & m \end{bmatrix} \tag{5.1}$$

where

$$m = \left(\kappa - \mathbf{k}^T \boldsymbol{C}_N^{-1} \mathbf{k}\right)^{-1} \tag{5.2}$$

$$\mathbf{m} = -m\, \boldsymbol{C}_N^{-1} \mathbf{k} \tag{5.3}$$

$$\boldsymbol{M} = \boldsymbol{C}_N^{-1} + \frac{1}{m} \mathbf{m}\mathbf{m}^T. \tag{5.4}$$

Now we don't have to create $\boldsymbol{C}_{N+1}^{-1}$ from scratch, which speeds up the process.

5.2.2 New Expected Improvement

We need to calculate the EI of the mean. To do this we integrate over the improvement multiplied by the distribution $p(y_\star|\mathbf{x}_\star, \mathcal{D})$. We need to find what the mean will be after a sample has been taken at $\mathbf{x}_\star$ with a height of $y_\star$.

$$\mu'_\star = \left[\mathbf{k}^T,\ \ \kappa - \sigma^2_{\text{noise}}\right] \boldsymbol{C}_{N+1}^{-1} \begin{bmatrix} \mathbf{y} \\ y_\star \end{bmatrix} \tag{5.5}$$

Because we only want to integrate over the $y_\star$ we can simplify the above by defining

$$[\mathbf{w} \;\; w_\star] = \left[\mathbf{k}^T, \;\; \kappa - \sigma^2_{\text{noise}}\right] \begin{bmatrix} \boldsymbol{M} & \mathbf{m} \\ \mathbf{m}^T & m \end{bmatrix}$$

for which $\mu'_\star = \mathbf{w}.\mathbf{y} + w_\star y_\star$.

$$\begin{aligned} \mathbf{w} &= \mathbf{k}^T \boldsymbol{M} + (\kappa - \sigma^2_{\text{noise}})\mathbf{m}^T \\ &= \frac{\sigma^2_{\text{noise}}}{\sigma^2_\star} \boldsymbol{C}_N^{-1} \mathbf{k} \end{aligned}$$

and

$$\begin{aligned} w_\star &= \mathbf{k}^T \mathbf{m} + (\kappa - \sigma^2_{\text{noise}})m \\ &= 1 - \frac{\sigma^2_{\text{noise}}}{\sigma^2_\star} \\ \sigma^2_\star &= \sigma^2(\mathbf{x}_\star) \quad \text{from eq 2.5} \end{aligned}$$

The lower limit for integration and improvement for this method is

$$\begin{aligned} y_0 &= \mu_\star + \frac{\mu_{\text{best}} - \mu_\star}{w_\star} \\ I(x_\star) \;\; &= \mu'_\star - \mu_{\text{best}} = (y_\star - y_0) w_\star \end{aligned}$$

We can convert to the standard normal the same way as original GPO to find the EI of the mean, which I denote $EI\mu$:

$$EI\mu(x_\star) = w_\star \sigma_\star \left[u\Phi(u) + \phi(u)\right]$$

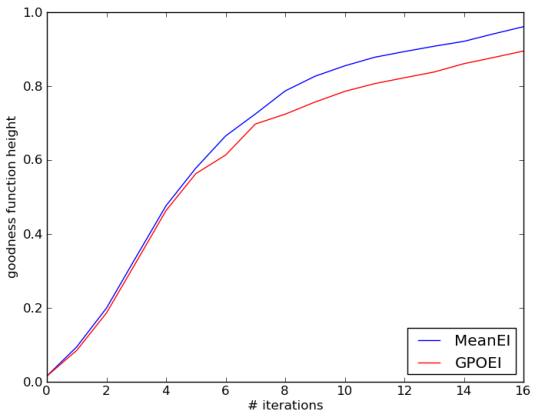
where

$$\begin{aligned} u &= (\mu_\star - y_0)/\sigma_\star \\ &= \frac{\mu_\star - \mu_{\text{best}}}{w_\star \sigma_\star} \end{aligned}$$

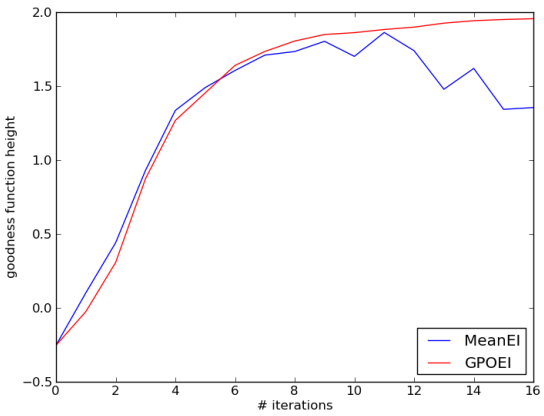
The difference between this and the existing EI is the introduction of $w_\star$. This quantity will be one when $\mathbf{x}_\star$ is distant from other samples, but it is smaller when the new sample is close to other samples. It is easiest to think of this as a weight between zero and one of how much this new potential sample will affect the mean. If we tried to find the EI of a point which already had a single sample, $w_\star$ would be 0.5, because both samples affect the new mean equally.

5.3 Results

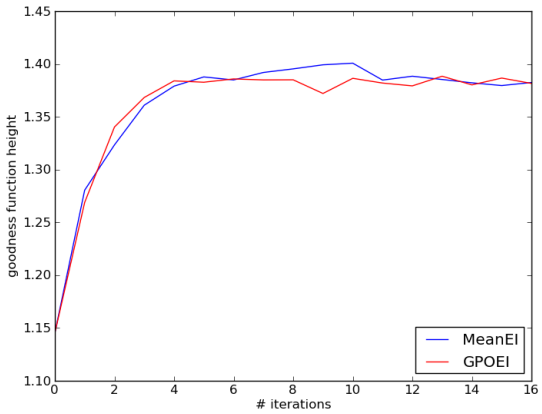
Using the best mean to calculate the EI doesn't show much difference to the ordinary algorithm when tested it on the test suite of functions. Figure 5.2 shows the results for the test suite functions, most cases don't show a significant difference between the two algorithms. The only noteworthy part of those results is in Figure 5.2(c) where the algorithms behave the same for the first part but then the `EImean` plot acts strangely. The problem here is that



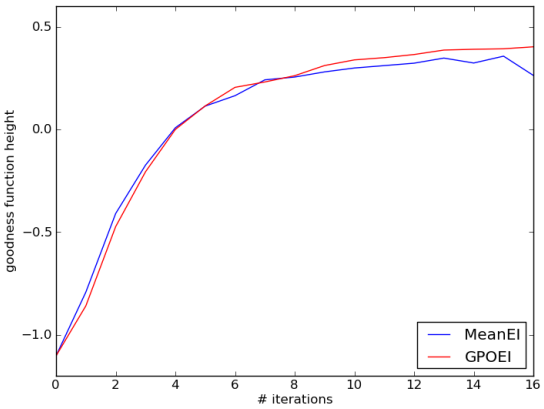
(a) Bird function results



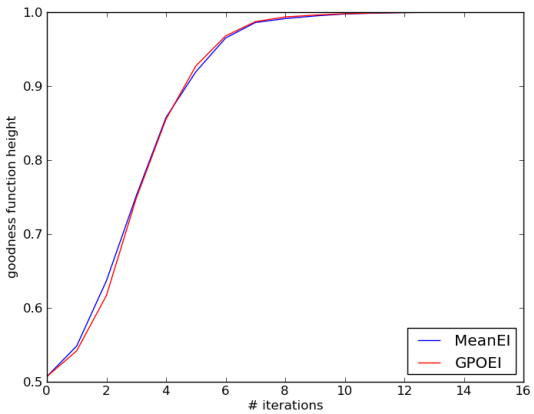
(b) Branin function results



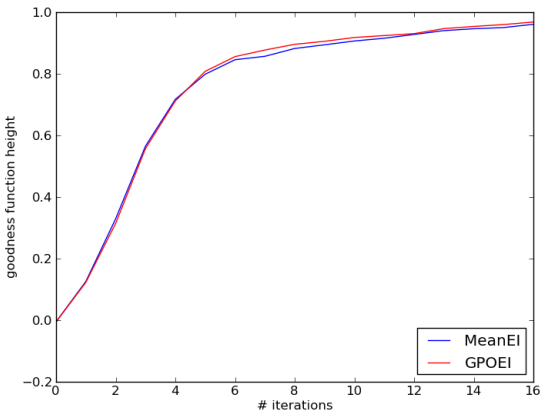
(c) Dropwave function results



(d) Six Hump Camel Back function results



(e) Dejongs function results



(f) Three Hump Camel Back function results

Figure 5.2: The results for the EI on best mean compared to the EI on best point for the test suite functions.

GP is using a bad predictive distribution and therefore not predicting the data accurately. This is further explained in the Future Work section (Chapter 7).

The EI on best mean solution is supposed to do better than the current algorithm be-

cause it doesn't sample near previous samples. This only really becomes apparent when the current algorithm is sampling in the same area when there is also a higher maximum to be found, i.e. when optimizing a difficult function. Figure 5.3 shows the results for the difficult function described in the evaluation chapter. The increasing performance with more iterations is because the algorithm was run multiple times and the results were averaged, the slope is due to the global maximum being discovered by more runs after each iteration. The original algorithm would often not find the global maximum because it sampled far too frequently around the local maxima. More samples didn't seem to help either because it would get quite stuck on the local maxima. These results really show the problem with the original algorithm, it will happily keep sampling around a high point because its much more likely to find an improvement near the high point. The fact that we already have many samples in that area doesn't deter it, if we use the improvement on the mean we are discouraging the new implementation from sampling near other samples.

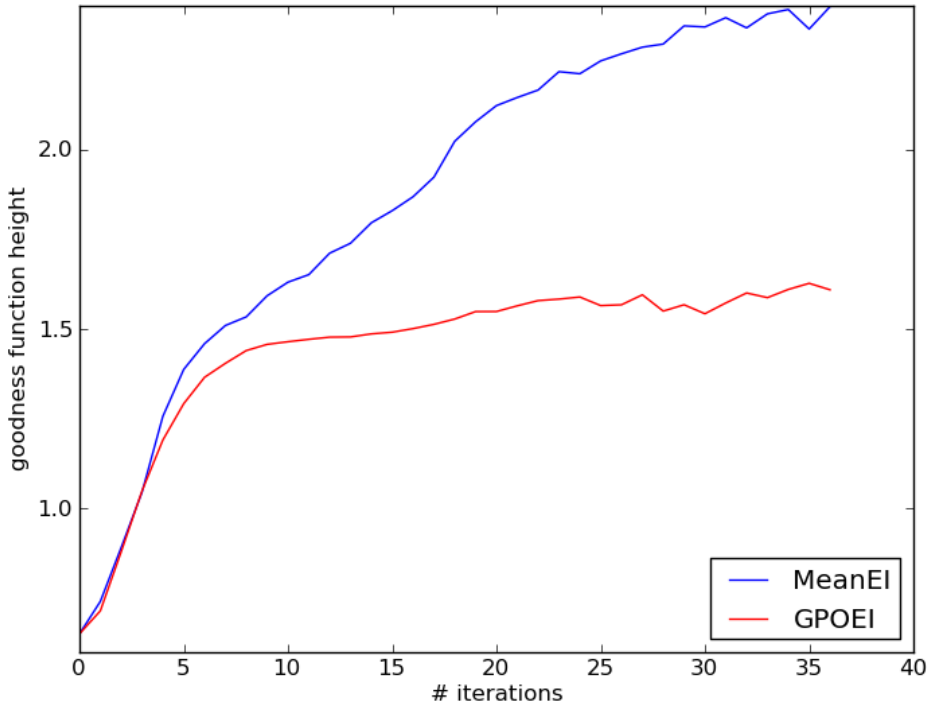


Figure 5.3: Results on a hard-to-optimize function averaged over 100 samples. EI on best mean shows significant improvement over the vanilla GPO when dealing with functions with multiple local maxima. They start similarly but the vanilla GPO doesn't find the global maximum as often as the EI on best mean version.

algorithm	#runs
EI over the best point	13
EI over the best mean	51

Figure 5.4: A table showing the number of runs which had found the global maximum after 30 samples. The numbers shown are for Figure 5.3. It shows a significant improvement in the number of runs which have been able to find the global maximum when using the new measure for EI.

To summarize, a problem was discovered with the way the current EI is evaluated, which leads GPO to sample around areas which already have many samples. A solution was proposed where the EI would aim to increase the mean of the predictive distribution instead of the best point. Evaluation shows very little difference between the algorithms when optimizing simple functions with no low local minima. However the results show a significant improvement when optimizing difficult functions, where the new algorithm will find the global maximum more often than the original algorithm.

Chapter 6

Future-Aware Optimization

It is possible to improve GPO by improving how it decides how valuable a point is to sample at for the next sample (Step 2 in Figure 1.2). The current algorithm considers a point valuable if it has a high EI over the best point so far. This only considers the EI for *the next sample*, it doesn't consider any samples further ahead. If there is EI somewhere near a point it could lead to higher EI for future samples. This chapter considers an alternative measure which considers how much EI there is in the area surrounding a point, and evaluates its effects on the GPO algorithm's performance.

6.1 Problem

The current EI measure looks at getting the best sample that it possibly can on the next sample, which makes it a greedy approach to optimization.

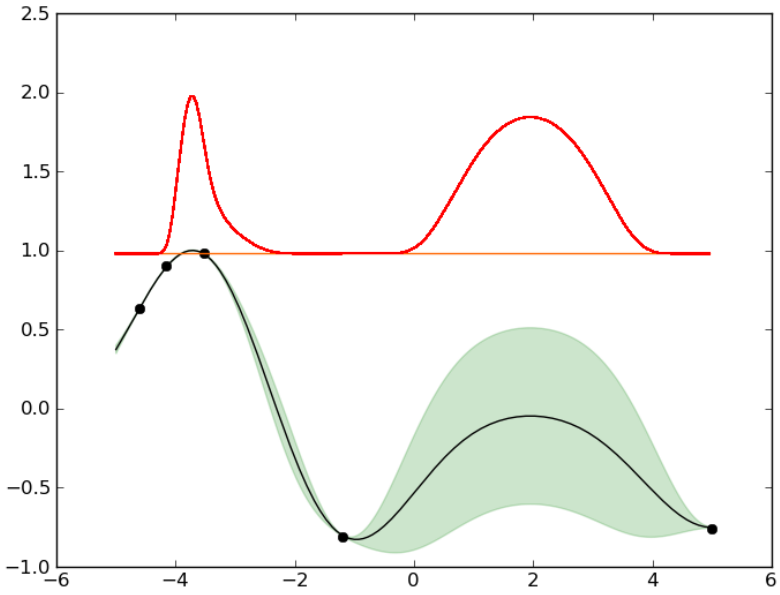


Figure 6.1: Greedy optimization would sample on the left, but is this the right thing to do?

We don't believe we will find the best sample on the next try, otherwise we would only ever need to take one sample! Usually when we are optimizing we know we will be taking

several more samples, and we should use this knowledge while considering where to place our next sample location.

Consider Figure 6.1. Should we sample at the highest peak? This is where the current greedy optimization would consider the best point. With only one sample left it would be best to sample at the highest peak, because that is where the most improvement is expected. However if there were two samples left, sampling at the high peak might find a higher point but it will remove the EI for the next sample. The last sample will get taken in the other peak but probably not at the highest point in the area. If the algorithm had sampled in the wider peak first it would discover more about the area, which will give us a better choice for where to take our last sample. If the new sample is bad, it doesn't change the highest peak because it was in a different area so there is still the same chance of a good sample. If the new sample is good, we can use our new EI curve which will probably be higher around where our new sample was taken. The aim of the second last sample shouldn't be to find the best sample, it should be to find the highest EI for the next sample.

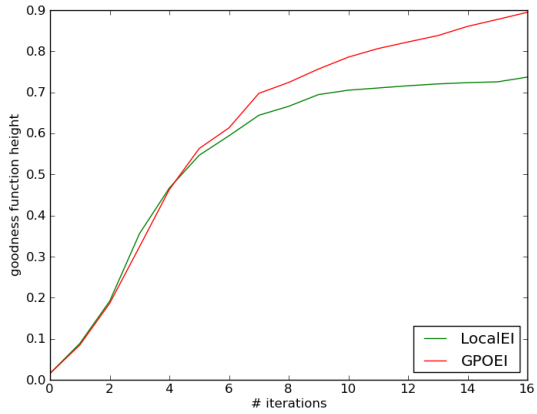
The idea of using the knowledge that we have more samples remaining is not a new one. (Osborne et. al. 2009) suggested the idea of multi-step lookahead, but to do it they integrated over all the possible places that the next sample could be taken. This was very mathematically complicated and makes it computationally completely intractable to calculate the EI. This chapter pursues a different approach that is both simpler and does not lead to a computational explosion.

6.2 Solution

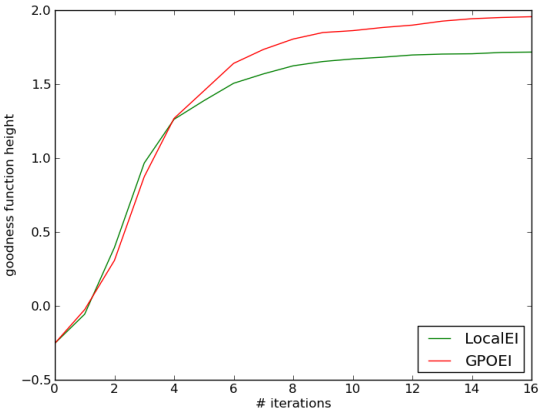
The solution aims to get a higher EI for the next few samples without doing a full integration of everywhere that a sample could get taken next. Sampling at thin peaks of EI usually results in the peak being removed, because the sample makes the predictive distribution more confident at the point where the sample was taken. However, sampling at wider peaks of EI sometimes results in higher EI around the sampled point. These observations led to the conclusion that wider areas of substantial EI were more likely to lead to more cumulative EI in the future than thin peaks of similar height. The measure of EI should be high in places which might lead to a high point being discovered.

As a proxy to the full integration, the algorithm used a measure for EI which considers the EI surrounding the point($\mathbf{x}_\star$). To do this we used the average EI for points sampled from a Gaussian distribution which was centered on $\mathbf{x}_\star$. The length scales for the covariance in each dimension were used as the standard deviation in each dimension of the Gaussian distribution. It makes sense to use this method because the length scale affects how quickly the covariance between points will decrease as the points are further apart. A dimension with a larger length scale will have a larger standard deviation because there is more covariance, and a sample will affect EI further away. The offset points were generated as distances from $\mathbf{x}_\star$ before doing the search for the highest EI. If this was not done then the random offsets resulted in different EI each time we calculated it for the same point, which led to a spiky graph. By using the same offsets for the entirety of the search the resulting EI curve was smoother.

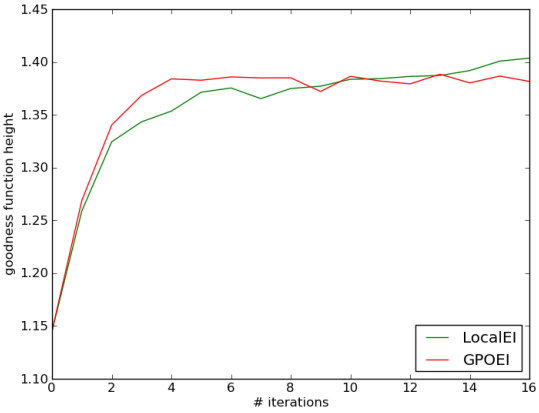
The algorithm implemented for this solution used the original GPO EI measure. The preliminary results for EI based on the best mean, were not showing significant improvements at that point, so it was decided that the future-aware measure of EI would aim to improve the original measure of EI. It would have been nice to use the EI on the best mean and this is my first suggestion for future work.



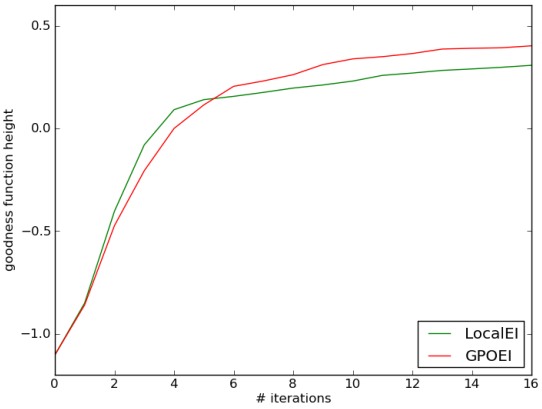
(a) Bird function results



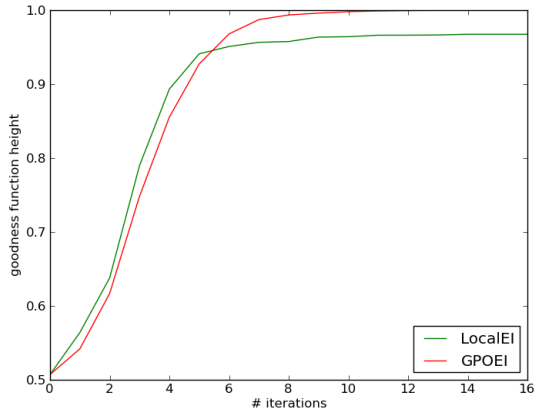
(b) Branin function results



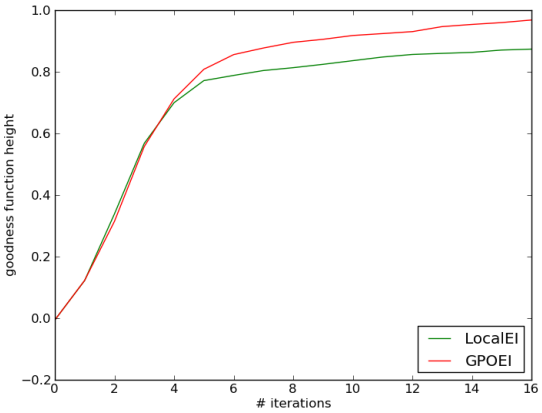
(c) Dropwave function results



(d) Six Hump Camel Back function results



(e) Dejongs function results



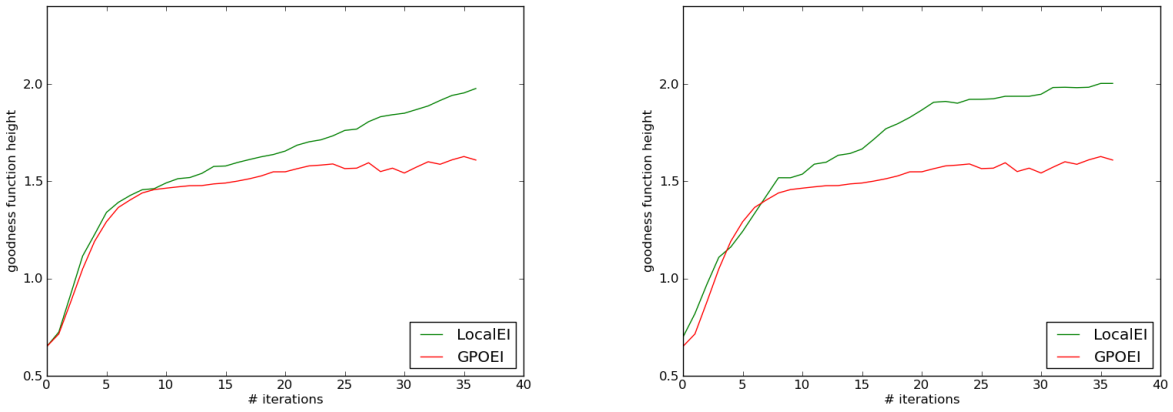
(f) Three Hump Camel Back function results

Figure 6.2: The results for the EI of the surrounding area compared to the vanilla GPO. The fraction used for the standard deviation was half

6.3 Results

The EI for the Local area measure performed slightly worse for most of the test suite functions, shown in Figure 6.2. It does start about the same but doesn't seem to do as well at finding the highest point of a peak. However the new EI measure does work significantly better on the difficult optimization functions.

The Gaussian distribution which was used to generate the offset points can have its standard deviation scaled by a fraction. The value of the fraction corresponds to the size of the surrounding area used by the measure. This fraction can be any number but if it is set to 0 then the behaviour will become the same as the standard EI because the EI will only be measured at the point, not in the surrounding area. If the fraction is set to a large number the EI for a point will depend on the EI for a large surrounding area. Figure 6.3 shows how different values for the fraction affect how much better the local EI measure will perform. Having a fraction of one showed worse results than either of the two fraction values displayed. Using a value of a half for the fraction showed the best performance in the early tests, so it was used for the results on the standard functions.



(a) Using a fraction of one fifth

(b) Using a fraction of half

Figure 6.3: The results for the different fractions of length scale used for the standard deviation of the local area Gaussian distribution.

There is evidence in the results which suggests that a large value of the fraction corrosponds to a more future-aware optimization algorithm. As the fraction's value is set closer to zero the algorithm becomes less future aware. Figure 6.3(a) shows that using a value of one fifth for the fraction results in steady improvement as more samples are taken. Figure 6.3(b) shows that using a value of half results in a steeper improvement earlier but later samples dont improve as much. Figure 6.4 shows a table containing the number of runs which had found the global maximum at iteration 30. Its possible that using a cooling scheme for the fraction would show good results. The fraction would start with a large value which corresponds to a future-aware algorithm. The fraction's value would get reduced as the number of remaining samples decreases which leads to the algorithm becoming less future-aware. For the last sample the fractions value would be zero which means the last sample will use the original greedy version of EI. This greedy approach is appropriate to use for the last sample as there is no point in being future aware with only a single sample left.

algorithm	#runs
EI over the best point	13
EI using a fraction of one fifth	18
EI using a fraction of half	20

Figure 6.4: A table showing the number of runs which had found the global maximum after 30 samples. They show that using the new measure of EI shows an improvement in the number of runs which have reached the global maximum after 30 samples.

To summarize a problem was discovered where the current measure for EI doesn't consider any samples except the very next one. A solution was proposed where the new EI measure tried to sample at points which would lead to high EI for the future samples. This was achieved by using the EI of the area around a point instead of just at the point. Evaluation showed that the new measure for EI lead to the new algorithm discovering the global maximum more often than the original algorithm.

Chapter 7

Future Work and Conclusion

While working on this project there were a few things which I would have liked to have explored further. This chapter considers the combination of EI of the best mean with the future aware EI. It explains a problem with the predictive distribution which showed once multiple samples were taken in close proximity. Lastly it discusses an idea about using the improvement of the Gaussian process mean over the entire distribution rather than at a single point.

7.1 Combined Expected Improvement

The results for EI of the best mean and future-aware optimization were both positive, it would be a good idea to combine them. The future-aware EI was built using the original measure for EI, it could be done using the EI of the best mean instead. Using the combination should show even better results as the EI of the best mean still doesn't consider the remaining samples when it decides where to sample next. Combining both will make a point good to sample at when it hasn't got samples around it and will lead to higher EI in the future.

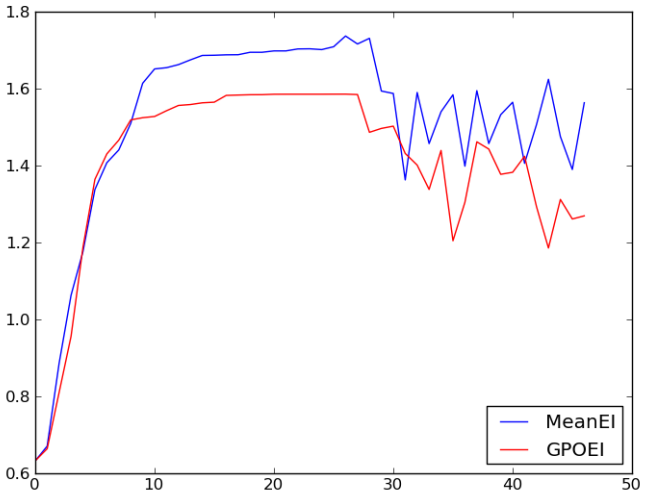


Figure 7.1: The results for a function where the GP model was getting some bad predictive distributions.

7.2 Gaussian Processes Problem

Something I noticed while doing my experiments was that once a maximum has been found and sampled at for a long time the predictive distribution started to be incorrect. I think the learned hyper-parameters were not very good and this showed in the results because the best point would spike up and down between 2 levels quite frequently. Figure 7.1 shows how the results would commonly look once many samples had been taken. To visualize this better I modeled a two dimensional function to see what the predictive distribution looked like.

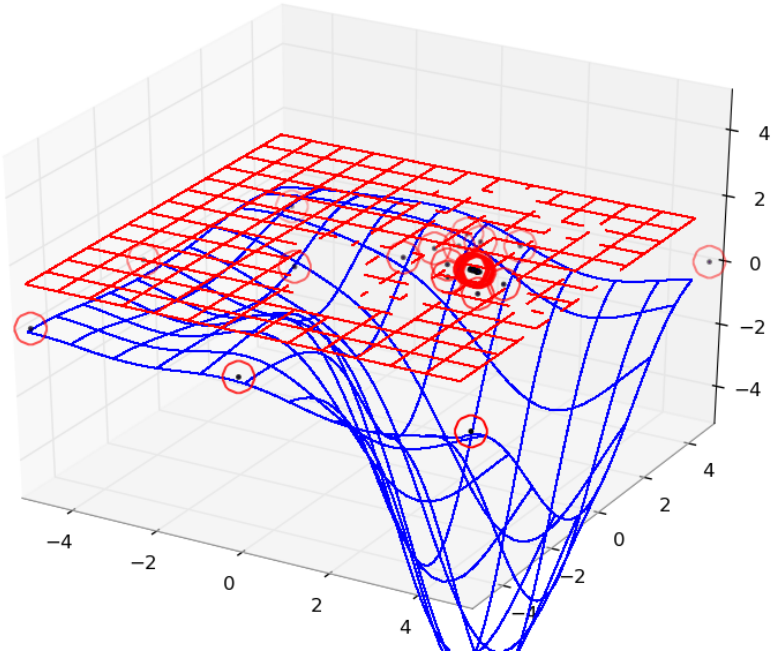


Figure 7.2: The blue surface is the predictive distribution, the red surface is the EI. Notice that the predictive distribution is very low near the point where most of the samples have been taken, this is quite wrong.

Figure 7.2 shows the bad predictive distribution which doesn't model the data very well at all. I believe the cause of this was when too many samples got taken at nearly the same place. The next step in improving the GPO algorithm should probably look closely at the step where the hyper-parameters are learned from the data.

7.3 Expected Improvement of Entire Predictive Distribution's Mean

During the implementation for the EI of the best mean, I considered using the EI of the mean everywhere. Instead of just calculating the improvement of the mean at a single point, the measure could integrate over all x and find the EI of the predictive distributions mean everywhere. This should lead to a better measure than just using the EI of a single point. It is possible that a low sample at $\mathbf{x}_\star$ leads to a large improvement somewhere else in the predictive distributions mean. The EI of the best mean at $\mathbf{x}_\star$ would not be very high but the total EI would be high. I did some calculations to find out how easy it would be to integrate over all $\mathbf{x}$ and managed to find a formula which gives the change in the mean at any point given a sample point $\mathbf{x}_\star$ which has a height of $\mathbf{y}_\star$ To get this formula I started with $\mu_{new} - \mu_{old}$

and expanded it using the calculations for the mean from equation 2.4 and the new mean from equation 5.5.

$$\mu_{new}(\mathbf{x}') = \mathbf{k}^T \boldsymbol{M}\mathbf{y} + \kappa \mathbf{m}\mathbf{y} + \mathbf{k}^T \mathbf{m} y_\star + \kappa m y_\star \tag{7.1}$$
$$\mu_{old}(\mathbf{x}') = \mathbf{k}^T \boldsymbol{C}^{-1}\mathbf{y} \tag{7.2}$$
$$\Delta\mu(\mathbf{x}') = \mu_{new}(\mathbf{x}') - \mu_{old}(\mathbf{x}') \tag{7.3}$$

I then used the partitioned matrix equations (5.1 to 5.4), to reduce $\Delta\mu$ to a simpler form.

$$\Delta\mu(\mathbf{x}') = m\,(\kappa - \mathbf{k}^\star \boldsymbol{C}^{-1}\mathbf{k})(\mathbf{y}_\star - \mathbf{k}^\star \boldsymbol{C}^{-1}\mathbf{y}) \tag{7.4}$$
$$= m\,(\kappa - \mathbf{k}^\star \boldsymbol{C}^{-1}\mathbf{k})(\mathbf{y}_\star - \mu_{old}(\mathbf{x}_\star)) \tag{7.5}$$

where

$$\mathbf{k}^\star{}_i = Cov(\mathbf{x}_i, \mathbf{x}_\star)$$
$$\mathbf{k}_i = Cov(\mathbf{x}_i, \mathbf{x}')$$
$$\mathbf{x}_i \in \mathcal{D}$$
$$\kappa = Cov(\mathbf{x}_\star, \mathbf{x}')$$

This equation finds the change to the predictive distributions mean at any point($\mathbf{x}'$) given that we take a sample point $\mathbf{x}_\star$ with a height of $\mathbf{y}_\star$. It seems like it would be possible to calculate the integral over $\mathbf{x}'$ and $\mathbf{y}_\star$ for this equation and use it to create a better EI measure.

7.4 Conclusion

Gaussian processes are a powerful tool and can be used quite successfully as a predictive distribution for optimization. Using a background mean which maximizes the likelihood of the GP model can improve the usefulness of the predictive distribution. It does show slight disadvantages when dealing with simple functions which have a higher mean than zero. However these can be overlooked when you consider how much better it performs on complex functions and also functions with a mean below zero.

EI of the best point is a greedy optimization method which always trys to get higher than previous samples, this is not the most effective way to search. By using the EI of the best mean we encourage the algorithm to look for points which can have a bigger influence on the predictive distributions mean. I also altered the algorithm to find areas which contained high EI as this should lead to good future EI as well. These changes lead to significant improvements in finding the global maximum in difficult optimization problems.

Bibliography

Osborne, M., Garnett, R. & Roberts, S. (2009), *Gaussian Processes for Global Optimization*, 'Learning and Intelligent Optimization Conference', volume 3, 2009.

Sasena, M., Papalambros, P. & Groovaerts, P. (2000), *Meta Modeling Sampleing Criteria in a Global Optimization Framework* , 'Proceedings of the 8th AIAA/NASA/USAF/ISSMO Symposium on Multidisciplinary Analysis and Optimization'.

Jones, D., Schonlau, M. & Welch, W. (1998), *Efficient Global Optimization of Expensive Black-Box Functions*, 'Journal of Global Optimization', **13**(4), pp 455-492.

Frean, M. & Boyle, P. (2009), *Using Gaussian Processes to Optimize Expensive Functions* , 'Proceedings of the 21st Australasian Joint Conference on Artificial Intelligence: Advances in Artificial Intelligence', **5360** of LNAI, pp. 258-267.

Frean, M. (2010), *More Approximately Optimal Optimization*

MacKay, D. (2003), *Information Theory, Inference, and Learning Algorithms*, 'Cambridge University Press', pp 535-546.

Rasmussen, C. & Williams, C. (2006), *Gaussian Processes for Machine Learning*, MIT Press, pp 43-44, 192.

Molga, M. & Smutnicki, C. (2005), *Test functions for optimization needs*

Sudhanshu, M. (2006), *Some new test functions for global optimization and performance of repulsive particle swarm method*

Koza, J. (1992), *Genetic Programming, On the Programming of Computers by Means of Natural Selection*, MIT Press

Kirkpatrick, S. & Gelatt, C. D. & Jr. & Vecchi, M. P. (1983), *Optimization by simulated annealing*,'Science', **220**, 4598 pp 671-679.

Bellman, R. (1969), *Adaptive control processes: a guided tour*, Princeton University Press