

EPSRC

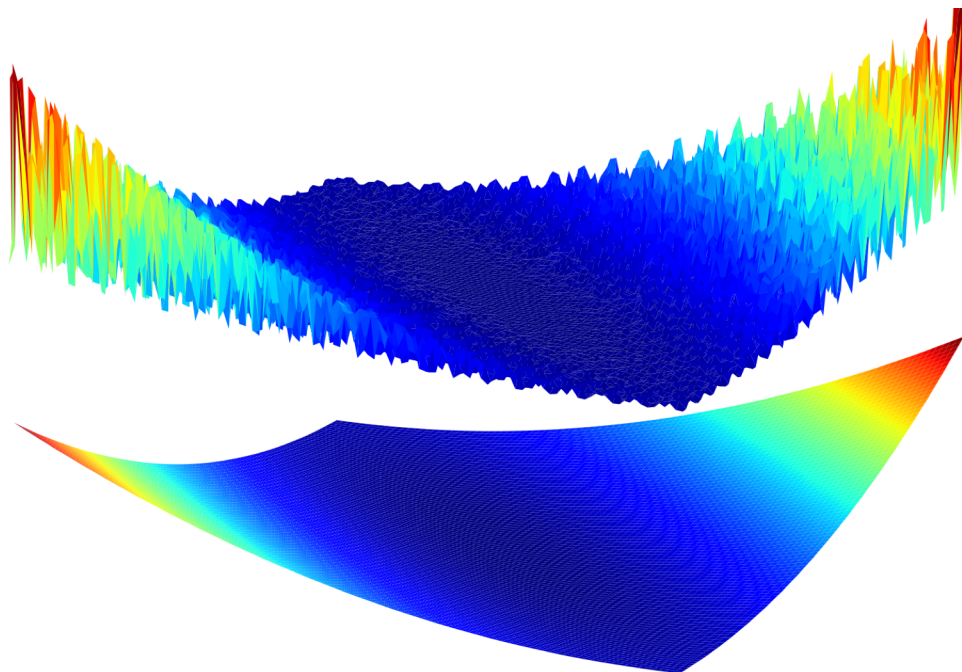
Engineering and Physical Sciences
Research Council



InFoMM

Industrially Focused
Mathematical Modelling

EPSRC Centre for Doctoral Training in Industrially Focused Mathematical Modelling



Bayesian Optimisation

Oliver Sheridan-Methven



UNIVERSITY OF
OXFORD

nag[®]



Contents

1. Introduction.....	2
2. Bayesian optimisation.....	2
Glossary of terms.....	3
3. Optimisation algorithms.....	4
4. Results.....	4
Noiseless objective functions.....	4
Noisy objective functions.....	5
Application to a real-world example.....	6
5. Discussion, Conclusions & Recommendations.....	6
6. Potential Impact.....	7

1. Introduction

Frequently in industry, academia, and everyday life, we encounter situations where we would like to find the best solution for a problem. This could be how to best allocate stocks in a financial portfolio so as to minimise overall risk exposure, or even what parameter values give the fastest code performance for producing random numbers or solving a system of differential equations.

For a number of problems it is well understood how to achieve the best results. If we have some measure of performance and know how it depends on the input parameters, then we can solve for stationary points. We select either the minima or the maxima, depending on whichever is sought after. However, solving for stationary points relies on knowing how the output changes as we vary the input, which corresponds to being able to compute *derivatives*. With the increase in evermore complicated models, larger datasets, and noisier data, either computing the derivative is too expensive, or completely inaccessible.

Frequently when training a neural network, or any machine learning algorithm, the model produced may contain thousands (possibly millions) of variables, and there is no chance to understand even how the model works. These types of models are often called *black-box functions* and are typically noisy and very expensive to compute.

Black-box functions work by some unknown mechanism. There is no access to derivative information, and they are frequently noisy, and expensive to evaluate.

As these black-box functions are usually very noisy, the machine learning and computational statistics communities are frequently trying to fine-tune and improve the parameters governing these models, in a process known as *hyper parameter optimisation*. Frequently, the parameter configuration is only the best when compared to similar configurations, in which case the configuration is known as a *local* optima. However, given the models are frequently noisy, we would like to do better than these local and somewhat “blinkered” solutions, and try to achieve the best performing solution across all possible configurations, known as the *global* optima. There is the possibility there may be multiple such global optima.

NAG are a leading producer of software and high performance computing services. They are interested in investigating *Bayesian algorithms*, which are a subclass of global optimisation algorithms, to see if they should develop these for their commercial library. Our aim is to assess the performance of a selection of optimisation algorithms including Bayesian, global, and local methods on a variety of synthetic black-box functions, showcasing in what scenarios each class of algorithm proves more (or less) effective.

2. Bayesian optimisation

To tackle the emerging torrent of hyper parameter optimisation problems arising from the machine learning community, Bayesian optimisation methods have gained popularity. These are a sub-class of global optimisation algorithms more generally. In order to describe the principles underlying Bayesian algorithms, we first need to introduce the optimisation problem more formally.

At the centre of global optimisation is the objective function $f(\cdot)$, which measures the quantity we want to optimise by a single number (e.g. profit, accuracy, speed, etc.). By default we will assume the optima we desire is when the objective function is minimised. The objective function takes an argument \mathbf{x} from some search space, where we are trying to find the global optimiser \mathbf{x}^* such that $f(\mathbf{x}^*)$ is a global minimum.

Traditionally, global optimisers and derivative-free methods have made progress with these problems by forming evermore complex approximations of the objective function, either making assumptions about convexity, producing linear or quadratic models, or partitioning the search space into many smaller optimisation problems. A common feature of many of these approaches is they often involve multiple evaluations of the objective function with little regard to how expensive each query is. However, if the model is the result of training a facial recognition model or a large scale astrophysical simulation, then it could take hours



or even days to evaluate the function even once, making a large number of function queries undesirable.

The bulk of the computational effort when using Bayesian algorithms is spent determining where the next sample should be taken. The key is to prioritise making a few well-selected evaluations in favour of producing complicated models which require lots of evaluations. To achieve this, Bayesian algorithms involve using an acquisition function $a(\cdot)$ to determine where to sample next. The acquisition function is a combination of the expected function value $\mu(\cdot)$ and the uncertainty $\sigma(\cdot)$. Minima of the acquisition function typically correspond to either *exploring* regions with high uncertainty, or *exploiting* regions which show promise for containing the global optima. We illustrate how such a scheme works in Figure 1, where after n iterations the algorithm has sampled n points (solid markers). The point which optimises the acquisition function is computed (hollow markers). This point is then sampled, the acquisition function updated, and the process iterated until termination.

Bayesian algorithms use an acquisition function to determine which points to sample.

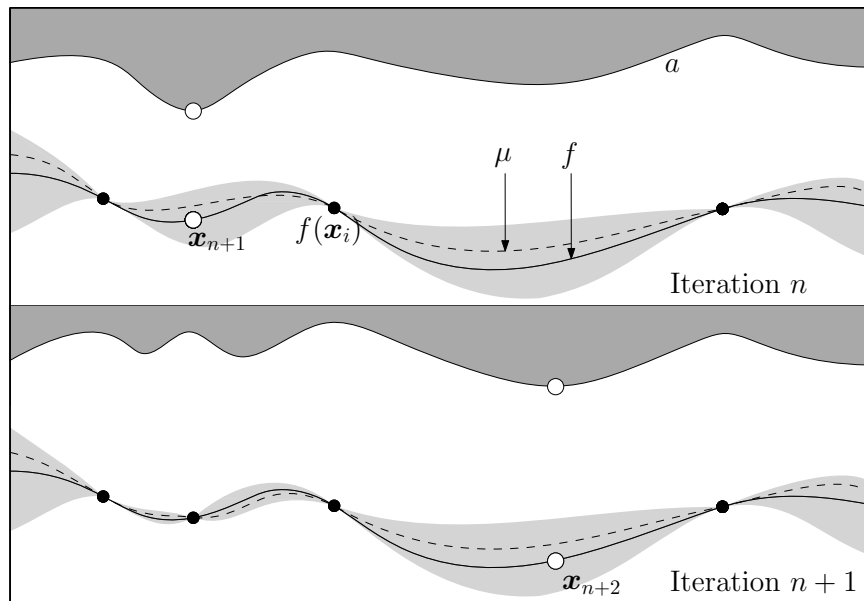


Figure 1: A schematic of the principle underlying Bayesian optimisation algorithms.

To construct the acquisition function, we must make assumptions about the objective function. As the function is usually noisy we use a probabilistic model, and the most popular choice is a Gaussian distribution with mean μ and variance σ^2 , which is known as a *Gaussian process* Bayesian algorithm. Alternative models include using *random forests*, *tree Parzen estimators*, and *regression trees*. Additionally a model for the covariance of the noise has to be assumed, and commonly used models are *exponential* distributions and *Matérn kernels*, where these covariance functions are generally called *noise kernels*.

Glossary of terms

- **Objective function:** The function we wish to optimise.
- **Black-box function:** A function where *nothing* is known about how it works or what output we might achieve for a given input. There is no access to derivative information, evaluations are expensive, and output may have noise.
- **Acquisition function:** The fundamental tool underlying Bayesian optimisation. It is an aggregate of the objective function's expected value and its uncertainty.
- **Noise kernel:** The model for the objective function's covariance structure.

Points are sampled based on optima of the acquisition function.

3. Optimisation algorithms

To investigate the performance of Bayesian optimisation algorithms, we need a collection of optimisation algorithms to compare. We pick 6 pre-existing solvers to use in our numerical experiments. These solvers involve a number of different mechanisms that we mention without explanation. The solvers are:

- **GPpyOpt**, a Bayesian algorithm using a Gaussian process.
- **HyperOpt**, a Bayesian algorithm using tree Parzen estimators.
- **PySMAC**, a Bayesian algorithm using random forests.
- **DIRECT**, a global algorithm which partitions the search space.
- **CMA-ES**, a global algorithm using a *covariance matrix adaptation* scheme.
- **BOBYQA**, a local derivative-free algorithm using linear or quadratic models.

As a proxy for real-world black-box functions, we used synthetic functions which were quick to evaluate and whose global minima were known.

To complement the solvers, we need a collection of challenging test problems. For practical purposes, it is preferable to use synthetic quick-to-compute functions rather than real-world expensive black-box functions. We used 29 problems selected from throughout the literature of global optimisation studies. Using a relative noise level ϵ between 10^{-3} and 10^{-1} , we made the functions noisy by introducing:

- **Deterministic noise.**
- **Multiplicative Gaussian noise.**
- **Multiplicative uniform noise.**
- **Additive Gaussian noise.**

4. Results

The performance of each optimisation algorithm is likely to vary between problems. A common measure taken to try and standardise a problem's difficulty is to measure the number of function evaluations scaled by the number of input variables D . We introduce the notion of the *computational budget* b which is the number of function evaluations divided by $D + 1$. Using this scale, the number of function evaluations required to compute a derivative corresponds to a computational budget of 1. In our experiments we permitted each solver a maximum computational budget of 100 for each problem, and we attempted each problem approximately 20 times from random starting points, and repeated this for each of the different noise models. The results we present are the performance of each algorithm averaged over all of its attempts.

The most informative measure available to assess how each of the solvers performed overall, and how they ranked amongst each other, is to look at the *data profiles* of the solvers. We use these to show what proportion of the test problems each algorithm solved on average for a given computational budget. To assess whether the final solution was approaching the global minima, and hence if the problem was "solved", we consider if the quoted solution is close enough to the known global solution, within some tolerance level τ relative to its first evaluation. We consider tolerance levels between 10^{-5} and 10^{-1} .

Noiseless objective functions

The data profiles for the noiseless objective functions are shown in Figure 2, alongside a histogram of the number of budgets used at termination. We can see from the histogram that, although we specified a computational budget allowance of 100, most of the solvers treated this more as a guideline rather than a strict terminating criteria. However, BOBYQA rarely went beyond a computational budget of 40.

Looking at the performance profiles between the different tolerance levels, we see that the solvers could solve the majority of the problems when $\tau = 10^{-1}$, but did not manage to solve more than half when $\tau = 10^{-5}$. This leaves $\tau \sim 10^{-3}$ as a suitable middle ground tolerance level. We can see in this regime that DIRECT and CMA-ES give the best performance and



scaling, while BOBYQA has a good performance for smaller budgets. Although the Bayesian solvers provide reasonable performances, they are never seen to be the best achieving nor the best scaling.

Data profiles show the average fraction of problems solved as we increase the allowance of function evaluations.

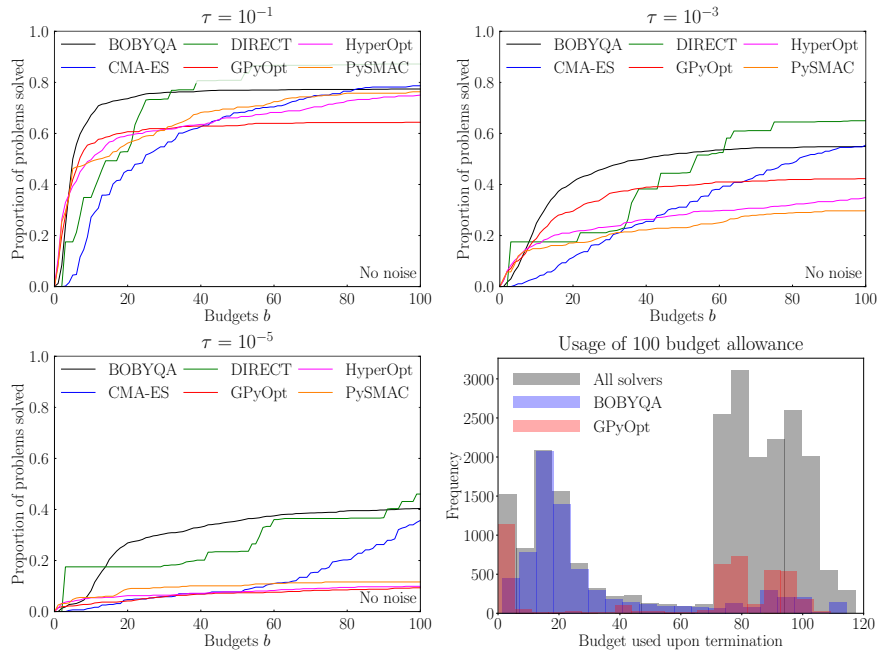


Figure 2: Data profiles for different tolerance levels. (Top left) $\tau = 10^{-1}$. (Top right) $\tau = 10^{-3}$. (Bottom left) $\tau = 10^{-5}$. (Bottom right) The computational budget used.

Noisy objective functions

When we made the functions noisy, the Java wrapper in PySMAC encountered technical issues, and so results for PySMAC are not shown. Similar issues were encountered by HyperOpt on the MNIST case study presented later.

We suspect the relative noise level $\epsilon = 10^{-1}$ gives too many false positives when testing for convergence, and propose noise levels $\epsilon \leq 10^{-2}$ as suitable.

We first introduce the results using multiplicative noise for the quite conservative tolerance level $\tau = 10^{-2}$, and we show the results in Figure 3, from which two key results emerge. Firstly, BOBYQA's performance plateaus earlier to a reduced level for a relative noise level ϵ between 10^{-2} and 10^{-1} . Secondly, HyperOpt's performance appears to increase. For increasing noise levels we intuitively expect the performances to either remain the same or decrease, but not improve. While this improvement is not colossal, we also observed this improvement for tolerances down to $\tau = 10^{-5}$. We conjecture that this is because when assessing if we converged on the global solution, the high noise levels produce an increased number of false positives when comparing against the global minima of the noiseless functions. These improvements were not observed for noise levels $\epsilon \leq 10^{-2}$.

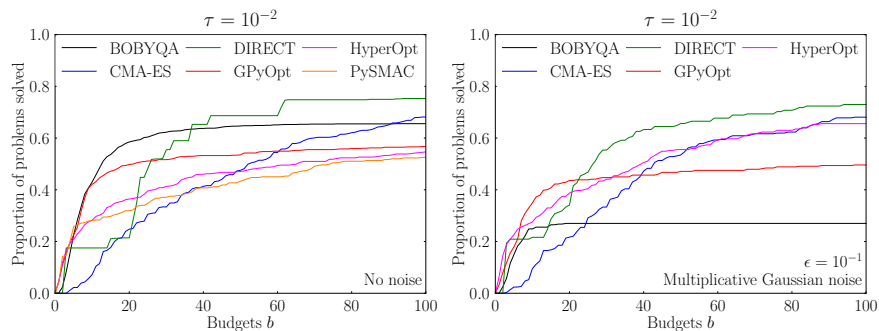


Figure 3: Data profiles. (Left) No noise. (Right) Multiplicative Gaussian noise.

We introduce additive noise and show the results in Figure 4. We can see from this that there is a much reduced performance across all of the solvers and that, for larger noise levels, DIRECT has the greatest reduction. CMA-ES is still the best performing optimiser, while the Bayesian solvers show a reasonable performance for smaller budgets.

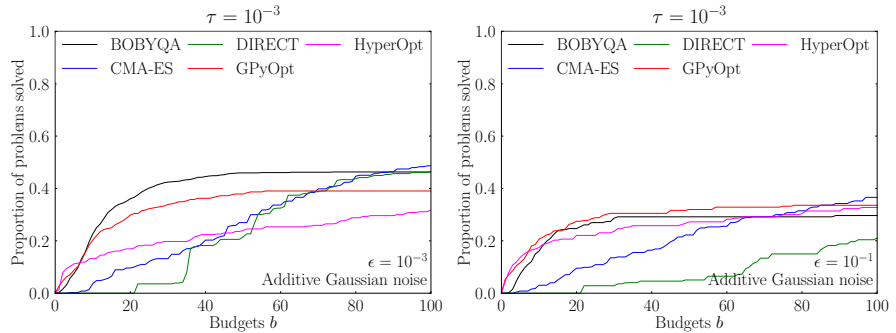


Figure 4: Data profiles using additive noise. (Left) $\epsilon = 10^{-3}$. (Right) $\epsilon = 10^{-1}$.

Application to a real-world example

The results presented so far were for synthetic problems. As a real-world example, we considered the MNIST dataset of 60,000 images of hand written digits. We used a classification scheme, which had three parameters which we could tune. The performance for each choice of parameters was slow to evaluate, noisy, and there was no prior idea of a ‘good’ choice of parameters, and so the problem was a good example of a genuine black-box hyper parameter optimisation problem.

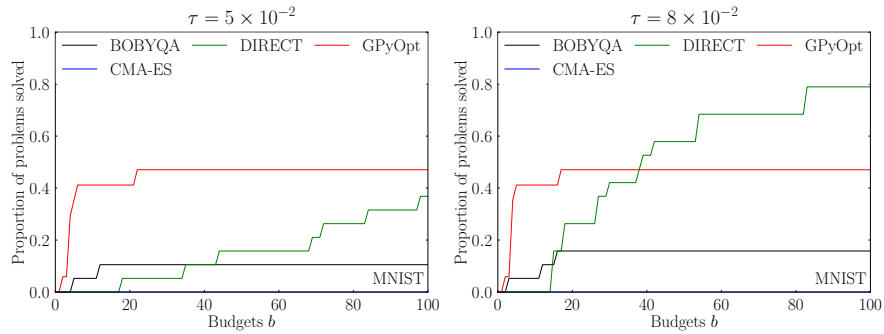


Figure 5: Data profiles for a classification scheme on the MNIST dataset. (Left) $\tau = 5 \times 10^{-2}$. (Right) $\tau = 8 \times 10^{-2}$.


Using MNIST, DIRECT achieved the best classification error.

DIRECT and CMA-ES were usually the top performing solvers.

The results from the MNIST example are shown in Figure 5, and show some surprising features. The most surprising is the non-existent performance of CMA-ES, and the very poor performance of BOBYQA. However, GPyOpt showed a rather unchanging performance across the different tolerances, whereas the performance of DIRECT was very dependent on the budget allowance and tolerance level. Overall it was DIRECT which achieved the best result with a classification error of 8.0%.

5. Discussion, Conclusions & Recommendations

We have surveyed the different solvers and have seen the difficulty in producing any blanket conclusions about “which of the solvers is best”. Generally we found noise levels $\epsilon \leq 10^{-2}$ and tolerances τ between 10^{-4} and 10^{-2} as suitable regimes. These avoided high levels of false positive solutions and that the solutions were assessed at an appropriate level of accuracy. For noiseless functions, we found DIRECT and CMA-ES solved the most problems within our computational allowance, and BOBYQA gave good performance for lower computational budgets. The Bayesian algorithms gave moderate performances overall, but were never shown to be the best scaling nor the best achieving. When we used noisy test problems, BOBYQA’s performance reduced considerably, DIRECT and CMA-ES remained the top competitors, and the Bayesian algorithms remained largely unchanged in their ranking. Bayesian methods did not appear to



demonstrate a significantly better resilience to noise above the other solvers considered. An example using MNIST showed similar results.

Recommendations for further work include adding more solvers and problems to those considered. The effect of increasing the number of the input variables would be a useful feature to differentiate performance. Lastly, all the solvers were run “out of the box”, and no attempt was made to “optimise the optimisers”. Fine-tuning the optimisation algorithms may have the potential to give new results, and would be worth exploring.

6. Potential Impact

The results presented here will hopefully act to advise NAG whether they should consider developing Bayesian solvers for their library. Our findings suggest that while Bayesian algorithms achieve good performance, they appear not to show a competitive edge above alternative methods. This suggests developing a Bayesian algorithm into a commercial product would need careful consideration and further investigation against other global optimisation methods.

Jan Fiala, a numerical software developer at NAG, said: *“We are always looking for new ways to improve our offering within the NAG Library and we know that a good quality global optimization solver would be very valuable to many of our users. The aim of this project was to find out if Bayesian optimization, a highly popular optimization technique within Machine Learning, would be applicable as a generic global optimization solver. Thanks to Oliver we have now gained insight to Bayesian optimization through his series of experiments and we can make a better decision.”*