Project Report - Gaussian Processes for Global Optimization

Martin Hjelm - mar.hjelm@gmail.com - Jong-Won Han - addnull@gmail.com

May 6, 2011

1 Introduction

Optimization is a recurrent problem in the field of engineering, from mechanics and chemistry to economics. It deals with the problem of finding the optimal input or parameter values for a mathematically formulated problem. Usually the problem of optimization is expensive to solve in the terms of computational cost and sometimes a closed-form expression does not even exist. Adopting a Bayesian viewpoint and introducing a Gaussian process framework one can do away with the need of a closed form expression and rely solely on measurements of the function to be optimized. Applying a loss function for the function evaluation and computing the expected loss, one arrives at a analytical expression for a suggested point, which can be readily optimized using normal optimization techniques. The problem of optimization is thus transformed from a function evaluation perspective to a much simpler expected loss evaluation perspective.

This report is based upon a paper by Osborne, Garnett and Roberts - Gaussian Processes for Global Optimization[1].

2 Gaussian process framework

A Gaussian process is a stochastic process such that the vector of random values has a multivariate normal distribution. The Gaussian process is uniquely defined by its mean vector and covariance matrix. We define a Gaussian process over the desired function to be optimized by

$$p(y|x, I) \equiv \mathcal{N}(\, y|\mu, \mathrm{K}) \equiv |2\pi\mathrm{K}|^{-\frac{1}{2}}\, e^{\,(-\frac{1}{2}(y-\mu)^T \mathrm{K}^{-1}(y-\mu))} \tag{1}$$

where I is the context, including prior knowledge of mean and covariance, K is the covariance matrix generated by the specific covariance function and μ is the mean vector.

The testing functions being super positions of periodic and non-periodic elements, makes it reasonable to model the covariance function in the same manner. We thus define it as the sum of two isotropic squared exponential functions where one of the exponentials contain a periodic term in form of a sinus function

$$K(x, x') \equiv h_A^2\, exp\,(-r_A^2/2) + h_B^2\, exp\,(-r_B^2/2) \tag{2}$$

where

$$r_A^2 \equiv \Sigma_D (x - x')^2 / w_A^2 \qquad r_B^2 \equiv \Sigma_D (sin\,\pi(x - x')/w_B)^2 \tag{3}$$

and where h_A and h_B are the output scales, w_A and w_B are the input scales and Σ_D is the empirical covariance.

If we denote the hyperparameters of the model by θ, including the input-output scales and a constant prior mean μ, and let I_0 be the conjunction of I and previous observations $(\mathbf{x}_0, \mathbf{y}_0)$ and finally taking θ to be known we can write the predictive distribution for x_* as

$$p(y_*|x_*, \theta, I_0) = \mathcal{N}(\, y_*|m_\theta(y_*|I_0), \mathrm{C}_\theta(y_*|I_0)\,) \tag{4}$$

where

$$m_\theta(y_*|I_0) = \mu_\theta(x_*) + K_\theta(x_*, x_0)\, K_\theta(x_0, x_0)^{-1}(y_0 - \mu_\theta(x_0)) \tag{5}$$

$$\mathrm{C}_\theta(y_*|I_0) = K_\theta(x_*, x_*) - K_\theta(x_*, x_0)\, K_\theta(x_0, x_0)^{-1}\, K_\theta(x_0, x_*) \tag{6}$$

2.1 Marginalization

In most cases θ is not known and we therefore must resort to marginalization or approximation techniques. Marginalization of θ using Bayes theorem gives us

$$\begin{aligned} p(y_*|x_*, I_0) &= \int p(y_*|x_*, \theta, I_0)\, p(\theta|I_0)\, d\theta \\ &= \left\{ p(\theta|I_0) = \frac{p(y_0|x_0, \theta, I)\, p(\theta|I)}{p(y_0|x_0, I)} \right\} \\ &= \frac{\int p(y_*|x_*, \theta, I_0) p(y_0|x_0, \theta, I) p(\theta|I) d\theta}{p(y_0|x_0, I)} \\ &= \frac{\int p(y_*|x_*, \theta, I_0) p(y_0|x_0, \theta, I) p(\theta|I) d\theta}{\int p(y_0|x_0, \theta, I) p(\theta|I) d\theta} \end{aligned} \tag{7}$$

However the marginalization of θ leads in our case to a non-analytical integral and hence we resort to Bayesian Monte Carlo(BMC) techniques for approximating the marginalization integrals[2].

For a set of sample points, $\phi_S = [\phi_1, ..., \phi_n]$, we evaluate the marginalization terms

$$q(\phi) = p(y_*|x_*, \theta, I_0) \tag{8}$$

$$r(\phi) = p(y_0|x_0, \theta, I) \tag{9}$$

Viewing this as samples of q and r, and thus uncertainties, we can assign a $\mathcal{GP}$ prior to these functions, which enables us to perform regression for $q(\phi_*)$ for any ϕ_* as well as for $r(\phi_*)$. We assign a square exponential covariance function to the $\mathcal{GP}$

$$K(\phi, \phi') = \mathcal{N}(\, \phi|\phi', w^T w) \tag{10}$$

and a Gaussian prior distribution for each hyperparameter

$$p(\phi|I) = \mathcal{N}(\, \phi|\nu, \lambda^T \lambda) \tag{11}$$

The BMC evaluation leads after some approximations and tricks[2] to the following expression for the predictive distribution

$$p(y_*|x_*, I_0) \simeq \mathbf{q}_S^T \frac{K(\phi_s, \phi_S)^{-1}\, \Pi_S\, K(\phi_s, \phi_S)^{-1} \mathbf{r}_S}{\mathbf{1}_{S,1}^T K(\phi_S, \phi_S)^{-1}\, \Pi_S\, K(\phi_S, \phi_S)^{-1} \mathbf{r}_S} \tag{12}$$

where for $i, j \in \mathcal{I}_S$

$$\Pi_S(\phi_i, \phi_j) \equiv \mathcal{N}\left(\begin{array}{cccc} \phi_i| & \nu, & \lambda^T\lambda + w^T w & \lambda^T\lambda \\ \phi_j| & \nu, & \lambda^T\lambda & \lambda^T\lambda + w^T w \end{array} \right)$$

This can be seen as a Gaussian process mixture model, the equation being a linear combination of the elements in $\mathbf{q}_S$. And finally we can write

$$p(y_*|x_*, \theta, I_0) \simeq \sum_{i \in S} \rho_i\, \mathcal{N}(\, y_*|m_i(y_*|I_0),\, \mathrm{C}_i(y_*|I_0)\,) \tag{13}$$

The last problem is how to speed up the calculations of the inverses for the weights. Having a large number of samples for our hyperparameters will lead to quite large matrices and a heavy load of Cholesky decompositions. However if we take the priors as independent and let the covariance structure be given by the product of terms over each hyperparameter, the product correlations rule will give

$$K(\phi, \phi') = \prod_e K_e(\phi_{(e)}, \phi'_{(e)}) \tag{14}$$

Finally if we make a grid of the samples, such that ϕ_S is the tensor product of a set of samples $\phi_{(e),S}$ over each hyperparameter, we can write the inverse covariance matrix product term in equation 12 as the Kronecker product of the equivalent term over each individual hyperparameter [3] and we get

$$\begin{aligned} K(\phi_S, \phi_S)^{-1}\, \Pi_S\, K(\phi_S, \phi_S)^{-1} = \\ K(\phi_{(1),S}, \phi_{(1),S})^{-1}\, \Pi_{(1),S}\, K(\phi_{(1),S}, \phi_{(1),S})^{-1} \\ \otimes K(\phi_{(2),S}, \phi_{(2),S})^{-1}\, \Pi_{(1),S}\, K(\phi_{(1),S}, \phi_{(1),S})^{-1} \\ \otimes ... \end{aligned} \tag{15}$$

So instead of having to calculate the Cholesky decomposition of a giant covariance matrix we are left with computing decompositions for every sampled parameter's covariance matrix. Of course the Kronecker product will be quite large and heavy to compute but in comparison we gained a lot.

3 Optimization

The basic idea of the optimization is to define a loss function and use it with our mixture model in order to evaluate the expected loss after having taken every function sample except for the last one. This will give us an analytical expression that will let us find the expected loss minimum by simply using the gradient.

3.1 The loss function

If we let η be the minimum of the function evaluations so far we can write $\eta \equiv min\ \mathbf{y_0}$ and define the loss function in the following way

$$\lambda(y) = \begin{cases} y \text{ if } y \leq \eta \\ \eta \text{ if } y \geq \eta \end{cases} \tag{16}$$

Basically the loss function will penalize if nothing was gained using current knowledge and in the case of a new minimum, the gain will be the difference between current knowledge and the new minima.

3.2 The expected loss

We define the expected loss as usual

$$\Lambda_1(x|I_0) \equiv \int \lambda(y)\, p(y|x, I_0) = \sum_{i \in S} \rho_i \mathrm{V}_i(x|I_0) \tag{17}$$

where

$$\begin{aligned} \mathrm{V}_i(x|I_0) \equiv \eta \int_{\eta}^{\infty} \mathcal{N}(y|m_i, \mathrm{C}_i)\ dy + \int_{-\infty}^{\eta} y\mathcal{N}(y|m_i, \mathrm{C}_i)\ dy = \\ = \eta + (m_i - \eta)\Phi(\eta|m_i, \mathrm{C}_i) - \mathrm{C}_i\mathcal{N}(\eta|m_i, \mathrm{C}_i) \end{aligned} \tag{18}$$

and Φ is the normal cdf, and m_i and C_i are dependent on the vector x. The expression above has a analytical expression for the gradient and we thus use a simple gradient descent to find a minima.

3.3 Optimization - minimizing the expected loss

To calculate the gradient of equation 18 we just take derivate w.r.t. x since it is not explicit dependent on x and extrapolate for multidimensional vectors.

After some tedious calculations we get

$$\frac{d}{dx}\mathrm{V}_i(x|I_0) = m_i'\Phi_i + \frac{1}{C_i^2}(\,(C_i'^2 - 1)C_i(\eta - m_i)m_i' + C_i'(C_i - 1)(\eta - m_i)^2\,)\,\phi_i \tag{19}$$

where $\Phi_i = \Phi(\eta|m_i, \mathrm{C}_i)$ and $\phi_i = \mathcal{N}(\eta|m_i, \mathrm{C}_i)$.

We proceed to calculate the derivatives of m_i and C_i with respect to x_* for the expression of the mean and covariance.

$$\frac{\partial m_i(y_*|I_0)}{\partial x_*} = \frac{\partial \mu_i(x_*)}{\partial x_*} + \frac{\partial \mathbf{K}_i(x_*, \mathbf{x}_0)}{\partial x_*}\,\mathbf{K}_i(\mathbf{x}_0, \mathbf{x}_0)^{-1}(y_0 - \mu_i(\mathbf{x}_0)) \tag{20}$$

$$\frac{\partial C_i(y_*|I_0)}{\partial x_*} = -\frac{\partial \mathbf{K}_i(x_*, \mathbf{x}_0)}{\partial x_*}\mathbf{K}_i(\mathbf{x}_0, \mathbf{x}_0)^{-1}\mathbf{K}_i(\mathbf{x}_0, x_*) + - \mathbf{K}_i(x_*, \mathbf{x}_0)\mathbf{K}_i(\mathbf{x}_0, \mathbf{x}_0)^{-1}\frac{\partial \mathbf{K}_i(\mathbf{x}_0, x_*)}{\partial x_x} \tag{21}$$

where the derivative of the K_i matrices is just the derivative of the covariance function for each element giving

$$\frac{\partial K(x_*, x)}{\partial x_*} = -\frac{h_A^2}{w_A^2}(x_* - x)\Sigma_D e^{-r_A^2/2} - \frac{h_B^2}{w_B^2}\pi\, sin\,\pi(x_* - x) cos\,\pi(x_* - x)\Sigma_D e^{-r_B^2/2} \tag{22}$$

also

$$\frac{\partial \mathbf{K}_i(x, x_*)}{\partial x_*} = -\frac{\partial \mathbf{K}_i(x_*, x)}{\partial x_*} \tag{23}$$

If we would have a multidimensional vector the only thing that changes in the derivates w.r.t. some specific dimension are the explicit x_*'s in 22.

By the anti-symmetry of equation 23, equation 21 becomes zero and equation 19 reduces to

$$\frac{d}{dx}\mathrm{V}_i(x|I_0) = m_i'\,(\Phi_i + \frac{(\,m_i - \eta\,)}{C_i}\,\phi_i) \tag{24}$$

As we can see from equation 24 the most heavy calculations involve the computation of the normal CDF, the rest is just calculations of the predictive mean and covariance and the primed predictive mean for every sample parameter combination.

4 Implementation

The basic idea of our algorithm was to choose a number of random sample points, calculate the mixture weights and then use steepest descent to find the minima. The implementation seemed pretty straightforward however it turned out there were several pitfalls.

4.1 Multivariate probabilities

Equation 12 involves the calculation of the likelihood of the given data. This is often lesser than machine epsilon and we get division by zero, a feat which only God and Chuck Norris can pull off. The usual log-trick does not work since if the $K^{-1}DK^{-1}$ is a m-by-n matrix then the map can be seen as just m scalar products. However the *log* of a scalar product can't be tampered with since $\log(a \cdot b) = \log(a_1b_1 + a_2b_2 + ...)$. Further on if we take the *log* of the individual elements we get $\log a_1 \log b_1 + \log a_2 \log b_2 + ...$ which in this case is not feasible since b in our case contains negative elements. Even if b just contained positive element there is no easy way of transforming it back since in general

$$\log(a_1b_1 + a_2b_2 + ...) \neq \log a_1 \log b_1 + \log a_2 \log b_2 + ... \tag{25}$$

and we thus can't take the *exp* to transform back.

The solution to harvesting some non-zero values at least is to take the log probability and deduct the max of it and transform back, this is possible since equation 12 have the dot product of the likelihood both in the denominator and numerator and we can cancel the factor that we subtract. The likelihood thus decides which hyperparameter sample vectors are interesting and which are not.

4.2 The Kronecker product

The Kronecker product for equation 15 was calculated using the Cholesky decomposition. Since K is spd by definition we can take the Cholesky decomposition and calculate the inverse product in the following manner.

$$K = LL^T \tag{26}$$

$$K^{-1}BK^{-1} = A \Rightarrow B = KAK \Rightarrow (L\backslash(L^T\backslash(L^T\backslash(L\backslash B))^T))^T = A \tag{27}$$

This save a lot of calculation time in comparison to using Matlab's inversion algorithm.

The Kronecker product also saves us a lot of calculations in that we don't have to solve a giant matrix system but can concentrate on solving smaller individual systems. The Cholesky decomposition scales roughly as n^3 so if we would have 10 hyperparameters and take 10 samples of each of them we will have a 10^{10} matrix to solve but if we divide it we get 10 matrices of size 10 to solve; this means 10^{25} operations plus the cost for the Kronecker product calculations and 10^{60} operations for the original system.

4.3 Setting the hyperparameters

In the original paper no extensive information was given of how to set the hyperparameters. Especially the mean and the prior weights for the hyperparameter samples were particularly difficult to pick out since the $\mathcal{GP}$-mixture weights would either become infinity or NaN.

The prior mean Its common to put the prior mean to zero in ordinary $\mathcal{GP}$ inference, however the result is much more accurate if some knowledge of the mean can be incorporated. The paper referred to a constant prior mean but gave no clues as how to choose it, so we resorted to trial and error. In the beginning of our algorithm we choose a number of random sample points to base the weight calculations upon. Putting the prior mean to the observed mean is typically a bad approach since the points chosen could be in areas with peaks or lows. Choosing the prior mean to be zero worked but was not really satisfactory especially for strictly positive or negative functions. The approach that seemed to work the best was to set the mean to the mean of the max, min and median value of our sampled points. This gave us bit more range to play with. There was also an idea to weight the median a bit more but we didn't have time to try it out.

Hyperparameter sample weights & hyperparameters The exponential of the sinus term turned out to be difficult to tune so we decided to opt out and just set its hyperparameters to zero, the algorithm seemed to make good decisions without it. We also scrapped the Σ_D term for the covariance function. There was no mention in the paper as to what it represented, we suspected it to be the empirical covariance of the observed points but as above with the mean that seemed like a bad approach since we could have sampled in a low change region or in a high change region; so we removed that and let the range of the prior h_a have a wider range that would be adjusted according to the function we were working with.

The mixture weights were set through trial and error rather than having some idea of how we would set them. We could conclude that we needed

h_a to be in a rather big range since it would represent the non-zeros in the likelihood vector, $\mathbf{r}$. A too low range would just give zeros and a too big range would be unrepresentative for the functions we were trying to model. The $\mathcal{GP}$ covariance weights were a though nut to crack since it was not apparent how they influenced the mixture weights. It seemed like high values would produce a highly oscillative $\mathcal{GP}$ mixture predictor and a too low value would just smooth it with dips for observed values. Some more pondering is obliviously needed to explain the behavior in more detail.

We set h_a to be equally spaced in the interval of $[0.6, 5.5]$ and w_a to the range $[0.1, 1.5]$ and the $\mathcal{GP}$ covariance weights in equation 10 to 0.15. The λ values in equation 11 were set to equal the covariance weight as argued in [2]. An easy way of testing what would be good values for h_a to make the expected loss work good turned out be to add a weighted covariance matrix of the observed values to the predictive covariance for the mixture and use that as feedback for how the range should be changed. Maybe that was what was intended with the Σ_D in the covariance function.

4.4 Optimization by steepest descent and a myopic strategy

We choose steepest descent to calculate the local minima. The steepest descent is not fastest method and has several limitations, slow convergence from zig-zag and finding a good learning rate being the major ones. However it is easy to implement when the gradient is known and time is of the essence, or you just want to try out the solutions.

We tried two approaches one where we would just choose n initial random points in the domain and then use the steepest descent to minimize the expected loss or the myopic approach mentioned in [1], of choosing one or two initial points, minimize the expected loss, and choose that point as the next evaluation point and so on for n points.

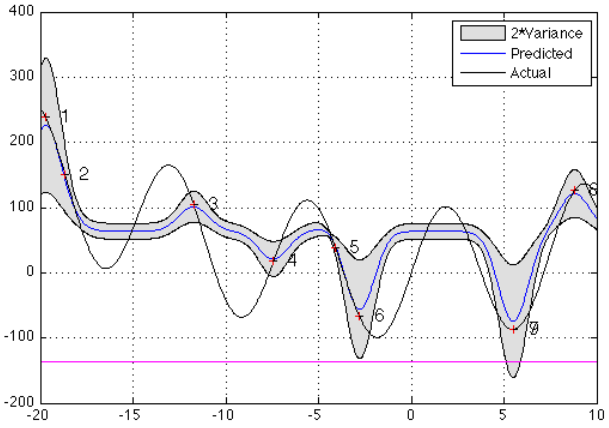
We also tried the approach of using the TOMLAB toolbox for minimizing the expected loss however without incorporating the gradient or hessian.

5 Results

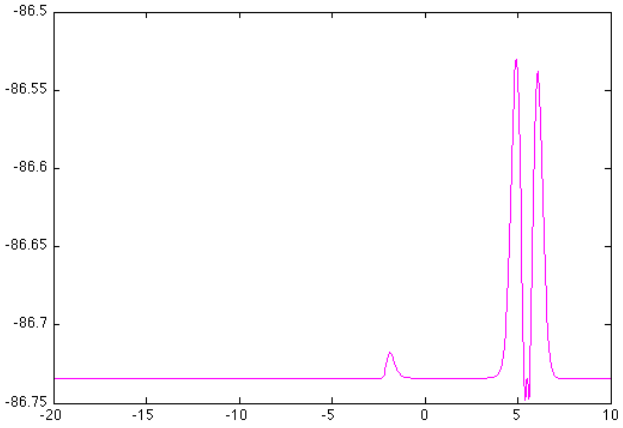
5.1 Testing in for one-variable functions

The testing yielded some interesting insights. First of all that the derivative of the expected loss was zero or near to zero for the most part and particularly dependent on the placement of the evaluated points for success. For example if most of the points were selected at high points above the minima

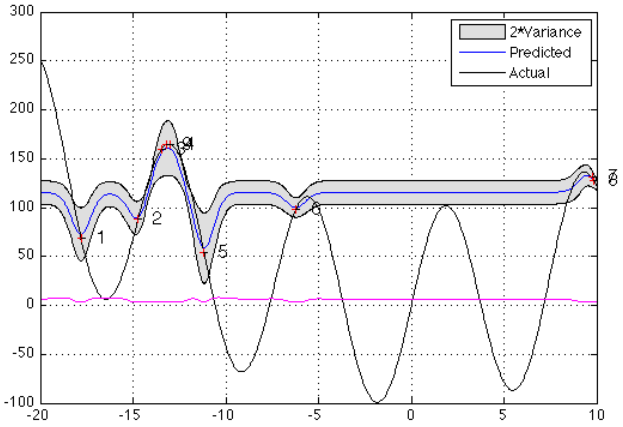
the expected loss function turned out almost flat. However the approach of selecting initial points at random is the best you can do if you have no prior knowledge of the function at hand, you just have to pray that they land in areas where the information gain is high. With the derivative close to zero the steepest descent method failed miserably in fact is just much faster to calculate the expected loss over the interval and then take minimum, unless we are are lucky as in (a) of figure 1. Of course instead of choosing the starting point at the minimum for the observed set we could have started between the minimum and its closest point or some other more exploratory strategy. However it seems like just doing some evaluation along the axis is more easy and fast.



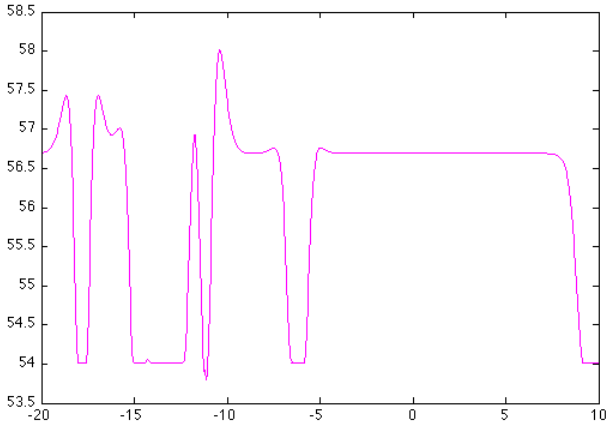
(a) Predictive mean for the $\mathcal{GP}$-mixture



(b) Expected loss



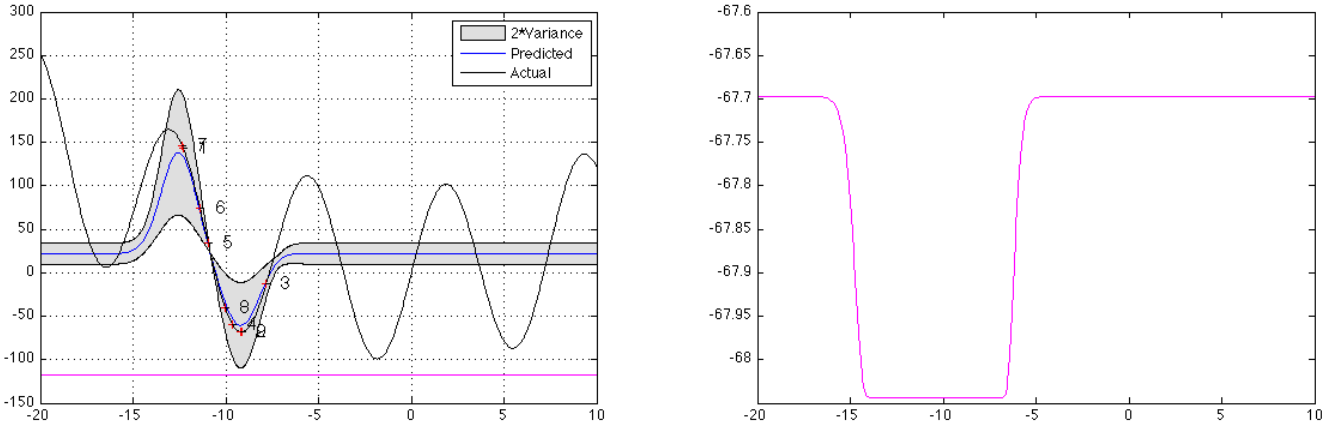
(c) Predictive mean for the $\mathcal{GP}$-mixture



(d) Expected loss

Figure 1: The plots shows the prediction of the $\mathcal{GP}$-mixture with random points in red, the last number being the selected point for evaluation by the algorithm. The right hand side shows the expected loss in a better scale.

The myopic approach based upon choosing the next evaluation points from the minimum of the expected loss turned out to be quite a bad approach. Most of the time the expected loss function would choose the points to be explored just next to the minima. Mostly this seemed to be due to the poor performance of the steepest descent method since the expected loss would just be zero at long stretches as can be seen in figure 2.



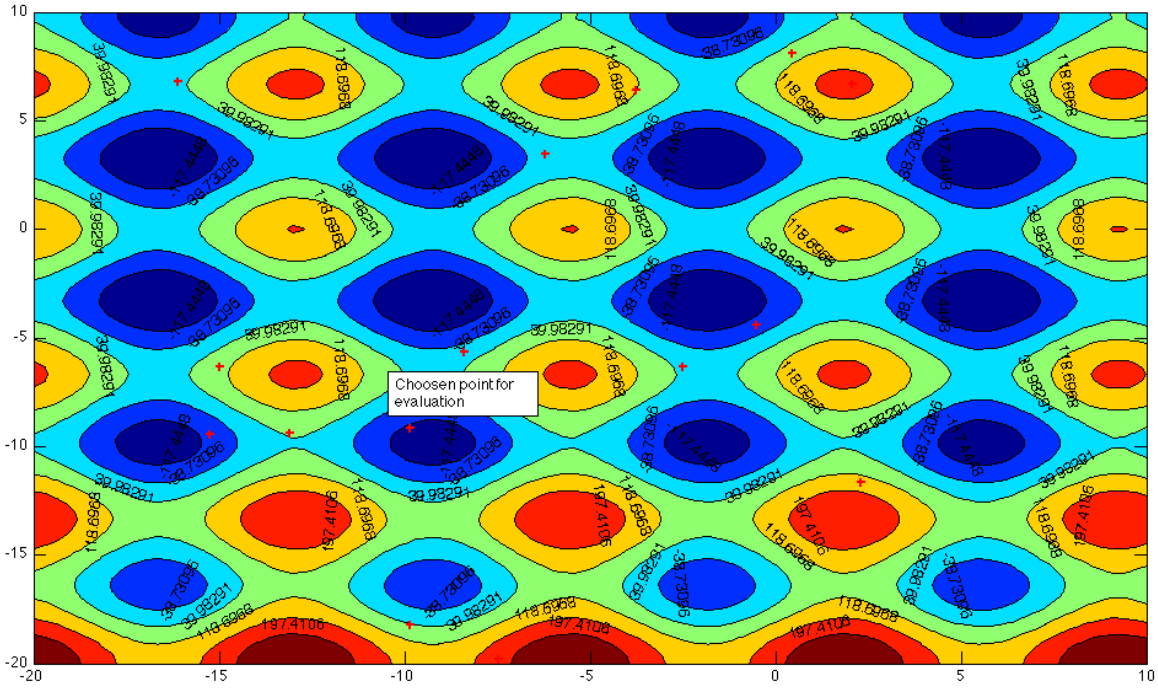
(a) Predictive mean for the $\mathcal{GP}$-mixture

(b) Expected loss

Figure 2: The myopic approach.

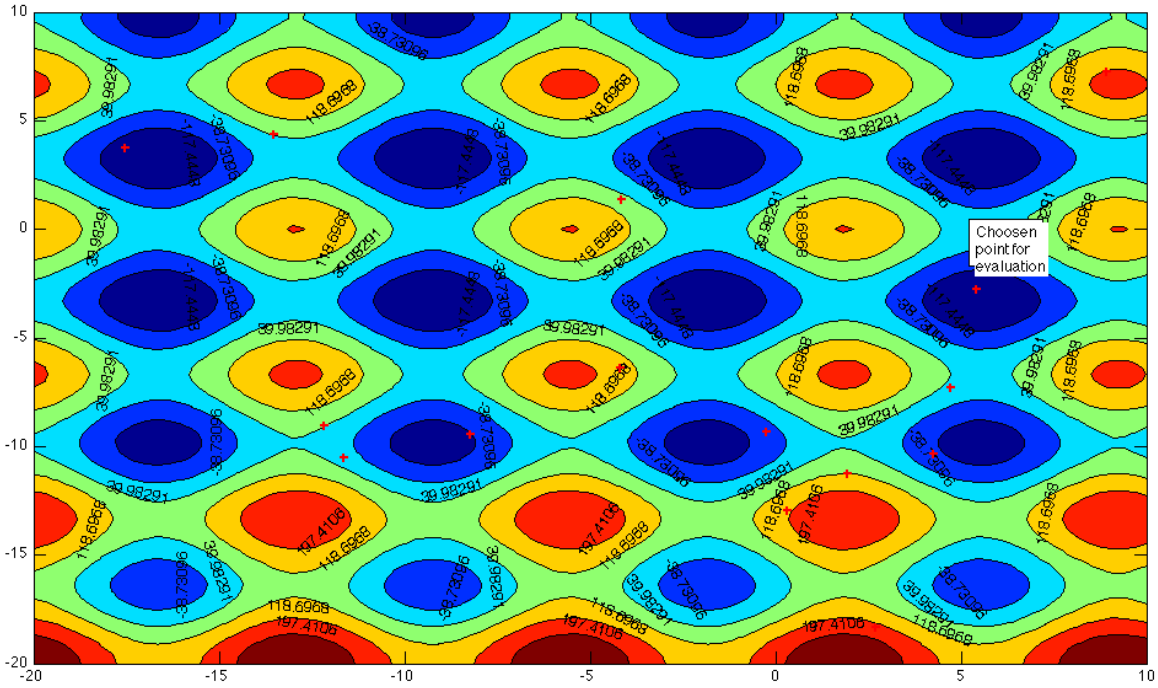
5.2 Testing in 3-d

For two variables the steepest descent typically works better, since we have two directions to go in i.e. zero gradient in one direction does not mean an overall status quo. Since we are covering a larger area we need more sample points and the myopic approach becomes to slow, especially for the starting points since we have to cover large areas to find a minima.



(a) Predictive mean for the $\mathcal{GP}$-mixture

Figure 3: Gradient descent for the expected loss function for our test function. Red dots are the sampled points and the white box shows the chosen point for our last evaluation.



(a) Expected loss

Figure 4: Gradient descent for the expected loss function for our test function. Red dots are the sampled points and the white box shows the chosen point for our last evaluation.

5.3 Testing on the benchmark functions

We have compared the results of our method, Gaussian Process Global Optimization (GPGO), with several proposed alternatives on an extensive set of standard test functions for global optimization.
To compare the various optimization methods, we chose a diverse list of five standard test functions (see Table 1) for global optimization methods. Each standard function additionally has a commonly used "standard" box-shaped test region containing the global minimum.
The implementations used for the efficient global optimization (EGO), the radial basis function (RBF), and the dividing rectangles (DIRECT) optimization algorithms were those in the TOMLAB/CGO toolbox version 7.6 (Nov. 2010) for MATLAB, version 7. The default settings were used for each, except that the number of function evaluations used in the initial samples (from which the methods derive their hyperparameters) was ten.

Because the objective functions we are considering are assumed to be very expensive to evaluate, we limit the number of allowed function evaluations. For very expensive functions, limiting the number of function evaluations is a natural stopping criterion.
Testing GPGO was performed ten times with 10, 30 and 50 samples. Because the ten testing has similar meaning with the ten evaluation of TOMLAB/CGO we chose the best result of the ten testings for each problem. Finding the minimum of the expected loss function was achieved by DIRECT optimization
We use the "gap" measure of performance to compare the method

$$G \equiv \frac{y(x^{(first)}) - y(x^{(best)})}{y(x^{(first)}) - y^{(opt)}} \tag{28}$$

where y is the objective function, $y^{(opt)}$ is the global optimum, and $x^{(first)}$ and $x^{(best)}$ are the first point evaluated and best point found at termination, respectively. For every method tested, the initial point $x^{(first)}$ was chosen to be the medial point of the test region.

5.4 Discussion

The results of testing are shown in Table 1. Because the result of GPGO depends on the samples and they are picked randomly, GPGO can sometimes generate a poor result. In Ackley function case, the result of 10 samples is better than 30 and 50 samples. Nevertheless, GPGO performed (or tied with) the best out of the three algorithms, EGO, RBF and DIRECT tested for five problems. Additionally, the grand mean performance of our method (even for the simplest model and least informative prior) surpasses the other methods by 9%. Hence it is reasonable that GPGO provides an improvement over other state-of-the-art solutions for global optimization of expensive functions.

Improvements that would certainly yield better results for our model are some of the improvements suggested in the original paper, incorporating the periodic covariance, using the derivative observations for the covariance, deriving some scheme for choosing points or just developing the more advanced myopic approach described in the original paper. There is also quite a lot to be done with adjusting the hyperparameters and coming up with some scheme for the prior mean. Typically you would want some table of good combinations, depending on the function at hand. The random samples chosen to calculate the weights could certainly be chosen with some better strategy, like some polynomial least square solution for where to choose the n samples, a typically myopic approach but without needing the heavy

calculations.

Table 1: Mean measure of performance G for various global optimization techniques. The first table presents results over test functions, the second table is the list of function name and test region.

				GPGO		
	EGO	RBF	DIRECT	10 samples	30 samples	50 samples
Br	0.911	0.966	0.915	0.984	0.986	**1.000**
G-P	**1.000**	**1.000**	0.986	**1.000**	**1.000**	**1.000**
Shu	0.242	0.147	0.175	0.380	0.688	**0.897**
G2	0.991	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
A2	0.727	**1.000**	**1.000**	**1.000**	0.622	0.601
mean	0.774	0.823	0.815	0.873	0.859	**0.900**

Function	Full Name	Test Region
Br	Branin	$[-5, 10] \times [0, 15]$
G-P	Goldstein-Price	$[-5, 5]^2$
Shu	Shubert	$[-10, 10]^2$
G2	Griewank 2	$[-600, 600]^2$
A2	Ackley 2	$[-32.8, 32.8]^2$

References

[1] Osborne, Garnett, Roberts, *Gaussian Processes for Global Optimization.* Department of Engineering Science, University of Oxford.

[2] Osborne, Roberts, *Gaussian Processes for Prediction - Technical Report PARG-07-01.* Department of Engineering Science, University of Oxford.

[3] Osborne, Roberts, Rogers, Ramchurn, Jennings, *Towards Real-Time Information Processing of Sensor Network Data using Computationally Efficient Multi-output Gaussian Processes.* Department of Engineering Science, University of Oxford and University of Southampton.