Optimising High-Dimensional Black-Box Functions with Gaussian Processes

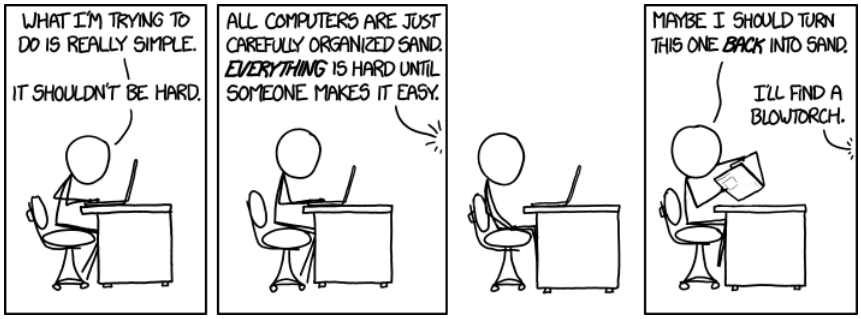
David A Roberts #42008921
Supervisor: Associate Professor Marcus Gallagher
School of ITEE, University of Queensland

22nd June 2014

Abstract

This report describes a simple Gaussian process-based optimisation model, with a particular emphasis on discussing its strengths and weaknesses. We begin by formally defining the model and outlining the necessary implementation details that need to be overcome. In addition to discussing the assumptions and approximation techniques utilised during this project, a number of potential variations on these are also suggested. Following this, experimental results are presented, including a brief and informal analysis of the behaviour of the model, as well as a slightly more rigorous evaluation on the black-box optimisation benchmark (BBOB). Although the results are promising, there are a number of practical deficiencies that prevent the model from being a competitive global optimiser at the current time. The report concludes with a number of possible directions for future research, in the hopes that Bayesian optimisation models will become a more competitive and practical option.

Disclaimer: The potential research directions suggested in this report should be considered entirely speculative, and although the author has attempted to review the relevant literature, time constraints meant that this was anything but exhaustive, and therefore some (or all) may have already been researched in the past. Any lack of citations, beyond what is considered to be common knowledge, does not necessarily indicate original ideas by the author, merely that he is as of yet unaware of any references pertaining to these issues.



http://xkcd.com/1349/

Contents

Chapter 1

Introduction

Global optimisation, and in particular black-box optimisation, can be generally described as the problem of maximising (or minimising) an unknown function, known as the *objective function*, by sampling its value at a number of points. Function evaluations are useless on their own though, so one also has to make certain assumptions about the function. At the very least, this includes the assumption that the function is "smooth" in some sense, but also that the overall structure of the function is relatively simple. This is not to say that the function itself may not be complex, but all information about this complexity should be derived from the observations rather than prior assumptions.[1] These prior assumptions may also be educated by what we have observed about this or similar functions in the past. In addition to this, we can assign problem-dependent preferences to different potential outcomes — for example the quality of the maximum may be much more important than the number of function evaluations required, or *vice versa*.

Bayesian methods provide an elegant way of encoding these prior assumptions and preferences. Information about the structure of a function can be encoded by a *prior distribution*, which is a probability distribution over the space of possible objective functions, also known as a *stochastic process*. A particularly convenient class of priors is that of the *Gaussian processes* (GPs) [1]. Preferences between different goals can be encoded by a *utility function*, which assigns a numerical value to each potential outcome. Decisions can then be made in an arguably optimal manner by maximising the expectation of this utility with respect to the *posterior distribution* induced by the prior and any observed data.

The purpose of this project was to determine whether GP-based optimisation (GPO) can be successfully applied to high-dimensional[2] objective functions. In principle, GPs should be a perfect fit to this problem, as they do not care about the dimensionality of the space *per se*, but only the correlation structure of the datapoints. Nonetheless, some perceive GPO to be either inherently intractable or simply incapable of obtaining decent performance in high dimensions. A number of difficulties in implementing GPO were encountered during this project, which could be influencing this perception. These are outlined later in the report, along with potential directions for future research that could reduce their impact.

1.1 The Model

This project mainly focused on the model described by the probabilistic program shown in Figure 1.1. This is quite a simple model — much more sophisticated variations are entirely possible, and are likely to offer significant performance gains. However, this model was chosen as it embodied enough of the features commonly seen in GPO models [2, 3, 4], whilst being simple enough to implement with minimal difficulty within the time constraints of the project. The individual components of the model, as well as justifications for why they where chosen over other possible variations, will be discussed in more detail in the following chapter.

[1] This is the same reason why one would have reservations about using an optimiser that contained huge hard-coded lookup tables, for example. One might also draw an analogy to the laws of physics, which are quite simple relative to the emergent complexity of the universe.

[2] Where we roughly define "high-dimensional" to be at least beyond single-digits.

maximise	$\mathbb{E}\,U$	▷ expected utility
subject to	$f \sim \mathcal{N}(\mu, \Sigma)$	▷ objective function with GP prior
	$\mu(\cdot) = 0$	▷ zero mean
	$\Sigma(\cdot, \cdot) = K_{\mathrm{SE}}(\cdot, \cdot; \boldsymbol{\theta})$	▷ squared exponential kernel
	$\boldsymbol{\theta} \sim \mathcal{U}(0, +\infty)^D$	▷ hyper-params with flat prior
	$U = f(x^*)$	▷ utility function
	$y_t = f(x_t), \quad t \leq N$	▷ observation model
with actions	$X = \{x_t \in \mathbb{R}^D : t \leq N\}$	▷ sampled points
	$x^* \in X$	▷ reported maximiser
and observations	$Y = \{y_t \in \mathbb{R} : t \leq N\}$	▷ observed function values
	$D \in \{1, 2, 3, 5, 10, 20\}$	▷ dimensionality
	$N \propto D$	▷ function evaluation budget

Figure 1.1: The GPO model considered for this project

It should be noted that the utility function defined in Figure 1.1 is in fact incomplete, as it does not make any reference to computational resource usage. As written, the model specifies that we are willing to spend arbitrary amounts of computational resources in order to make exact inferences and decisions. Clearly this is not the case in practice, as fast approximations are preferred even if they sacrifice a small amount of accuracy or certainty. Our *actual* utility function — the one residing within our skull — penalises resources consumed by executing the program, hence more accurately reflecting the relative importance we place on correctness and computational efficiency. Updating the GPO model to better imitate this would allow these two factors to be optimally balanced in a principled manner, by selecting the appropriate approximations to combine in the implementation in order to maximise the expected utility of executing the generated program. However, this was beyond the scope of this project, so a number of common approximations were instead selected on an *ad hoc* basis.

A number of implementation details (including selecting appropriate approximations) need to solved, in order to translate this abstract probabilistic program into a concrete program that can be (tractably) run on a computer:

- fast techniques and approximations for computing Gaussian conditional distributions, in particular for (pseudo-)inverting large covariance matrices (§2.2)
- a sampler or optimiser for performing inference of any unknown quantities, particularly GP hyper-parameters (§2.3)
- a decomposition of the full maximum expected utility problem into an approximate decision policy, which may itself require an optimiser for computing the preferred actions (§2.4)
- any algebraic manipulations necessary to be able to compute required quantities efficiently[3]
- runtime visualisations and diagnostics in order to identify any high-level performance problems that need to be addressed, and to facilitate efficient human-computer interaction and communication

[3] This is not usually regarded as an implementation detail as such, but it can be performed semi-automatically during the implementation process with the aid of a computer algebra system. For example Theano [5] employs automatic symbolic differentiation to compute gradients, Jacobians and Hessians directly without any manual effort required.

In principle, it should be feasible to perform many of these tasks semi-automatically, with the aid of a library of automated implementation and approximation strategies common to a wide variety of inference and decision problems.[4] However, in the absence of such tools,[5] the translation was performed manually for this project, targeting the Python programming language using the SciPy library [6].

These implementation details are non-trivial for GPO in comparison to some competing optimisation methods. This is due to the fact that Bayesian models generally separate the model and implementation specifications.[6] Although this does cause some difficulties in tractably implementing models, it is in fact a benefit of the Bayesian formalism. By removing all but the most essential details from the model specification, it is much simpler to invent new models and adapt existing ones. Likewise, by separating out the implementation details, these can be studied separately and generalised across a wide variety of different models. Therefore, although Bayesian models require much more groundwork to be performed in advance, in principle it is possible to institute a much more efficient division of labour, especially in comparison to models that shift some of the implementation complexity into the definition of the abstract model itself.

1.2 Optimisation

As discussed earlier, any black-box optimiser needs to make assumptions about the structure of the objective function, as well as the performance criteria that it needs to fulfil. Usually it is assumed that the function exhibits some degree of "smoothness," but the precise details of these implicit assumptions are not always obvious in certain optimisers. It is clear that different optimisers do make different assumptions, as they all have an *inductive bias*, performing better on some functions than others.

In principle, it is possible to elucidate these assumptions empirically. Suppose we have a variety of possible assumptions in the form of prior distributions over objective functions, and a range of different utility functions. The expected utility of any particular optimiser under these assumptions can be empirically estimated by a Monte Carlo integral approximation. More specifically, we could sample a number of objective functions from the prior, execute the optimiser on each, and average the utilities corresponding to each run. The better optimiser performs, the better it must match those assumptions.

For example, in order to evaluate a GPO implementation, we might sample objective functions from our GP prior, and compute the average utility across a number of runs (see Figure 3.1). How well the implementation performs then indicates how well it approximates the abstract model. This also provides a formal way of comparing a number of different implementations of the same abstract model, which could be used to automate the implementation process, at least in principle.

Roughly speaking, we can separate black-box optimisation into two main sub-problems. The first is the amount of effort, in terms of the number of function evaluations (FEs), required to extract all of the necessary information from the objective function. For example, if the location of the maximum strongly depends upon a large source of random bits in a non-trivial manner, then this will set a lower bound on the required number of FEs, with the total number needed in practice depending on the accuracy of the prior assumptions.

The second sub-problem, assuming we have obtained enough information to build an accurate model of the function, is the computational difficulty of finding the maximum of this model. This may depend on details such as whether the model provides a convex optimisation problem or not (for example), as well whatever (approximate) methods are being used to optimise it. The degree to which either of these sub-problems matters depends on the relative cost of FEs and computation, as expressed by our (actual) utility function.

[4] In addition to significantly reducing the amount of time and effort the human user needs to spend on comparatively low-level details, allowing much more expressive models to be used without requiring impractical amounts of human effort, this would also provide the opportunity for different approximations to be selected in response to evidence gathered at runtime, analogously to the way in which just-in-time (JIT) compilers perform different optimisations in response to observed runtime behaviour.

[5] But see `http://probabilistic-programming.org/` for examples of recent work in this direction.

[6] However, one should not fall into the trap of thinking that the model and implementation are completely unrelated, as the two are in fact intimately linked — at the very least, decisions need to be made regarding trade-offs between accuracy and efficiency, which will depend on the model and approximations combined.

Chapter 2

Assumptions, Inference, and Decisions

In this chapter, we briefly discuss the specific assumptions made in the model described in Figure 1.1, as well as possible alternatives to these assumptions. In addition, a number of techniques and approximations for inference and decision-making are outlined, which allow us to trade accuracy for computational efficiency. It is important that each component of the model can be implemented as efficiently as possible (without sacrificing too much accuracy) as any computational savings in one area can be utilised to improve accuracy in another. For example, if we are able to reduce the time required to invert large matrices, we can spend more time on inferring hyper-parameters with greater accuracy.

2.1 Objective Function Prior

We model the objective function by a GP prior, and make the convenient and conventional assumptions that it has a constant zero *mean function* $\mu : \mathbb{R}^D \to \mathbb{R}$ and that the *covariance function* $\Sigma : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$ is specified by the squared exponential kernel

$$K_{\text{SE}}(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}) = \exp\left(-\frac{1}{2}\sum_{i=1}^{D}\left|\frac{x_i - y_i}{\theta_i}\right|^2\right)$$

where the $\boldsymbol{\theta}$ hyper-parameter is a vector of *characteristic length-scales* for each dimension [1, §5.1]. If all of the length-scales are constrained to be equal, then the kernel is known as *isotropic*. It is also common to assume that the kernel is multiplied by a *signal variance* coefficient θ_0, which represents the output scale of the function. However, some brief experiments during this project showed that it is quite easy to overfit this parameter with maximum-likelihood, particularly in the initial phases of the optimisation process when only a few datapoints have been observed, which results in poor overall performance of the optimiser. Therefore, $\theta_0 = 1$ was assumed — more accurate hyper-parameter inference methods (§2.3) would be required in order to be able to relax this assumption.

More sophisticated assumptions than this are of course possible. For example, instead of fixing the mean, it could be given a vague prior and inferred from the data [7]. It is also possible to consider a variety of different kernels. For example, [8] claim that the Matérn kernel performs better on common black-box function benchmark problems, although it was not found to make a significant difference in this project (§3.2.2).

One can also superimpose different kernels, for example two kernels of differing length-scales [9], in order to model a combination of low- and high-frequency components in the objective function. It is also possible to use an *additive kernel* [10] in order to model the objective function as a sum of low-dimensional components. Furthermore, [11] describe a framework for combining an arbitrary number of base kernels to model richly structured functions, and present an approximate greedy algorithm for searching over the space of composite kernels. It should be possible to utilise this framework within a fully Bayesian (hierarchical) model, which could allow more accurate approximate inference procedures.[1]

[1] The main practical issue that needs to be solved for such a model is in determing the promising regions of the composite kernel space that are deserving of the computational resources required to investigate them.

2.1.1 Hyper-parameters

For simplicity, the *hyper-prior* on $\boldsymbol{\theta}$ was chosen to be the improper uniform distribution over the region $\boldsymbol{\theta} \in (0, +\infty)^D$. In some sense, this is the least informative prior possible, and means that we assume all of the information about the hyper-parameters will be provided by the likelihood of the data.[2] When a large amount of data is available, it is often reasonable to assume that the likelihood will be strongly peaked around a single value, and hence the specific choice of any (sufficiently vague) prior will have comparatively little impact on the hyper-parameter posterior distribution. Another consequence of this assumption is that we can estimate a single likely value of $\boldsymbol{\theta}$ without losing too much information. This can be done by either selecting the maximiser of the likelihood function (maximum likelihood inference) or of the posterior (so-called maximum *a posteriori* inference).

However, when only a small amount of data has been observed — in the case of GPO, when only a small number of function evaluations have been performed so far — this assumption is likely to be invalid. Therefore, a somewhat informative prior is desirable, such as the log-normal prior used by [4]. In addition, estimating $\boldsymbol{\theta}$ by a single value becomes a poor approximation, so a more accurate marginalisation of the hyper-parameters, such as a Monte Carlo integral approximation, should be used. This issue is discussed further in §2.3.

2.1.2 Other Alternatives

The computational cost of dealing with high dimensions can be ameliorated slightly by imposing additional constraints on the assumed structure of the objective function. For example, if one assumes that not all of the dimensions are equally informative, it would not be unreasonable to transform the function into a lower-dimensional space prior to performing the necessary inference and optimisation tasks [12, 13, 14]. However, although this may work for some applications, it is quite a strong assumption for general black-box functions, and so was not considered for this project. It is also not entirely clear how one should select the dimensionality of the lower-dimensional space without specific prior knowledge, although in principle a hierarchical model should be able to infer this, perhaps in a similar manner to how the number of components in a clustering problem can be automatically determined with the aid of a Dirichlet process model.

Another possibility is to allow the objective function to have a more heterogeneous structure than allowed by individual GPs with stationary kernels. For example, we may assume that the function behaves differently in different regions of the space, and allow the model enough flexibility to infer these regions from the data. Each region can then be modelled by a GP with a different kernel and/or hyper-parameters. In the one-dimensional case, this is known as *changepoint detection* [15, 16]. Although these techniques are limited to a single dimension, in principle it should be possible to generalise them to higher dimensions, by considering hyperplanes separating the regions rather than single points.

And finally, for some applications a GP may simply be incapable of accurately modelling the structure of the objective function. In such cases, an entirely different prior distribution over the space of functions may be required [12], but this is beyond the scope of this report.

2.2 Gaussian Conditionals

By definition, any finite collection of points from a Gaussian process follows a (high-dimensional) multivariate Gaussian distribution,[3] and so inference techniques for the latter apply to the former also. In particular, the predictive distribution of a GP given a set of observations is simply a conditional Gaussian distribution. Suppose we have sampled at points X with corresponding observations $Y_X = f(X)$, and wish to compute the *predictive distribution* of $Y_T = f(T)$ at a set of test points T. Clearly the joint distribution

$$\begin{bmatrix} Y_X \\ Y_T \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_X \\ \mu_T \end{bmatrix}, \begin{bmatrix} \Sigma_{XX} & \Sigma_{XT} \\ \Sigma_{TX} & \Sigma_{TT} \end{bmatrix} \right)$$

where $\mu_X = \mu(X)$ gives the (column) vector of means of Y_X for each point in X, $\Sigma_{XT} = \Sigma(X, T)$ gives the matrix of covariances with rows corresponding to X and columns to T, and likewise for the others.

[2] Although, since this is a scale parameter, a Jeffreys prior or similar may have been more appropriate.

[3] Conversely, one could consider a multivariate Gaussian to simply be a GP defined on a finite set of points.

We can then apply the appropriate identity [17, §8.1.3] to obtain the conditional (predictive) distribution

$$Y_T|Y_X, T, X \sim \mathcal{N}\Big(\quad \mu_T + \Sigma_{TX}\Sigma_X^{-1}(Y_X - \mu_X), \\ \Sigma_{TT} - \Sigma_{TX}\Sigma_X^{-1}\Sigma_{XT} \quad \Big) \tag{2.1}$$

where $\Sigma_X^{-1} = (\Sigma_{XX})^{-1}$ is the inverse covariance, or *precision*, matrix of Y_X. The precision matrix is closely related to the partial correlation matrix, which can be obtained by a suitable rescaling of the entries. Roughly speaking, the partial correlation between pairs of variables in Y_X indicates how strongly they are correlated when conditioned on the remaining variables in Y_X, and reduces to zero when the pair is conditionally independent given the others. To put it another way, if we assume we have already observed all but two points x and x', the corresponding entry in Σ_X^{-1} tells us how much information observing $f(x)$ will give us about $f(x')$.

If we assume that the observations of function outputs are corrupted by independent Gaussian noise (§2.5) with standard deviation σ_{noise}, very little change is required. We simply need to increment each entry on the diagonal of Σ_{XX} by σ^2_{noise}, prior to inverting it.

2.2.1 Computation

We now turn to the problem of actually computing the inverses and matrix products in (2.1). Standard computational techniques will not be discussed, as these are usually built into most general-purpose linear algebra libraries. However, the covariance matrices we are utilising have greater structure than arbitrary matrices, and we can exploit this for computational gain.

This is quite an important implementation detail, as the fact that these matrix operations need to be performed after each observation means that computational efficiency can degrade quite quickly as the number of function evaluations increases. A naive implementation of GPO can easily have $\mathcal{O}(N^4)$ time complexity, making it intractable to be able to handle more than a few hundred or thousand function evaluations, depending on available resources. This causes difficulties for GPO to be able to efficiently scale to high dimensions, as higher-dimensional problems tend to require a greater number of function evaluations to be able to achieve similar levels of accuracy.

Unfortunately, most of the following computational techniques were not implemented for this project due to time constraints, but it is vital that such approximations be considered in the future if GPO is to be implemented in a scalable way. To put the problem in context, we have available to us multiple implementations of the abstract *matrix (pseudo-)inverse* function, each with different performance characteristics. We need to select the implementation most appropriate for this application which, as discussed earlier, could be formalised by introducing appropriate penalties into the utility function.

Intuitively, the factor $\Sigma_{TX}\Sigma_X^{-1}$ in (2.1) tells us that in order to determine the predictive distribution at some point t, we need to consider all neighbouring points x in the dataset with non-negligible covariance $\Sigma(t, x)$, and all points z conditionally correlated with x such that $\Sigma_X^{-1}(x, z)$ is also non-negligible. It is possible to exploit this fact, as we would rather spend most of our computational resources on highly informative pairs $\{x, z\} \subset X$, as they will have the most influence on the predictive distribution.

A simple way of doing this is to simply zero out all matrix entries smaller (in magnitude) than some threshold, so that we are only required to multiply sparse matrices.[4] This can be accomplished by explicitly truncating the kernel function, and enforcing sparsity constraints on the precision matrix [18]. Assuming the sparse matrix has many fewer non-zero entries than the original, this can result in significant improvements in computational efficiency. It is also possible to re-interpret these sparse matrices as the adjacency matrix of a weighted (undirected) graph over the dataset, producing the network generated by pairs of highly correlated points. From the geostatistics literature, [19] utilise this idea to translate GP models into *Gaussian Markov random fields* (GMRFs),[5] which allow greater computational performance due to increased localisation. [20] discusses some further applications in the context of machine learning.

[4] This approximation is not significantly different to the numerical rounding errors encountered when using floating-point arithmetic, except for in magnitude. The error threshold could either be fixed in advance, or automatically adapted somehow in order to optimally balance accuracy against efficiency.

[5] Thanks to Thomas Taimre for directing me to this paper.

In some situations, simply ignoring small correlations may be too crude an approximation. However, as before, we would still like to spend computational effort proportionally to the amount of information we gain by doing so. [21] draw an analogy to the n-body problem, in which one is required to simulate the gravitational interaction between a number of physical bodies (or *particles*). The similarity is that, for the kernels being considered here, covariance decays with increased distance in a similar manner to gravitational attraction. A common approximation in this problem is to consolidate groups of distant particles into a single point, with the aid of a recursive partitioning of the space. [21] suggest a similar but slightly different approach, in which blocks of the covariance matrix corresponding to groups of points with low mutual correlations are approximated as low-rank matrices.[6] Once this decomposition has been computed, the inverse and determinant of the matrix can be obtained with comparatively little effort. However, the authors note that this method suffers a constant-factor increase in runtime as dimensionality increases.[7]

Another potential source of efficiency gains occurs when an observation $f(x)$ is highly predictable given the other data. In this case, the presence (or absence) of this datapoint has little impact on the posterior distribution. For example, if we have a cluster of points along a small interval with similar function values, throwing away a subset of these points is unlikely to make a substantial impact on the quality of our inferences. [1, §8] extend this idea by removing all but a subset of points, known as the *active set*, from the inference process. They also describe a number of approximate techniques for doing so, with the intent of retaining only the most informative observations. However, computing the active set can be an expensive process in itself, so this method is only applicable if the dataset is large enough to offset this cost by the subsequent performance gains. It also has the issue of completely ignoring potentially useful information, which may be undesirable for some problems.

It is also possible to exploit the fact that the size of the dataset gradually increases with each iteration. For example the *matrix inversion lemma* can be used to incrementally update the inverse as the covariance matrix is modified [1, §A.3]. Similarly, methods for incrementally updating singular value decompositions are also available [22], which can then be used to compute the matrix pseudo-inverses used in the following section.

2.2.2 Numerical Stability

In addition to concerns of speed, we need to be mindful of the accuracy (or *stability*) of our numerical approximations. In particular, problems can arise when we are dealing with clusters of nearby points, as the corresponding rows/columns in the covariance matrix will be almost identical [23, §6.4], making inversion difficult due to ill-conditioning. According to [24, p22]:

> The dot product [of Σ_{TX} and $\Sigma_X^{-1}Y_X$] may turn out to be the sum of very large positive and negative values although its magnitude may be small. This may lead to inaccuracies in the evaluation of the [predictive distribution] when the model is implemented on a computer. The problem is caused by an ill conditioned covariance matrix, for example, when the model assumes a small noise level. To reduce such numerical errors, we can use the LU decomposition of [the covariance matrix] or eigenvector/eigenvalue decompositions. Another way to deal with ill-conditioning is to split [the covariance matrix] into several pieces and make use of the [matrix inversion lemma].

Figure 2.1 illustrates the problems this ill-conditioning can cause (the format is the same as the plots in Appendix A). On the left, a number of samples from the objective function have been taken, including the point $f(2.1) \approx .6$. The plot on the right shows the extreme change to the predictive distribution after observing $f(1.9) \approx .635$. Note that the y-axis has increased by almost an order of magnitude. Clearly this observation should not have caused such a drastic change to the posterior distribution.

The problem is that, although values of the objective function are observed without any noise explicitly added (§2.5), there is still a small amount of *numerical noise* being added by the computational approximations and rounding errors. This must be taken into account. Otherwise, in regions near previous samples the predictive variance will be very low, and so even a small numerical error in the observation will push it outside of the high-density region of the predictive distribution, causing a large

[6] Thanks to Marcus Frean for directing me to this paper.

[7] It is possible that this is due to the fact that is more difficult to order high-dimensional points in order to concentrate covariances along the diagonal of the matrix, as required by the decomposition method.

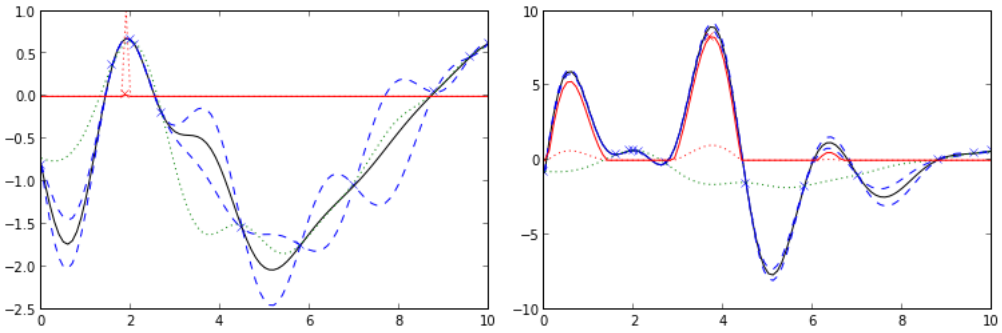


Figure 2.1: Sudden change to posterior caused by ill-conditioning

change to the posterior. This is a case of unpredictable observations shifting beliefs much more than predictable ones.

A particularly simple solution to this problem is to allow the model to assume a small amount of observation noise. Independent Gaussian noise with standard deviation $\sigma_{\text{noise}} = 10^{-3}$ was found to work quite well in this project, although ideally the magnitude of noise should be set taking into account problem-specific factors as well as other numerical approximations being used in the implementation.[8] This allows the model to cope with small numerical errors, even if the observations would ideally have zero noise in an exact implementation. It also ensures that no two rows or columns in the covariance matrix are almost-identical, due to the additional noise term added to the diagonal (§2.2).[9] One might also draw parallels between this and regularisation techniques like ridge regression.

Another way of preventing numerical instability is to avoid explicitly computing the matrix inverse in favour of other techniques. One way of doing this is noting that we can replace the matrix inverses in (2.1) with (Moore-Penrose) *pseudo-inverses*, which can be computed by many linear algebra libraries.[10] Unfortunately, although the asymptotic computational complexity of doing so is no greater than computing the actual inverse, in practice there is a constant-factor increase in computational expense. This factor depends on the method being used to calculate the pseudo-inverse, of which some libraries provide several alternatives. In particular, when we are dealing with square matrices, methods based on the *singular value decomposition* (SVD) are known to be faster, and possibly more numerically stable, than those taking a least-squares approach.[11]

An interesting side-effect of computing the pseudo-inverse is that one can determine the *effective rank* of the covariance matrix, simply by counting the number of non-negligible singular values.[12] Unfortunately this does not directly lead to any significant computational gains, due to the expense of actually computing the SVD in the first place, but it does give an indication of how "difficult" the inversion problem is. That is, it should be possible to invert matrices with low effective rank more easily than those with high rank without a significant reduction in accuracy, due to the fact that such matrices can be effectively "compressed" to low-rank matrices [25].[13]

In the case of a GP with a squared exponential kernel, informal experimentation suggests that the effective rank of the covariance matrix is the same as the "effective" size of the dataset, by which we mean the size after all points closer than a length-scale apart have been removed. This may provide some

[8] In particular, this noise should not be too small as doing so has little difference to assuming zero noise, nor should it be significantly larger than the amount of numerical error actually present, as this will cause other problems with accuracy due to the model underfitting the data.

[9] [23] suggests instead taking finite differences between nearby observations and using them to approximate derivative observations. However, this seems like a much more complicated solution without any clear benefit.

[10] As the matrices were are dealing with are technically invertible, the pseudo-inverse is still equal to the actual inverse in theory, but tends to be more robust to numerical errors.

[11] For example, a simple experiment showed that the time required to pseudo-invert a random 1000×1000 matrix can differ by an order of magnitude between the two methods. Using the SciPy library, one should therefore use the SVD-based `pinv2` function for square matrices rather than the standard `pinv` function. When the matrix is symmetric, the `pinvh` function can be used to halve the computational expense, requiring just over four times as long as computing the matrix inverse. Further details at `http://vene.ro/blog/inverses-pseudoinverses-numerical-issues-speed-symmetry.html`

[12] Such as those with magnitude of at least 1% that of the largest singular value, for example.

[13] One might draw an analogy to other SVD-based dimensionality reduction techniques, such as *principal components analysis* (PCA) and *latent semantic analysis* (LSA).

guidance for selecting the size of the "active set" in the approximations outlined earlier, since they can be interpreted as a case of reduced-rank approximation [1, §8.1]. However, as this low-rank compression [25] does not simply remove rows/columns from the matrix, but rather creates linear combinations over a small set of basis vectors, it may be possible to achieve better performance by *combining* datapoints in a similar manner, rather than completely ignoring some of them.

2.3 Hyper-parameter Inference

In order to infer the infer the kernel hyper-parameters $\boldsymbol{\theta}$, we need to compute their posterior distribution conditioned on the observations, which is the product of the hyper-prior and the likelihood of the observed data. A convenient transformation of the likelihood is the negative logarithm, and is the same as that of a multivariate Gaussian

$$-\mathcal{L}(\boldsymbol{\theta}; X, Y) = -\log p(Y|X, \boldsymbol{\theta}) \propto Y^\top \Sigma_X^{-1} Y + \log|\Sigma_X| + \text{const.} \tag{2.2}$$

As discussed earlier, if we assume that this function has a single strong peak, and the hyper-prior is quite flat and uninformative, then it is reasonable to approximate the relevant marginalisations with a single point estimate $\hat{\boldsymbol{\theta}}$ using *maximum likelihood* (ML), by minimising (2.2). Alternatively, when we have a suitably informative prior, it is better to maximise the posterior instead — so-called *maximum a posteriori* (MAP) inference — as it may differ significantly from the likelihood.

However, as the results presented in the following chapter show (§3.2.3), the single-strong-peak assumption required for ML or MAP inference to be accurate is not necessarily justified in the case of GPO. These approximate inference methods can also cause problems by overfitting the hyper-parameters [8, 12] — particularly in the initial stages of the optimisation when little data has been observed — as well as issues with hyper-parameter estimates jumping suddenly between iterations [9].

In order to avoid these problems, a more accurate approximation to marginalising out the hyper-parameters is necessary. For example, we can utilise Monte Carlo samples from the hyper-posterior in order to approximate integrals by sample averages. [3] discuss how this can be performed in order to marginalise expected improvement (§2.4), and demonstrate that this achieves better performance in practice than taking single point estimates with a method such as maximum-likelihood.

However, these improvements in accuracy do come at the expense of added implementation complexity. Therefore, for this project, $\boldsymbol{\theta}$ was simply estimated using maximum-likelihood, minimising (2.2) by conjugate gradient, randomly restarting twenty times by sampling from a standard log-normal distribution.[14]

2.4 Utility

The *utility function* $U = f(x^*)$, where x^* is the maximiser reported by GPO, was chosen both for its simplicity, as well as compatibility with the commonly-used *expected improvement* (EI) criteria (2.3). However, it should be noted that this utility function encourages a specific type of behaviour from the optimiser. In particular, since we are considering the *value* of the maximum, the optimiser will prefer situations in which there is a small chance of a very large maximum (and possibly a high chance of a poor one) to those in which there will be a moderate maximum with near certainty, for example. The extent to which this will impact on actual performance depends on the particular application. If such behaviour is undesirable, a different utility function should be considered that more accurately reflects the priorities of the problem.

As discussed earlier, ideally one should also take into account the costs of (computational) resources consumed by the optimiser. For example, [26, 3] explicitly consider the costs of performing function evaluations, so that the optimiser can avoid taking samples that would provide little information relative to the expense of doing so. [2, §2.3.2] suggest discounting EI by an exploration/exploitation parameter ξ, to make the optimiser ignore samples that are likely to produce only small improvements to the maximum, which could be considered an approximation to this idea. However, these concerns were beyond the scope of this project.

[14] This was simply a convenient distribution to use since we are favouring simplicity over accuracy.

2.4.1 Justification

In the preceding discussion we have implicitly assumed that maximising expected utility (MEU) is a sensible thing to do, although this is not necessarily a foregone conclusion. Theoretically, it can be justified somewhat by the von-Neumann-Morgenstern axioms [27], or the Bayesian analogue due to Savage [28],[15] in a similar way to how Bayesian inference can be justified by Cox's axioms [29]. It is also one of the few commonly-used frameworks that allows us to make decisions in a principled manner within a Bayesian context, without having to resort to *ad hoc* decision rules. However, it does have some potential pitfalls, so appropriate care needs to be exercised in order to avoid them.

The classic illustration of some of these issues is the St. Petersburg paradox.[16] Suppose we play a game in which you only win a small amount most of the time, but always have a small chance of winning an arbitrarily large amount, such that the expected payout of any given game is infinite. Blindly maximising expected utility would cause us to bet arbitrarily large amounts of money on this game, as the expected utility of playing would always be greater than that of not playing.

The most obvious problem is that unbounded utilities simply don't make sense in a finite world — any utility function for a practical problem is inherently bounded, and assuming otherwise can lead to undesirable behaviour.[17] The utility U defined earlier for the GPO model is in fact unbounded, but in practice it will be limited by the precision of the machine, so we could easily bound it at a suitably large value (10^{100} say) without causing too much of a difference. Unbounded utilities are simply a mathematical convenience. One might draw an analogy to the use of improper priors which, although being quite convenient, make little sense when taken literally and can cause difficulties in some situations. What we are really doing is taking the limit of an appropriate sequence of proper priors (or bounded utilities), and problems occur precisely when this limit is nonsensical.

However, suppose for a moment that we were to allow unbounded utilities, to force us to look at other potential issues. As the expectation is non-finite, it clearly doesn't make sense to conceptualise the average payout over a large number of trials, as the law of large numbers simply doesn't apply.[18] Therefore, we need to consider how many games we intend to actually play. Figure 2.2 shows a number of simulations of the average payout trajectory.[19] It can be seen that, other than the occasional large win, the "typical" average payout tends to gradually increase with the number of trials. Therefore, we might be inclined to bet a greater amount if we intended to play a larger number of trials, as we'd be more likely to win it back than if we played only a few.

This example illustrates the fact that decisions cannot always be made in isolation, the context in which they are made also need to considered. In the case of GPO, one should ideally consider how it is going to be used. If we are only going to optimise a single important function and then throw away the program, then more conservative behaviour may be desirable, in order to have the best chance of a favourable outcome. On the other hand, if it is to be run on a large number of functions over its lifetime, we may want it to take more risks that may result in worse results in a number of isolated cases but better results overall.

Another piece of context that needs to be taken into account is the "initial capital" of the agent. In the game above, if we bet a significant chunk of our wealth on each trial, then there is a greater chance that we will go bankrupt before being able to win it back (the so-called *gambler's dilemma*). In other words, we need to make sure that long-term gains aren't overshadowed by short-term losses. In the case of our optimiser, suppose we have no idea whether it performs well, and are trying to quickly evaluate it. If it performs poorly throughout our limited testing, we are likely to terminate it and focus on another solution, so the optimiser would be wise to behave more conservatively in order provide a better first impression. On the other hand, if we are willing to give the agent a much longer time in order to prove itself, it would not be unreasonable to risk more losses for greater overall gains.

[15] See `http://lesswrong.com/lw/5te` for a summary of the axioms.

[16] Expected utility was in fact originally proposed as a solution to this paradox, but it is generally possible to modify it to display the same problems for any unbounded utility, simply by redistributing probability mass in order to make the payout have non-finite expectation.

[17] In fact, depending on the axioms one subscribes to, utilities may be bounded by definition.

[18] In the same way that averaging samples from a Cauchy distribution never converges to a single point, no matter how samples are taken. See [30] for a colourful discussion of a similar betting game that utilises the Cauchy distribution.

[19] Also see `http://www.science20.com/print/96549` for an (informal) discussion of this idea.

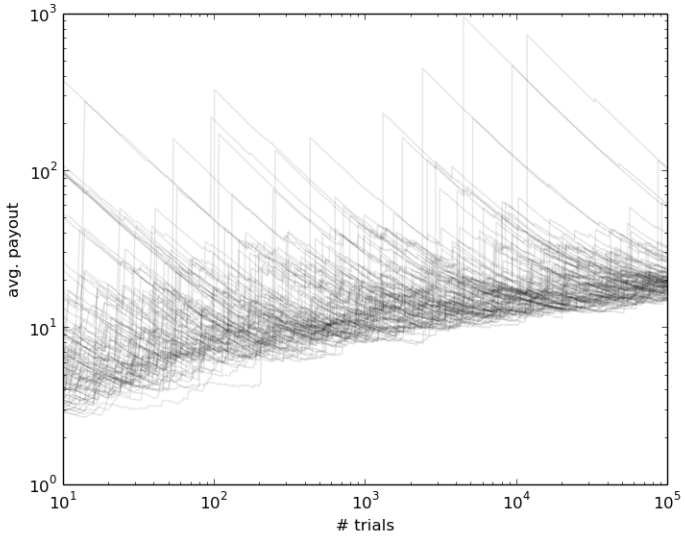


Figure 2.2: 100 independent simulations of 10^5 St. Petersburg games played in sequence

Of course, it is not completely incorrect to treat individual problems independently from their context. Doing so may be a sufficient approximation for the application, in the same way the *naive Bayes* independence assumption is sufficient for a number of problems. However, it may be helpful to reconsider this if the types of issues outlined above are likely to be an issue.

It is also possible to reduce the disparity between inference and decision-making. Suppose we are dealing with bounded utilities (as argued above), and furthermore that they take values in the range $[0, 1]$. This is not difficult to do, since utilities are equivalent under (positive) affine transformations, by the linearity of expectation.[20] It is then possible to re-interpret utilities as (conditional) probabilities. Specifically, the utility $U(\omega)$ of some world-state ω can be identified with the probability of some notion of "ultimate success" conditioned on the state ω [31]. MEU is then equivalent to simply maximising the (marginalised) probability of success. Or, to put it another way, if we put a uniform prior over the set of actions, we're selecting the most likely actions conditioned on success [32].

There are several useful repercussions to this. The first is that common decision-making approximations can be analysed in a familiar probabilistic setting. For example, [31] show how time-discounted rewards can be re-interpreted as a prior over the lifetime of the agent. Not only does this give a clearer insight into the problem, which may be a helpful mental aid for constructing and analysing utility-based models, but it also allows techniques to be shared across the two problems. As an example of this, [31] show that the *expectation-maximisation* (EM) algorithm, commonly used for performing inference by maximum-likelihood, can be re-targeted as a technique for making decisions by maximising expected utility.

This interpretation may also highlight the connection between Monte Carlo inference techniques and stochastic optimisation procedures. For instance, [33] describes the link between state-of-the-art global optimisers such as CMA-ES and *natural gradients*, which can in turn be linked to *variational methods* for Bayesian inference [34]. Although this correspondence is somewhat superficial, it would be interesting to investigate whether the connection runs deeper — for example, whether it is possible to re-interpret optimisers such as CMA-ES as performing approximate (variational) Bayesian inference.

These concerns were beyond the scope of this project, but they may be interesting directions to investigate in the future.

[20] The maximiser of $\mathbb{E}\,U$ is unchanged (assuming $a > 0$): $\max \mathbb{E}(aU + b) = \max(a \cdot \mathbb{E}\,U + b) = a \cdot \max(\mathbb{E}\,U) + b$.

2.4.2 Maximising Expected Utility

In this section we derive the optimal decision policy for selecting the actions x_t ($t \leq N$) and x^* so as to maximise the expected utility $U = f(x^*)$. Unsurprisingly, this optimal solution is generally intractable in practice, so we also discuss several approximations that will allow us to select samples efficiently.

Firstly, we assume that the number N of *function evaluations* (FEs) the optimiser is allowed to make is proportional to the dimensionality D. For this project, ten function evaluations were allowed for each dimension, so that $N = 10 \cdot D$. This was mainly for practical purposes, to allow experiments to run for a practical amount of time. However, it can be partially justified, as [8] claim that GPO's main strength in comparison to other optimisers is precisely this low-budget case. This is not to say that a larger budget is not possible, but the speed of the implementation does limit this somewhat. Fast approximations for increasing this limit were discussed in §2.2.

Now, let $M_k(x_{1\ldots k}, y_{1\ldots k})$ be the expected utility we will obtain if we choose the next sample x_{k+1} optimally — that is, so that it maximises expected utility — given we have already sampled $x_{1\ldots k} = \{x_t : t = 1 \ldots k\}$ and observed $y_{1\ldots k} = \{y_t : t = 1 \ldots k\}$. Note that, although we only consider sequential sampling schemes here, it is also possible to sample batches of points in parallel [3].

M_N is a special case as we have already chosen all of our samples, and so the final $(N+1)$th decision is to report the maximiser $x^* \in X = x_{1\ldots N}$. Therefore we have

$$M_N(x_{1\ldots N}, y_{1\ldots N}) = \max_{x^* \in X} \mathbb{E}[\underbrace{f(x^*)}_{U} | x^*, x_{1\ldots N}, y_{1\ldots N}] = \max_{x^* \in X} f(x^*) = \max_{y^* \in Y} y^*$$

since all potentially reportable maxima $f(x^*) = y^* \in Y$ are known with certainty by this point. Taking one step back to M_{N-1}, we have two decisions to make in sequence, x_N and then x^*. Since we're assuming x_N and x^* will both be chosen optimally, and M_N gives the expected utility when x^* is chosen optimally, we can invoke the tower property of conditional expectation to obtain

$$M_{N-1}(x_{1\ldots N-1}, y_{1\ldots N-1}) = \max_{x_N} \mathbb{E}[M_N(x_{1\ldots N-1} \cup \{x_N\}, y_{1\ldots N-1} \cup \{y_N\}) | x_N, x_{1\ldots N-1}, y_{1\ldots N-1}]$$

Note that all of the terms in the expectation are known except for y_N. We can simplify this expression by substituting definitions

$$\begin{aligned} M_{N-1}(x_{1\ldots N-1}, y_{1\ldots N-1}) &= \max_{x_N} \mathbb{E}\left[\max_{y^* \in Y} y^* \middle| x_{1\ldots N}, y_{1\ldots N-1}\right] \\ &= \max_{x_N} \mathbb{E}[\max\{y_N, \eta\} | x_{1\ldots N}, y_{1\ldots N-1}], \quad \text{where } \eta = \max y_{1\ldots N-1} \\ &= \eta + \max_{x_N} \mathbb{E}[I | x_{1\ldots N}, y_{1\ldots N-1}], \quad \text{where } I = \max\{y_N - \eta, 0\} \end{aligned}$$

where I is the *improvement* of $y_N = f(x_N)$ over the previous maximum η. By induction we can provide a similar definition for M_k for the more general case $k < N$

$$M_k(x_{1\ldots k}, y_{1\ldots k}) = \max_{x_{k+1}} \mathbb{E}[M_{k+1}(x_{1\ldots k} \cup \{x_{k+1}\}, y_{1\ldots k} \cup \{y_{k+1}\}) | x_{k+1}, x_{1\ldots k}, y_{1\ldots k}]$$

with the special cases

$$\begin{aligned} M_1(x_1, y_1) &= \max_{x_2} \mathbb{E}[M_2(\{x_1, x_2\}, \{y_1, y_2\}) | x_2, x_1, y_1] \\ M_0 &= \max_{x_1} \mathbb{E}[M_1(x_1, y_1) | x_1] \end{aligned}$$

It follows that the optimal actions are defined by the following equations

$$\begin{aligned} x_1 &= \arg\max_{x_1} \mathbb{E}[M_1(x_1, y_1) | x_1] \\ x_2 | x_1, y_1 &= \arg\max_{x_2} \mathbb{E}[M_2(\{x_1, x_2\}, \{y_1, y_2\}) | x_2, x_1, y_1] \\ x_k | x_{1\ldots k-1}, y_{1\ldots k-1} &= \arg\max_{x_k} \mathbb{E}[M_k(x_{1\ldots k-1} \cup \{x_k\}, y_{1\ldots k-1} \cup \{y_k\}) | x_k, x_{1\ldots k-1}, y_{1\ldots k-1}] \\ x_N | x_{1\ldots N-1}, y_{1\ldots N-1} &= \arg\max_{x_N} \mathbb{E}[I | x_{1\ldots N}, y_{1\ldots N-1}] \\ x^* | x_{1\ldots N}, y_{1\ldots N} &= \arg\max_{x^* \in X} f(x^*) \end{aligned}$$

2.4.3 Approximations

The recurrences derived above show that our decision problem displays the optimal substructure required for a dynamic programming implementation. However, given the large space of possible action $x_{1\ldots k}$ and observation $y_{1\ldots k}$ trajectories, there is unlikely to be enough overlapping sub-problems for this to be done tractably. Therefore, we need to turn to approximations.

Let's begin by looking more closely at how to select the final sample x_N. Since our objective function is modelled by a GP, the improvement I is simply a truncated Gaussian, with mean $\mathbb{E}(y_N|y_N > \eta) = \mu + \sigma \cdot \varphi(Z)/\Phi(Z)$ where $Z = -(\eta - \mu)/\sigma$, μ and σ are the mean and standard deviation of the predictive distribution of $y_N = f(x_N)$, and Φ and $\varphi = \Phi'$ are the standard Gaussian cdf. and pdf. respectively. Also note that y_N is greater than η with probability

$$\Pr(y_N > \eta) = \Pr\left(\frac{y_N - \mu}{\sigma} > -Z\right) = \Pr\left(\frac{y_N - \mu}{\sigma} < Z\right) = \Phi(Z)$$

Therefore, the *expected improvement* (EI)

$$\begin{aligned}\mathbb{E}[I|x_{1\ldots N}, y_{1\ldots N-1}] &= \mathbb{E}(y_N - \eta|y_N > \eta) \cdot \Pr(y_N > \eta) + \mathbb{E}(0|y_N \leq \eta) \cdot \Pr(y_N \leq \eta) \\ &= [\mathbb{E}(y_N|y_N > \eta) - \eta] \cdot \Phi(Z) + 0 \\ &= (\mu - \eta) \cdot \Phi(Z) + \sigma \cdot \phi(Z) \end{aligned} \tag{2.3}$$

It can be seen that this quantity is quite easy to compute, with x_N as its maximiser.[21] We will return to the question of how to actually find the maximum later. For now it is sufficient to note that finding the optimal value of x_N is much easier than finding that of x_k for $k \ll N$. Therefore, a potential approximation is to treat every sample as if it's the last, completely ignoring the issue of how the current sample will affect any future ones. This type of approximation is known as *greedy*, *myopic*, or *single-step lookahead*. How well the approximation performs in practice will be affected by the actual strength of the dependence between successive samples.

This EI criteria [2, 3, 4] is the approximation that was utilised during this project, but it is still of interest to consider other more accurate approximations. The obvious extension is a two-step lookahead, where we treat every sample as if it were the penultimate one. In particular, we have

$$x_{N-1}|x_{1\ldots N-2}, y_{1\ldots N-2} = \arg\max_{x_{N-1}} \mathbb{E}\left[\max_{x_N} \mathbb{E}[I|x_{1\ldots N}, y_{1\ldots N-1}] \middle| x_{1\ldots N-1}, y_{1\ldots N-2}\right]$$

It is not quite as easy to solve this expectation analytically as it was before, and therefore we must approximate it numerically. This is complicated by the fact that EI needs to be maximised with respect to x_N within the integrand, which is non-trivial in itself. However, as we are using a GP with a squared exponential kernel, the correlation between points further than a few length-scales apart is negligible,[22] so our posterior on the objective function undergoes only localised changes with each new sample. Therefore, any potential increases in EI will only occur in this local region.

[7] exploits this fact by averaging the EI over a sample of nearby points, in order to recognise the regions where a future increase in EI is possible. A more accurate, but potentially more expensive, approximation to $\mathbb{E}U$ at any given x_{N-1} is the following. Note that the only stochastic quantity in the integrand is $y_{N-1} = f(x_{N-1})$, within $\eta = \max\{y_{N-1}, \max y_{1\ldots N-2}\}$. Therefore the above expectation is only a 1-D integral, easily approximated by sampling it at a number of points, either by randomly drawing from the predictive distribution, or deterministically dividing it into a number of quantiles.

For each sample of y_{N-1}, there are two possibilities. The first is that the local changes to EI are not significant enough to cause its overall maximum to increase, in which case the integrand is simply equal to the current maximum EI obtained outside the local region. The second is that the change in the posterior of f is significant enough to cause the EI maximum to move to a new point x' nearby x_{N-1}. In this case the integrand is increased to the new EI at this point x'.

[21] [23, §6.2] provides a similar derivation, but erroneously claims the expression should contain the variance σ^2 rather than the standard deviation σ. He also claims that the expression as stated here is dimensionally invalid but, as far as the present author can determine, it is not — μ, η and σ are all measured in function-output units, and hence it is σ^2 that would be dimensionally invalid.

[22] In certain applications this may in fact be a reason *not* to use these kernels, when long-range dependencies are important.

Determining whether EI has increased significantly in the neighbourhood of x_{N-1} is not an entirely trivial matter. However, it may be sufficient to use local optimisation methods to search the local maxima surrounding x_{N-1}. This could be done reasonably efficiently by exploiting the structure of the EI surface, using gradient information at the very least.

Extending multi-step lookahead further beyond this becomes increasing difficult. [23, §6.3] describes how it may be done in principle, but notes that a naive implementation will be quite computationally demanding in practice. Therefore, it is clear that being able to look more than a few steps ahead will require aggressive approximations. It would be of interest to investigate whether *Monte Carlo tree search* (MCTS) techniques [35] would be applicable to this problem. MCTS has been successfully used in a number of difficult sequential decision problems, such as playing computer Go.[23] It is possible that there may also be other relevant techniques in the fields of (stochastic) control theory, operations research, or automated planning, by re-interpreting GPO as a *partially observable Markov decision process* (POMDP) [37]. However, investigating this any further was beyond the scope of this project, and is left as a potential future research direction.

2.4.4 Making Decisions

We should note that maximising EI (2.3), and multi-step extensions thereof, is not the only option for making decisions within GPO. Two other commonly used alternatives are *probability of improvement* (PI), and the *upper confidence bound* (UCB) criteria from multi-armed bandit theory [2, §2.3]. However, [2, §2.3] shows that PI can perform quite poorly compared to EI, but EI and UCB tend to induce similar behaviour. Additionally, as neither PI nor UCB are based on expected utility theory, they lack many of the benefits of such a coherent approach, such as the conceptual ease with which EI can be extended to multi-step lookahead. Therefore, neither of these alternatives were considered for this project, and they do not in themselves appear to be particularly promising research directions.[24]

Whatever function we choose to maximise in order to select samples, be it EI or otherwise, we require an optimisation algorithm to find the maximum. Clearly this is not a trivial matter — if it were, then there would be little purpose to inventing new global optimisers such as GPO. Selecting good samples is essential as, unsurprisingly, poor samples can have a significant impact on overall performance (§3.2.2). However, this problem does not seem to get anywhere near the attention it deserves, and appears to be something of an afterthought much of the time — with people simply plugging in general purpose global optimisers such as DIRECT [2], conjugate gradient (CG) with random restarts [4], or even state-of-the-art competitors to GPO such as CMA-ES [8]. The latter is particularly unacceptable as not only will it result in GPO being much slower than directly applying CMA-ES to the problem, but it also raises questions as to whether it is GPO or CMA-ES that is doing all of the heavy lifting.

Ideally, we would like to tailor an optimiser to the structure of the EI function, in a similar way to how Markov chain Monte Carlo (MCMC) samplers are tailored to the structure of probabilistic models in order to perform inference tractably.[25] In principle, it should be possible to exploit the structure of

[23] It is interesting to note that MCTS encounters similar problems to GPO in terms of trading exploration for exploitation, and needs to decide which branches of the decision tree would be most helpful to sample, similarly to how GPO needs to decide which points to sample from the objective function. Therefore, it may be possible to decompose an implementation of GPO with MCTS into a hierarchy of (progressively simpler) decision sub-problems. Since each decision problem would have multiple sub-problems, computational expense becomes much more important further down the hierarchy. We can no longer afford to make the assumption that computation is free by leaving it absent from the utility function, as otherwise the recursive partitioning into sub-problems would likely never reach a base level that can be cheaply computed. If these (non-trivial) issues were to be taken care of, it is possible that GPO could be implemented in a fully Bayesian manner, with approximations being applied at each level of the hierarchy to sufficiently accommodate the computational resource usage penalties described by the relevant utility functions. However, this should all be treated as speculation, obviously it was beyond the scope of this project to investigate such options more thoroughly. See [36] for some recent work on this *Bayesian meta-reasoning* problem, with applications to MCTS.

[24] If we wanted the optimiser to report the *most probable* maximum, we could set the utility to be the indicator function $U = [x^* = \arg\max_x f(x)]$ which takes value one (and zero otherwise) if and only if the reported maximiser x^* is the true (but unknown) maximiser of the objective function f. If instead we wished to maximise the probability that the maximum was greater than some threshold $\hat{y}$, then we could set $U = [f(x^*) > \hat{y}]$. In both cases, $\mathbb{E}\,U$ is simply the probability of the corresponding proposition. Finding good approximations to utilities such as these would likely result in better performance than the rather myopic and *ad hoc* PI criteria.

[25] In fact, by unifying decision-making with inference (§2.4.1), it may even be possible to apply inference algorithms like MCMC directly to this problem.

the EI surface[26] in order to locate the maximum with much less effort than a general-purpose optimiser supplied with none of this structural information.

Another problem is the fact that EI has the same input dimensionality as the external objective function, and therefore the scalability of GPO to high dimensions directly depends on that of the EI optimiser. Utilising structural information could help to alleviate this problem also. We again draw an analogy to MCMC inference methods which, by exploiting information about the structure of joint probability distributions, are known to scale quite well to high dimensions [38, §29.1].

Unfortunately, investigating this issue in any great detail was beyond the scope of this project, so EI was simply maximised with CG,[27] restarting twenty times from points sampled uniformly from the input space. [4] provides the necessary analytic expressions for calculating the gradient of EI.

2.5 Observation Noise

For this project, it was assumed that *noiseless observations* $y_t = f(x_t)$ of the objective function are available. It is relatively simple to update the model to handle noisy observations $y_t = f(x_t) + \varepsilon_t$, where $\varepsilon_t \sim \mathcal{N}(0, \sigma^2_{\text{noise}})$ is independent Gaussian noise. However, for significant amounts of noise, the implications on the approximations required to implement the model can be non-trivial.

In particular, EI (2.3) does not handle large amounts of output noise very well, which is not particularly surprising as it was derived from a noise-free assumption. In order to see what would need to change in this case, let's repeat the derivation from the previous section with a noisy observation model. Now

$$M'_N(x_{1\ldots N}, y_{1\ldots N}) = \max_{x^* \in X} \mathbb{E}[\underbrace{f(x^*)}_{U} | x^*, x_{1\ldots N}, y_{1\ldots N}] = \max_{x^* \in X} \mu_N(x^*)$$

where μ_N is the posterior predictive mean of the GP at x^*, given the previous observations $x_{1\ldots N}, y_{1\ldots N}$. Note that, since function values are not known with certainty at the sampled points, we now have to maximise the *predictive mean* rather than maximising the sampled value [2, §2.4]. It follows that

$$M'_{N-1}(x_{1\ldots N-1}, y_{1\ldots N-1}) = \max_{x_N} \mathbb{E}\left[\max_{x^* \in X} \mu_N(x^*) \middle| x_{1\ldots N}, y_{1\ldots N-1}\right]$$

That is, instead of calculating the expected improvement over the previous best sample, we determine that of the predictive mean over the set of sampled points.

Note that the improvement to the predictive mean does not necessarily occur at the location of the new sample x_N, because in some cases the sample at x_N could in fact cause the predictive mean to increase somewhere else [7]. However, if we assume that the latter is not of significant concern, [7] shows that the expected improvement in mean is given by $(\mu - \hat{\mu}) \cdot \Phi(Z) + w \cdot \sigma \cdot \varphi(Z)$ where $Z = (\mu - \hat{\mu})/(w \cdot \sigma)$, $w = 1 - \sigma^2_{\text{noise}}/\sigma^2$, $\hat{\mu}$ is the previous best mean, and μ and σ^2 are the parameters of the distribution of y_N. He also demonstrates that this method performs significantly better than standard EI in some cases, particularly when dealing with (noisy) multi-modal objective functions.

The above derivation assumes that we are using the same utility as before, and obviously would differ for other choices of U. For example, if the utility penalised large deviations between the reported maximum and the true function value at that point, then M'_N would also refer to the predictive uncertainty of the underlying function value $f(x^*)$. This would also allow us to relax the constraint on the maximiser from $x^* \in X$ to $x^* \in \mathbb{R}$, without fear of x^* being selected from a highly uncertain region.[28]

[26] Roughly speaking, EI is small immediately surrounding previous samples and the regions around samples with low function values, moderate in highly uncertain regions with no existing samples, and high in the regions around samples with high function values.

[27] Simulated annealing with standard parameters was also tested, which obtained similar levels of performance, and therefore results of which are omitted for brevity.

[28] [23, §6.5] suggests simply disallowing regions where predictive variance exceeds a set threshold, but this seems rather *ad hoc*, and does not account for the fact that we may be willing to tolerate a greater amount of uncertainty depending on the size of the mean.

Chapter 3

Experimental Results

This chapter presents experimental results demonstrating the behaviour and performance of GPO, firstly with respect to a few informal test functions, followed by the more rigorous Black-Box Optimisation Benchmark (BBOB).

3.1 Informal Behaviour Analysis

Appendix A provides some detailed visualisations of the behaviour of GPO using EI (GPO-EI) on both a simple sinusoidal test function, as well as the case where the objective function is actually sampled from GP, in one and two dimensions. From these plots, it can be seen that GPO-EI tends to alternate between *exploring* uncertain (high-variance) regions and *exploiting* sampled high-mean regions. If it is in the exploration phase and stumbles upon a large function value, it immediately switches to exploitation mode and takes a number of samples in the surrounding area, in an attempt to improve on the local maxima. Once it has taken enough samples to reduce the level of uncertainty — such that it deems that it has located the maximal point in that region — it reverts to exploration mode, sampling points spread across the regions of high uncertainty. This process continues until it has exhausted all promising regions of the space.

It should be noted that none of this behaviour has been explicitly programmed, it simply emerges naturally from the EI criteria. However, due to the myopic nature of EI, this two-mode behaviour is a rather simple strategy. If multi-step approximations were to be used instead (§2.4) then one would expect more strategic behaviour to emerge. For example, the sampling process might focus on exploring the space initially, before turning to exploitation in the later stages of the optimisation process — rather than spending a large number of samples exploiting every local maxima encountered.[1]

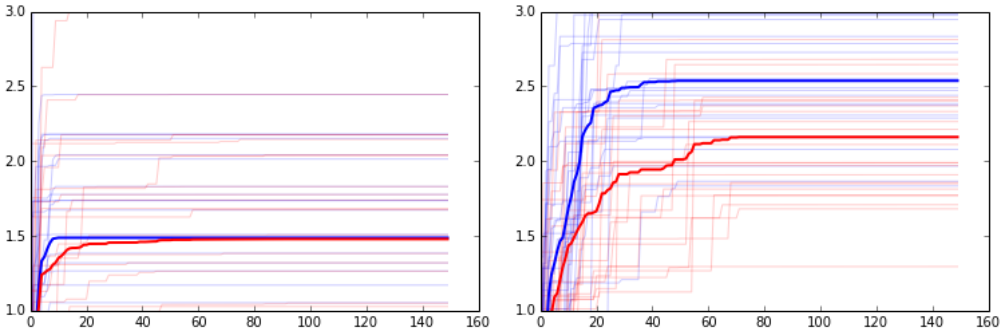
Figure 3.1 provides a simple performance comparison between GPO-EI and simulated annealing.[2] Thick lines show the average maximal function value obtained over a "large" number of trials, and can therefore be interpreted as (a Monte Carlo estimate of) the expected utility obtained by the optimisers for a range of FE budgets. Clearly simulated annealing is not the most formidable opponent to be casting comparisons against, but this brief experiment does demonstrate that GPO has the potential to perform quite well, particularly when its assumptions accurately match the objective function.[3] Of particular interest is the fact that GPO does not get stuck on local maxima, and continues to explore the space as long as it feels the need to gather more information.

These plots also demonstrate the need for benchmarking high-dimensional functions, as both optimisers perform reasonably similarly in 1-D. However, as dimensionality increases, so does the number of function evaluations required to differentiate between the optimisers, which (as discussed earlier) can be something of a challenge for GPO implementations.

[1] One could argue that EI corresponds to the assumption that the optimiser could be terminated at any time without warning, and so greedily exploiting maxima whilst it has the chance is the sensible thing to do under the circumstances.

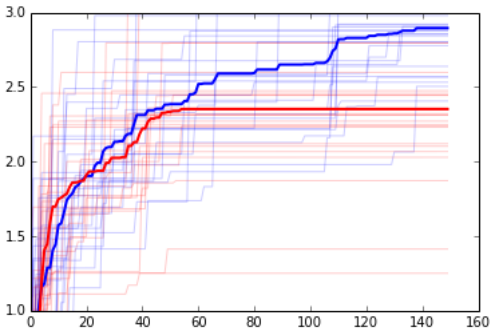
[2] Specifically, the `anneal` procedure provided by SciPy was used with default parameters.

[3] In this case, the objective function is sampled directly from the prior distribution on f.

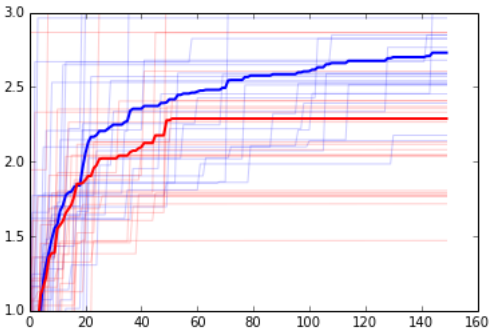


(a) 1-D

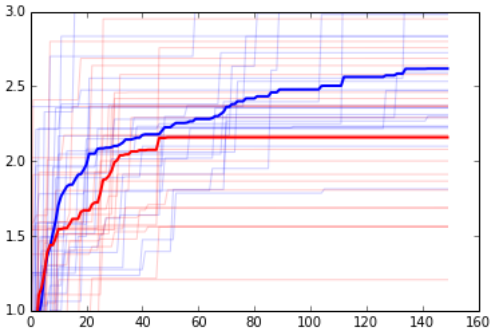
(b) 2-D



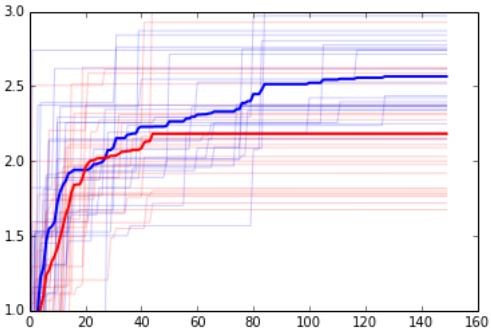
(c) 5-D



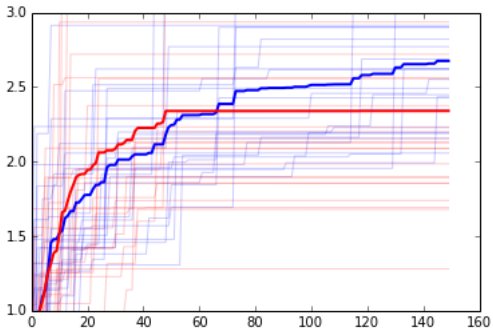
(d) 10-D



(e) 20-D



(f) 50-D



(g) 100-D

Figure 3.1: Comparison of GPO with simulated annealing, on objective functions sampled from a GP These plots show the greatest function value obtained by either optimiser (y-axis) versus the number of function evaluations used (x-axis). GPO is in blue, and simulated annealing in red. The ghosted lines show the results of 25 independent trials, with the thick lines being an average of these. In each trial, the objective function was independently sampled from a zero-mean GP with the unit-lengthscale squared-exponential kernel.

3.2 Black-Box Optimisation Benchmark

The Black-Box Optimisation Benchmark (BBOB) [39, 40, 41] is a collection of test functions that have been commonly employed in the literature as a means of benchmarking black-box optimisation algorithms. It also provides libraries for automatically running an optimisation algorithm on a number of random variations of these functions, as well as tools for producing the visualisations provided below in Figures 3.2–3.5.

3.2.1 Analysis

We begin by briefly analysing how BBOB relates to the concerns outlined in §1.2. The benchmark encodes our prior assumptions about the kinds of functions black-box optimisers should be able to handle in practice — as expressed by the prior distribution from which the objective functions are generated, as well as the utility function generated by the ranking criteria.[4] If these assumptions are an accurate reflection of real optimisation problems, then it is not necessary to be superstitious about an optimiser "overfitting" the benchmark. The empirical results (averaging over a number of trials) will essentially be a Monte Carlo estimate of the expected utility of running each optimiser, and hence the optimiser obtaining the best performance on the benchmark will also be the best choice according to MEU.

However, the question is whether the BBOB test functions are an accurate reflection of the range of difficulty that can be encountered in real optimisation problems. It is not too difficult to see that, in terms of the two sub-problems outlined in §1.2 — the minimum number of function evaluations required to extract the necessary information from the objective function, and the subsequent computational difficult of optimising the inferred model — most of the test functions should not be overly difficult to solve in principle. Specifically, all of the functions are generated from parametric models with relatively few (randomly-selected) parameters, and these models can be maximised analytically once sufficient information has been gathered in order to determine the relevant parameters.

Of course, the problem is somewhat more difficult if we assume we cannot make use of the definitions provided for the models used to generate the objective functions. However, assume that we only knew that these generative models existed and could analytically defined quite concisely. In principle, it would be possible to search over the space of such models in a manner similar to [42].[5] Obviously actually implementing this was beyond the scope of this project, but it illustrates the fact that the functions included in benchmarks such as BBOB likely do not capture the full complexity of optimisation problems that may be encountered in practice.

3.2.2 Results

Testing was performed utilising the standard BBOB framework and scripts. Due to time constraints, the optional 40-D objective functions were not considered. A number of trials were performed, with the objective function being randomly generated at the beginning of each trial. Each trial began by providing GPO with a single observation uniformly sampled from the $[-4, 4]^D$ hypercube, and any further samples chosen by GPO were constrained to lie within the $[-5, 5]^D$ hypercube. For each of these samples, GPO was provided with the corresponding value of the objective function. The GPO implementation terminated and returned the best point observed so far when it either reached the FE budget, found a maximum[6] greater than the target value provided by BBOB, or was unable to find a point to sample with non-negligible EI. EI was considered "negligible" when it fell below one standard deviation of the numerical noise assumed by the GP model, in this case $\sigma_{\text{noise}} = 10^{-3}$. This was done to prevent GPO from continuing to sample known local maxima on top of or very close to existing samples. In the case of early termination, BBOB continued to restart GPO from different initial samples until it exhausted its entire function evaluation budget.

[4] This ranking may not be entirely trivial, as it depends on how one combines all of the information from the various plots and statistics in order to assign preferences between different algorithms. To simplify things, we might simply consider how many trials in which the optimiser "solved" the maximum of the objective function to within a specified threshold after a fixed budget of FEs. In a real application, this would likely be set to more accurately model problem-specific concerns.

[5] Or RIES: `http://mrob.com/pub/ries/`

[6] It should be noted that BBOB phrases optimisation in terms of minimisation, so function values were always negated when being passed in or out of the GPO implementation.

Results are presented in Figures 3.2–3.5. From these plots, it can be seen that GPO performs respectably in the low-budget case considered here, being able to solve a reasonable proportion of the objective functions to within a threshold set by other state-of-the-art optimisers previously run on the benchmark (Figures 3.2–3.4), and often requiring fewer function evaluations on average (Figure 3.5).

It is interesting to compare these results with those of [8], who were able to obtain much better performance with a similar model. There were two main differences between their implementation and that utilised in this project. Firstly, [8] used an (isotropic) Matérn kernel instead of the simpler squared-exponential kernel used for this project. However, experimentation performed during this project revealed little difference in performance between the two isotropic kernels, at least for the GPO implementation we were using.

[8] also claim that the isotropic kernel performed better in practice than the axis-aligned elliptical kernel used here, as the greater number of free parameters in the latter caused the model to be more prone to overfitting by maximum-likelihood. However, this claim was also unable to be replicated for this project, with the more flexible kernel obtaining noticeably better performance than the isotropic one (at least for the squared-exponential case). In fact, the isotropic kernel was not observed to perform significantly better than when all hyper-parameters were fixed to one.

For the sake of brevity, no evidence to support these observations is provided here, as it was beyond the scope of this project to perform a rigorous comparison of the different kernels. However, it suffices to say that it would be of interest to perform such a comparison more thoroughly than provided here.

The second difference between the two implementations is that [8] spent much more time and effort maximising EI, by applying the DIRECT and state-of-the-art CMA-ES global optimisers to the subsidiary optimisation problem. By the authors' own admission, this played a significant role in the success of their implementation. The fact that this single point of difference results in such a large discrepancy in performance shows that the EI-optimisation problem deserves much more attention than it currently receives. For GPO to depend so strongly on a *competing* black-box optimiser is a rather unsatisfactory solution.

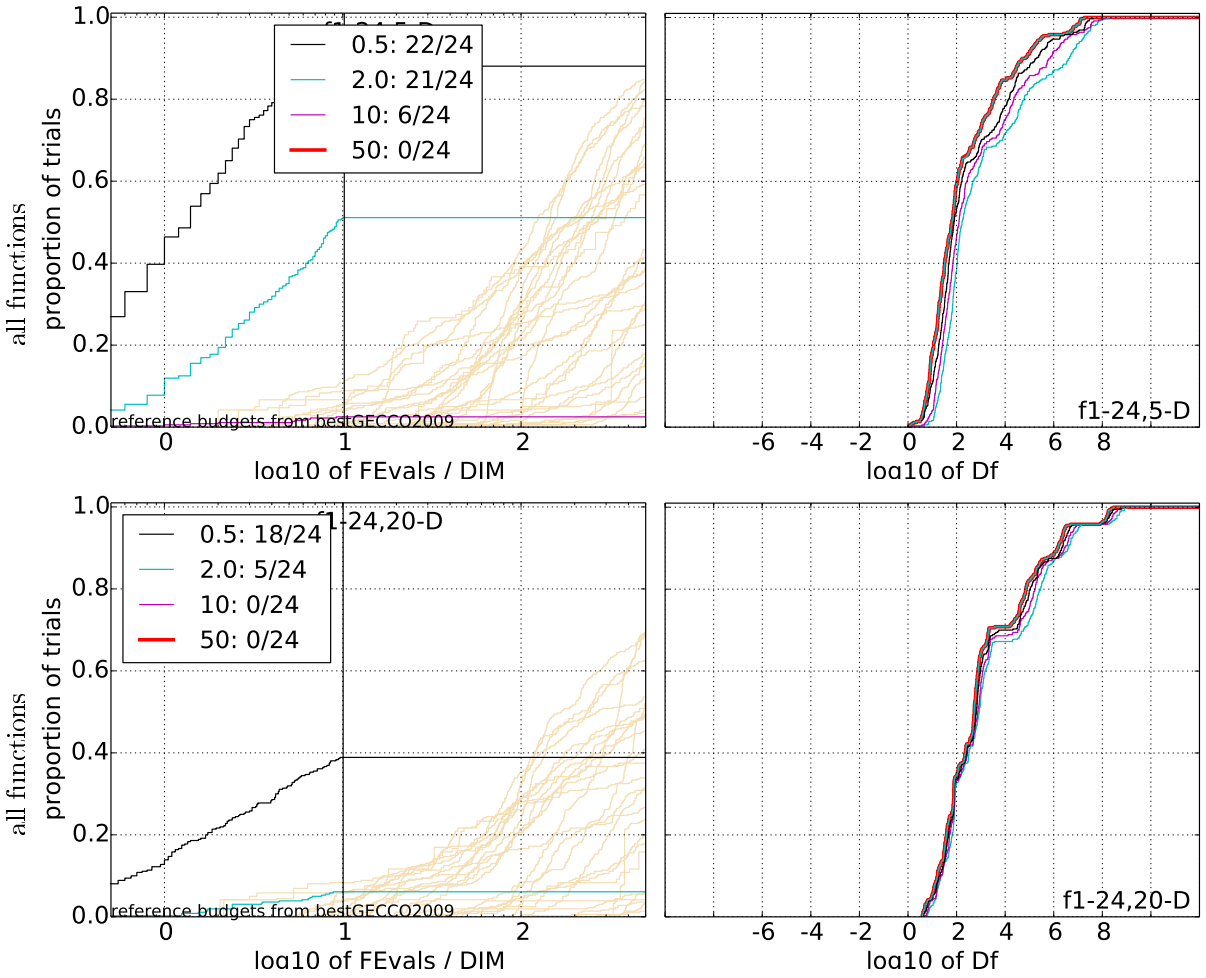


Figure 3.2: Empirical cumulative distribution functions (ECDF), plotting the fraction of trials with an outcome not larger than the respective value on the x-axis. Left subplots: ECDF of number of function evaluations (FEvals) divided by search space dimension D, to fall below $f_{\mathrm{opt}} + \Delta f$ where Δf is the target just not reached by the GECCO-BBOB-2009 best algorithm within a budget of $k \times$ DIM evaluations, where k is the first value in the legend. Legends indicate for each target the number of functions that were solved in at least one trial within the displayed budget. Right subplots: ECDF of the best achieved Δf for running times of $0.5D, 1.2D, 3D, 10D, 100D, 1000D, \ldots$ function evaluations (from right to left cycling cyan-magenta-black...) and final Δf-value (red), where Δf and Df denote the difference to the optimal function value. Light brown lines in the background show ECDFs for the most difficult target of all algorithms benchmarked during BBOB-2009.
The top row shows results for 5-D and the bottom row for 20-D.

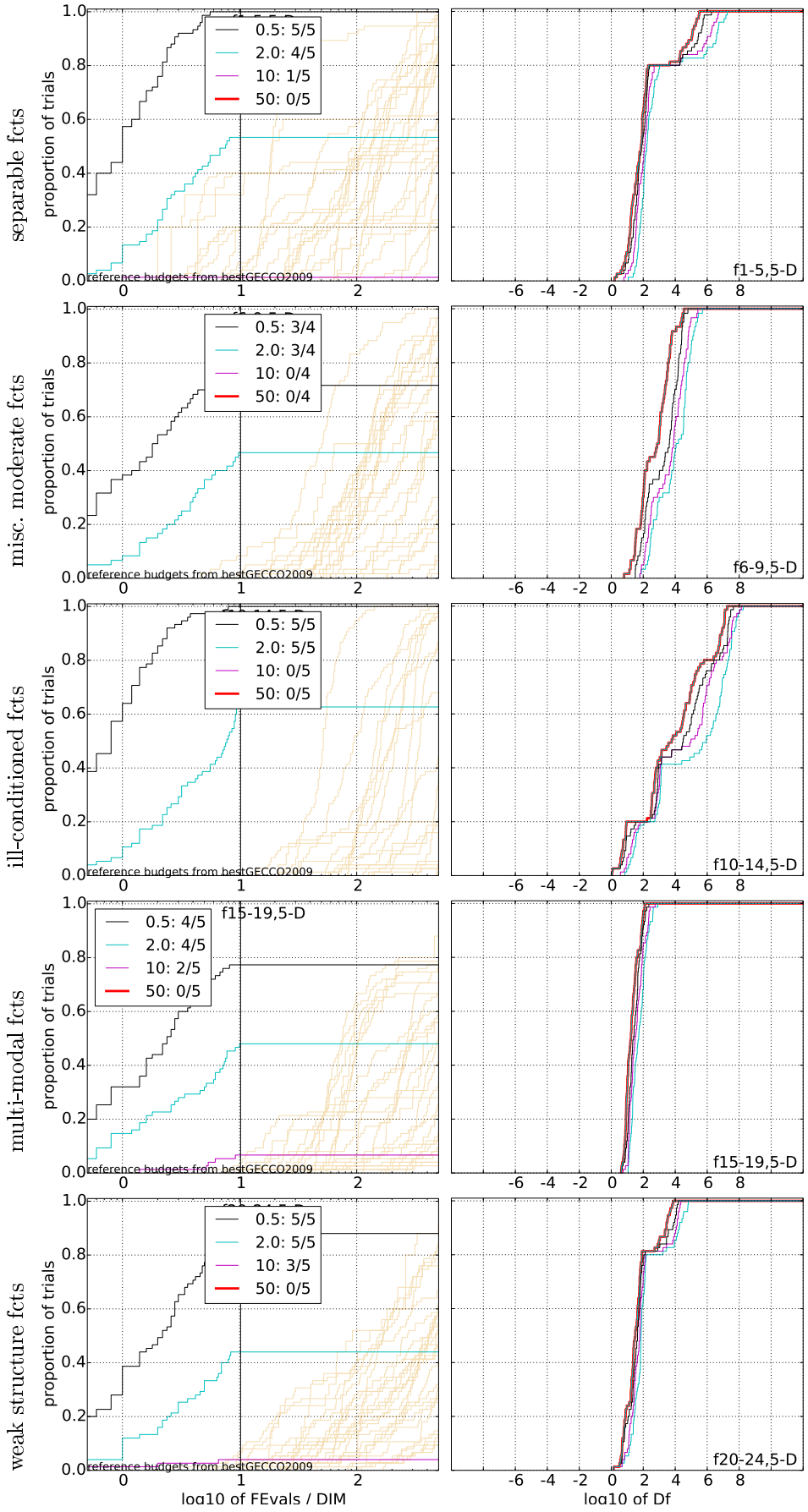


Figure 3.3: Subgroups of functions 5-D. See caption of Figure 3.2 for details.

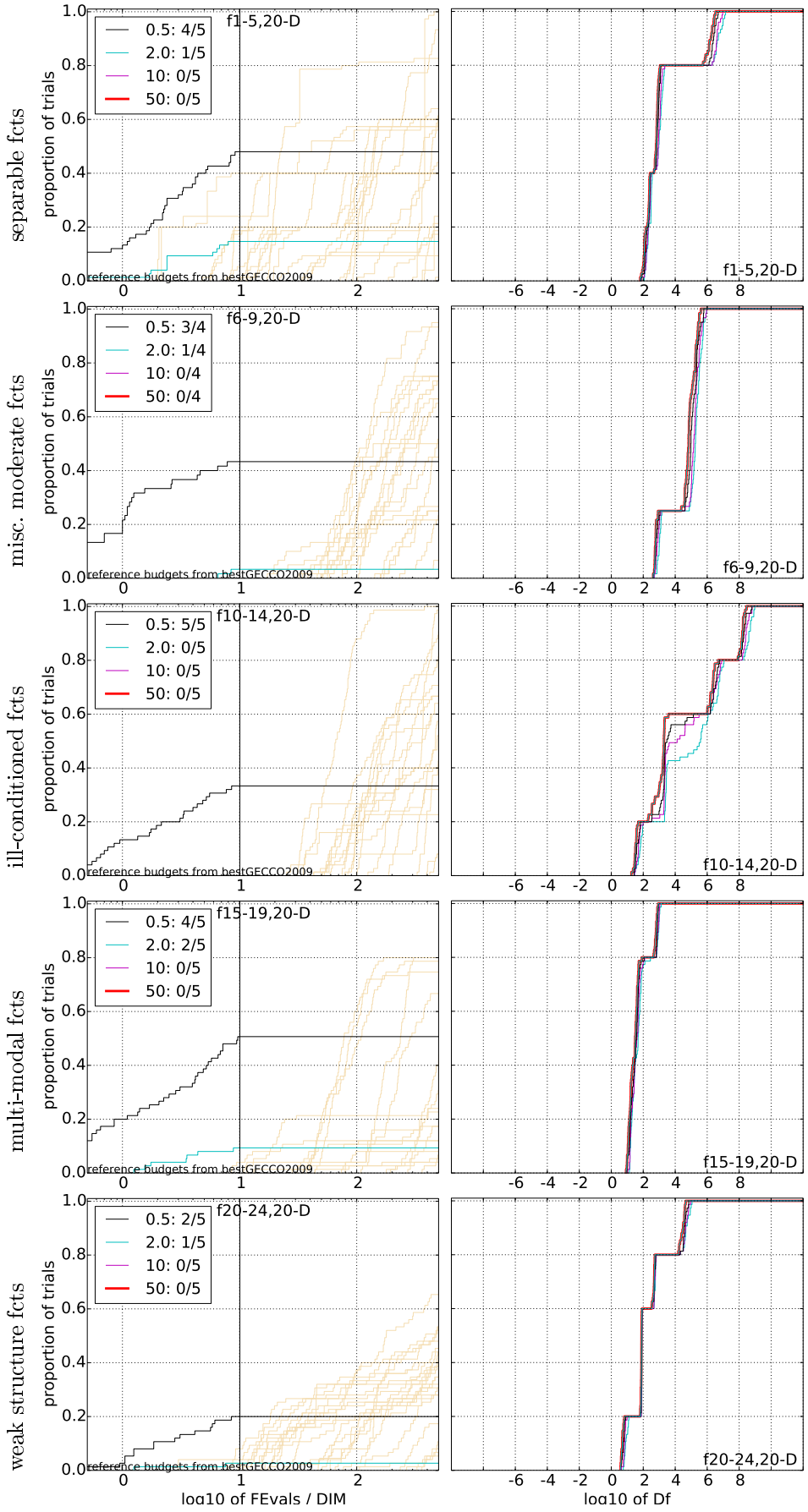


Figure 3.4: Subgroups of functions 20-D. See caption of Figure 3.2 for details.

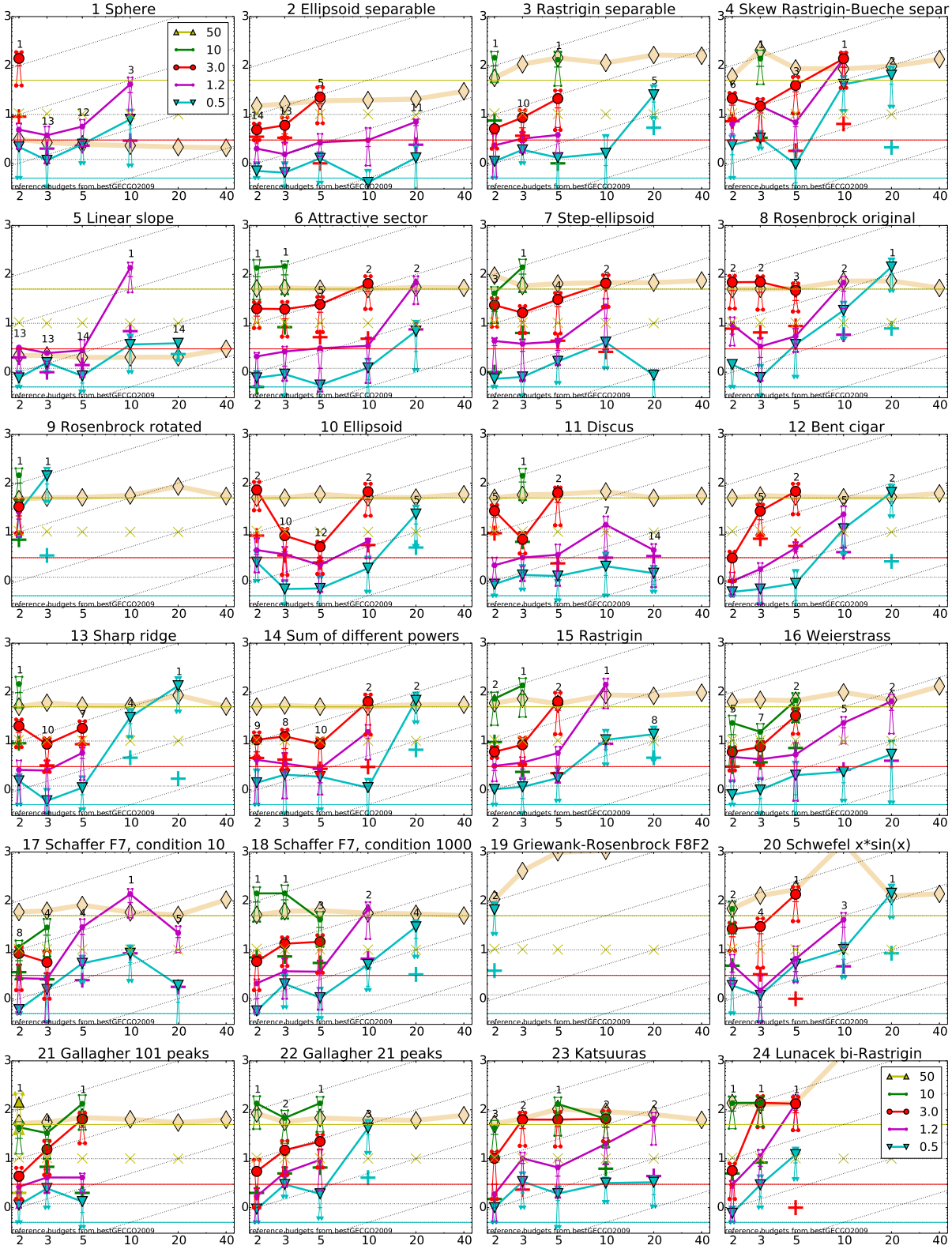


Figure 3.5: Expected number of f-evaluations (ERT, lines) to reach $f_{\mathrm{opt}} + \Delta f$; median number of f-evaluations ($+$) to reach the most difficult target that was reached not always but at least once; maximum number of f-evaluations in any trial ($\times$); interquartile range with median (notched boxes) of simulated runlengths to reach $f_{\mathrm{opt}} + \Delta f$; all values are divided by dimension and plotted as $\log_{10}$ values versus dimension. Shown is the ERT for targets just not reached by the GECCO-BBOB-2009 best algorithm within the given budget kDIM, where k is shown in the legend. Numbers above ERT-symbols indicate the number of trials reaching the respective target. The light thick line with diamonds indicates the respective best result from BBOB-2009 for the most difficult target. Slanted grid lines indicate a scaling with $\mathcal{O}(\mathrm{DIM})$ compared to $\mathcal{O}(1)$ when using the respective 2009 best algorithm.

3.2.3 Convergence

Figures 3.6 and 3.7 illustrate the convergence behaviour of the EI and log-likelihood functions respectively. It can be seen from Figure 3.6 that, generally speaking, conjugate gradient (with random restarts) has relatively more trouble converging to a unique EI maximum in the early stages, tapering off towards the end. This is not to necessarily say that there are fewer local maxima later on, but simply that they are not spread throughout the input space as much. It is possible that the local maxima of EI tend to concentrate into small regions of the space, making uniform random restarts much less effective.[7]

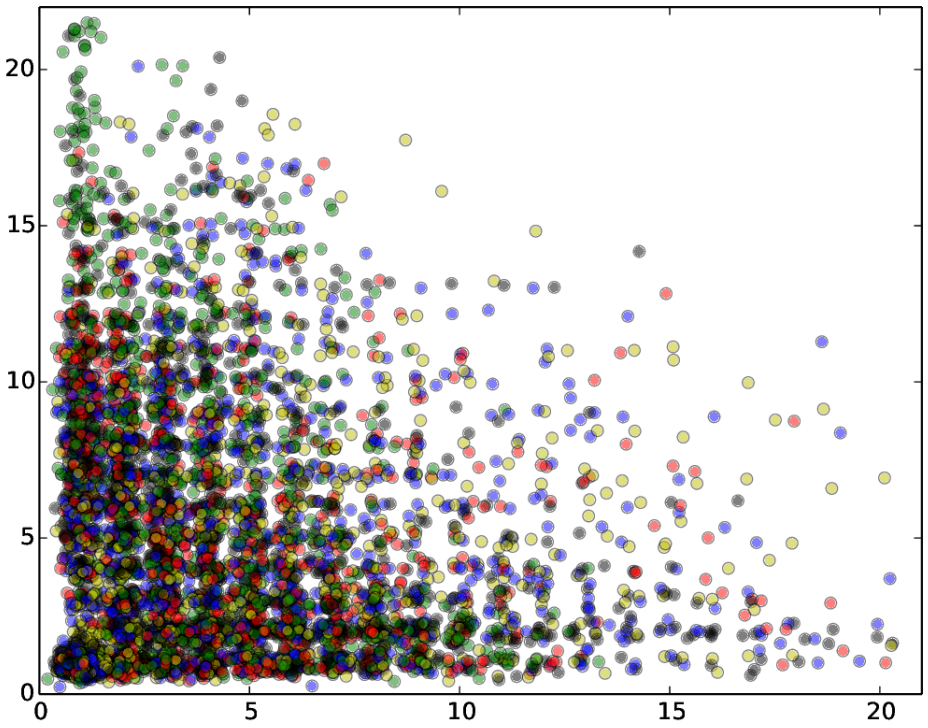
Differences in behaviour between groups of objective functions becomes much more apparent in higher dimensions, particularly the 20-D case. Functions 1–14 tend to have a large number of convergence points initially, with 1–9 causing GPO to terminate soon afterwards, in contrast to 10–14 in which GPO continues sampling but observes fewer maxima. On the other hand, functions 15–24 tend to have a relatively lower number of convergence points, and GPO takes many more samples before terminating.

Figure 3.7 shows that the log-likelihood has a much higher number of convergence points in comparison to EI, and does not appear to decrease as quickly over time. These plots do not display how much the heights of the convergence points differ from one another, so it is possible that the number of significantly different local maxima is much lower than shown in the plots. However, it does demonstrate that the assumption of a unique peak in the likelihood required for ML and MAP methods cannot be justified without a more thorough investigation of the likelihood surface. Once again, differences between objective functions are most apparent in 20-D, with the ratio of convergence points similar to that observed in Figure 3.6.

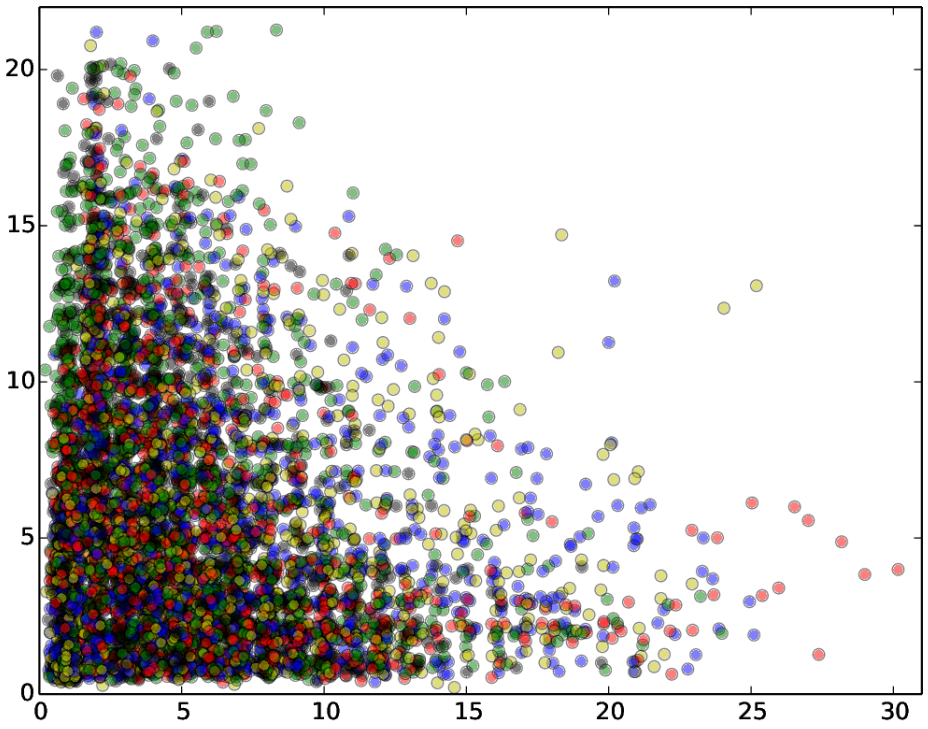
Another interesting feature to be noted from these plots is the much higher density of points towards the left. This means that a large proportion of the trials terminated before reaching the FE budget, due to GPO being unable to find a potential sample with non-negligible EI. In fact, depending on the type of objective function, 35–55% of all trials terminated immediately after the initial sample. Without further investigation it is difficult to locate the exact cause of this, but a possible scenario is that these are the cases where the initial sample is so large that mean $\mu = 0$ and signal variance $\theta_0 = 1$ assumptions cause GPO to arrive at the conclusion that this is most likely the maximum of the function. If this is the case, relaxing these assumptions would likely result in an improvement, taking care to avoid additional problems due to overfitting.

Figure 3.1 demonstrates that GPO does not have this early-termination problem when its assumptions match the actual structure of the objective function more closely. It therefore seems reasonable to conclude that such behaviour is a symptom of a significant mismatch between the model and reality. The assumptions and implementation details used in this project would need to be revised in order to prevent the optimiser from running out of potential actions so quickly.

[7] [4] suggest restarting from previously sampled points, but care needs to be taken to avoid CG getting stuck in the sharp local minima at these points — it seems a small amount of random perturbation is necessary to do so.

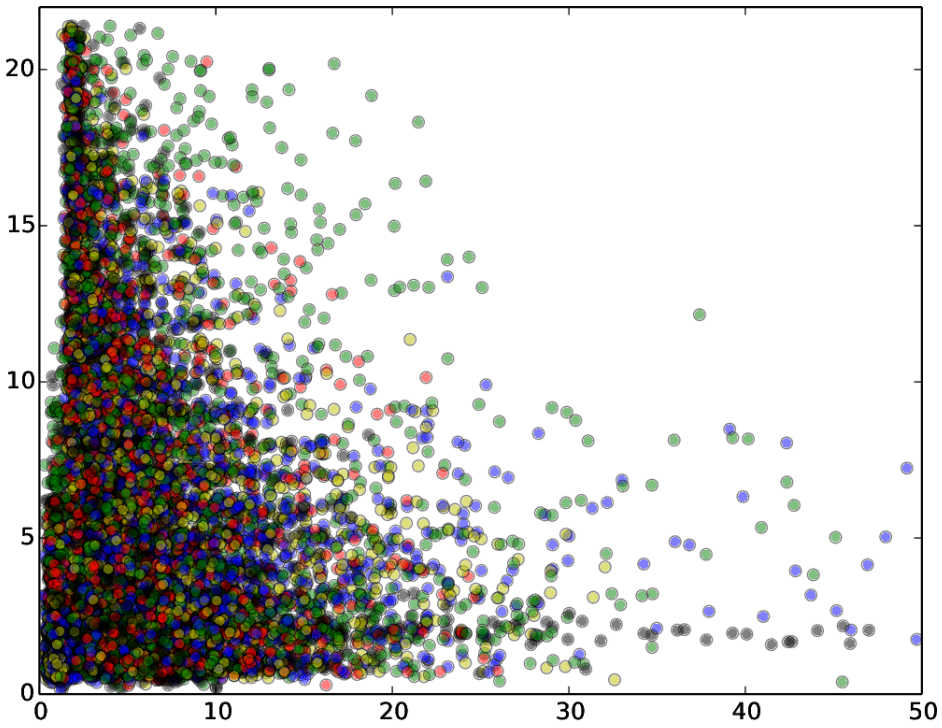


(a) 2-D

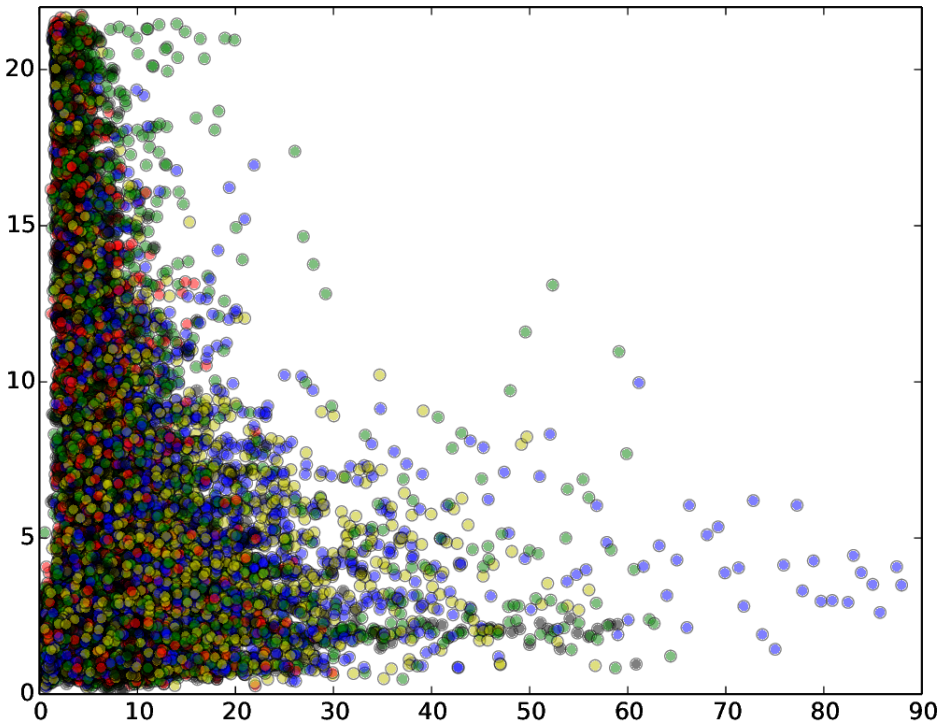


(b) 3-D

Figure 3.6: (continued on next page)

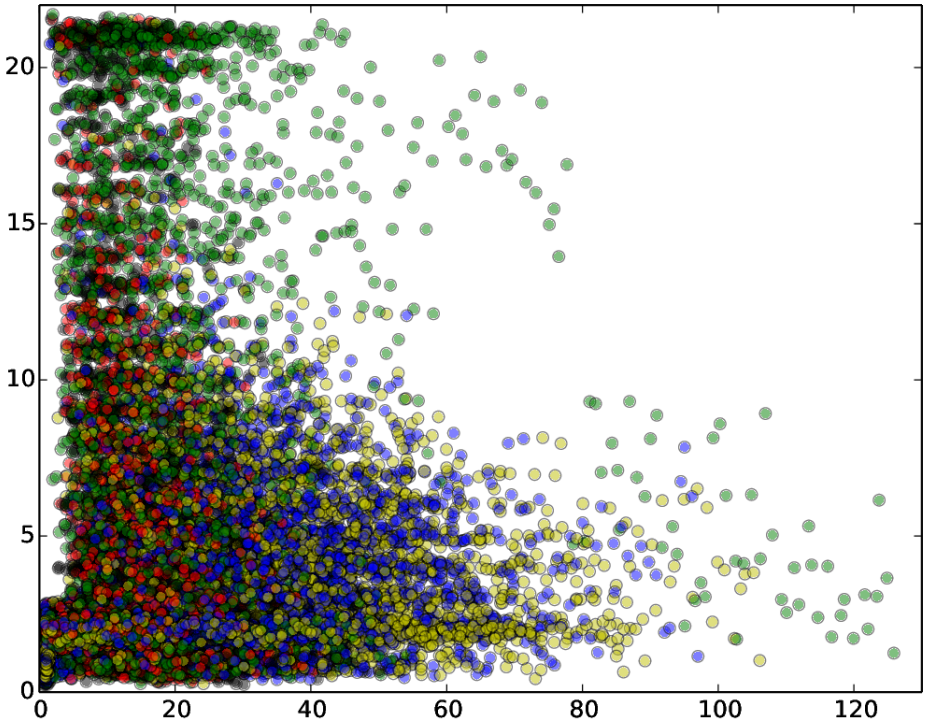


(c) 5-D



(d) 10-D

Figure 3.6: (continued on next page)



(e) 20-D

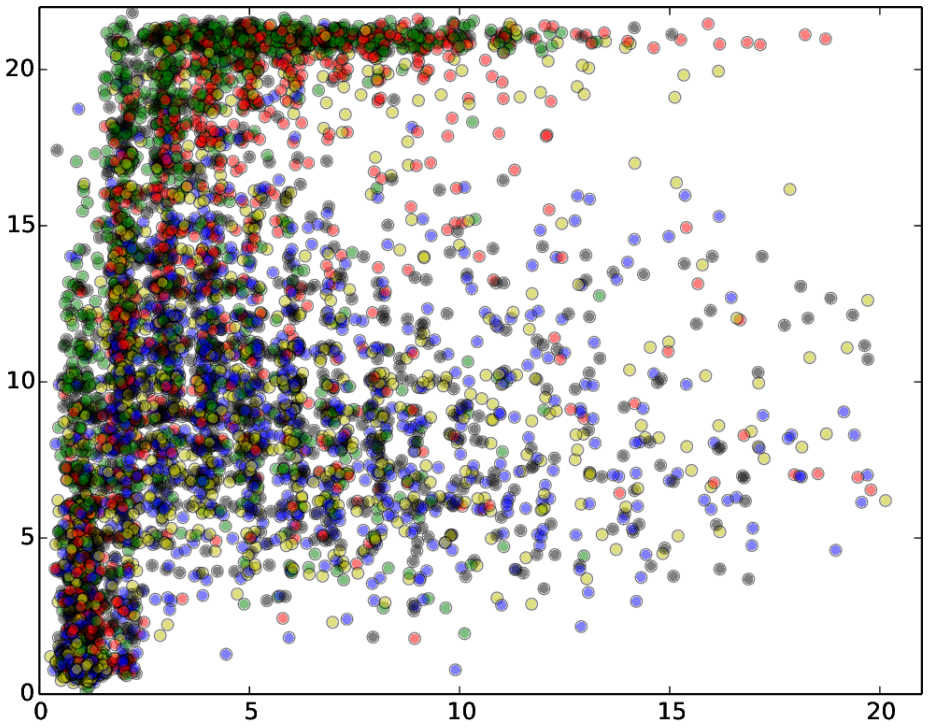
Figure 3.6: Convergence points of EI surface

The number of convergence points (y-axis) is defined as the number of local maxima observed across random restarts with heights differing by more than 10^{-3}, versus the number of function evaluations that have been observed by GPO so far (x-axis).

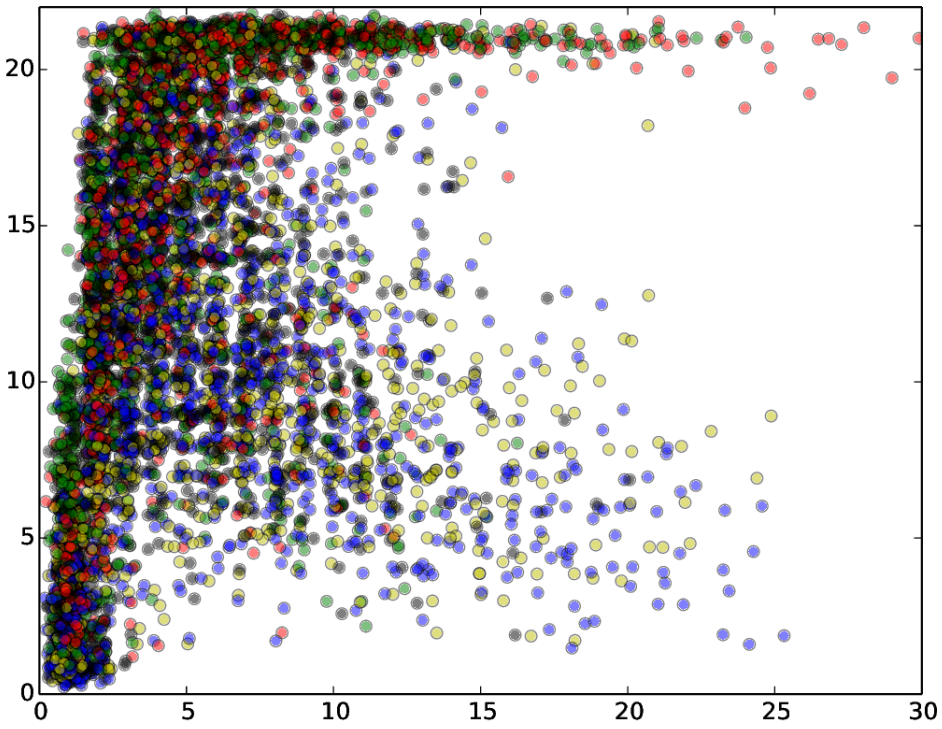
This threshold of 10^{-3} is probably too small if the maxima are all quite large, but it is enough to give us a rough impression of the structure of the function. More thorough analysis would be required for more rigorous conclusions.

Each group of objective functions is given a different colour: functions 1–5 (separable) in **black**, 6–9 (low or moderate conditioning) in **red**, 10–14 (high conditioning and unimodal) in **green**, 15–19 (multi-modal with adequate global structure) in **blue**, and 20–24 (multi-modal with weak global structure) in **yellow**.

A small amount of random jitter has been added to the plots to aid visibility.

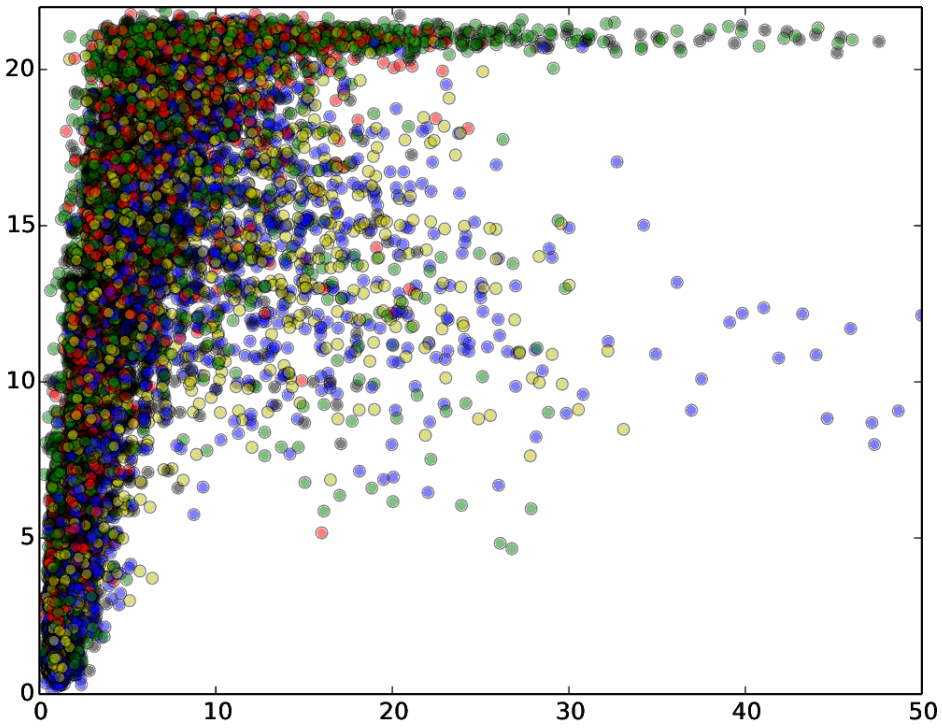


(a) 2-D

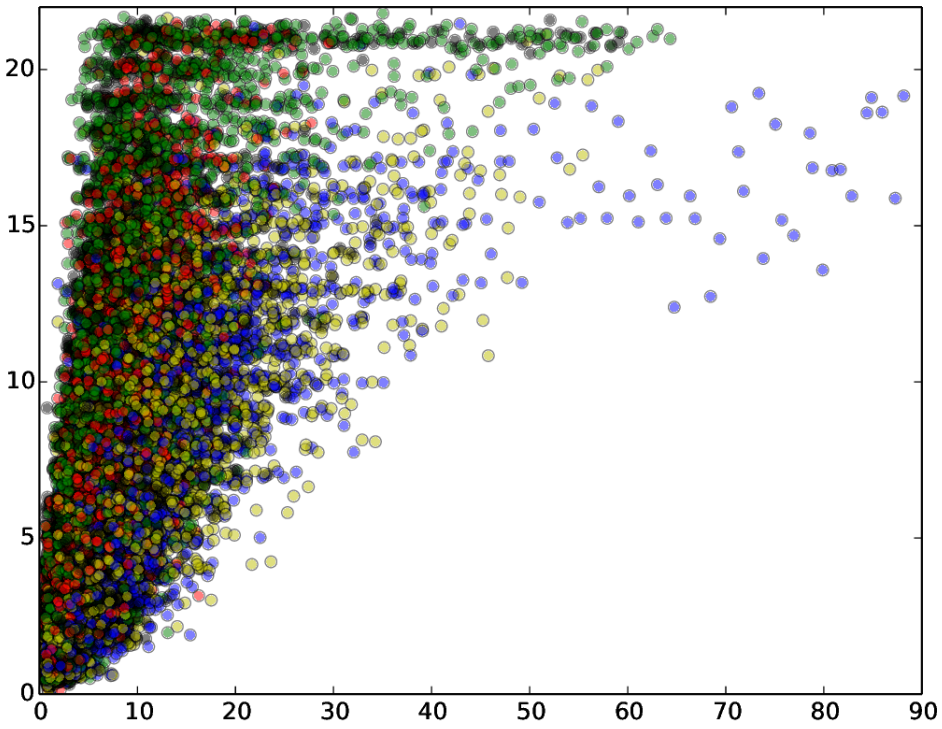


(b) 3-D

Figure 3.7: (continued on next page)

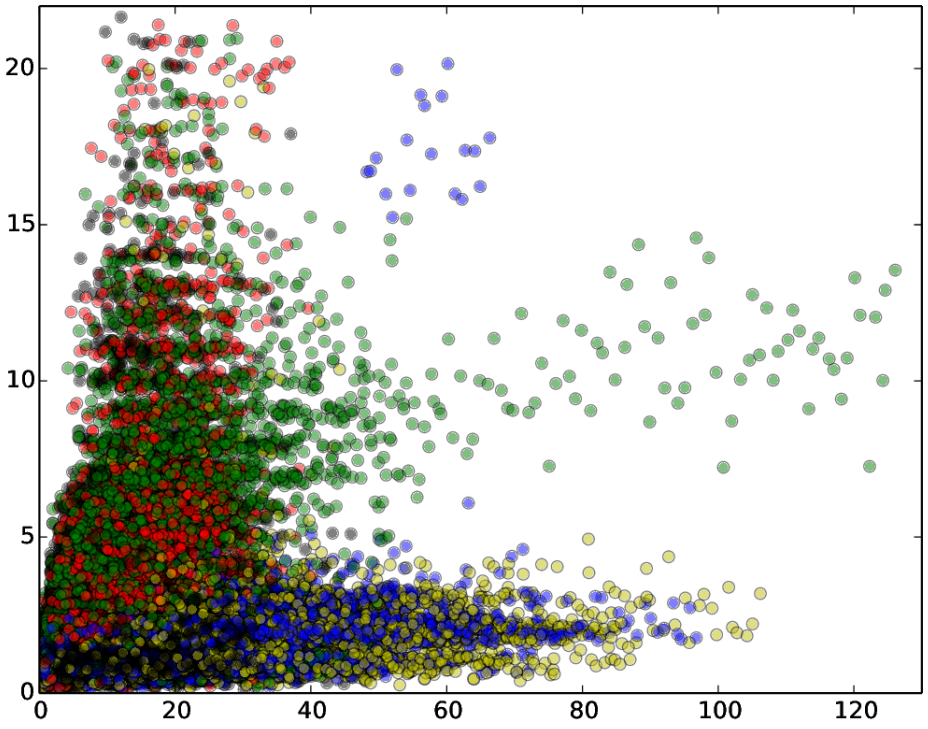


(c) 5-D



(d) 10-D

Figure 3.7: (continued on next page)



(e) 20-D

Figure 3.7: Convergence points of log-likelihood surface
See the caption of Figure 3.6 for details.

Chapter 4

Conclusions and Future Work

The experimental results obtained in this project showed that, whilst being quite a simple model at its core, Gaussian process-based optimisation (GPO) is capable of performing respectably on high-dimensional optimisation tasks, especially when only a small budget of function evaluations is available. However, a number of deficiencies and practical difficulties were uncovered, which suggest several avenues for future research, in the hopes of making GPO a more competitive and practical candidate for black-box optimisation.

A particularly concerning problem that was encountered during this project was the fact that GPO often terminated itself after a relatively small number of function evaluations. Since this problem is not observed when the objective function closely matches GPO's prior assumptions, it seems reasonable to conclude that this behaviour indicates a large disparity between the assumptions and reality. There are a number of ways in which these assumptions can be improved.

The most obvious improvement is to alter the mean and covariance functions of the Gaussian process (GP) prior. The mean function could simply be left constant but inferred from the data, or we could imbue it with prior knowledge about the general structure of the objective surface. A number of standard kernel functions are available to encode different assumptions about covariance structure, and it is possible to combine several kernels to encode even more powerful assumptions [9, 10, 11]. There is also the potential for greater flexibility by setting a prior over a set of potential kernels and allowing the best one to be automatically inferred from the data, an idea common in hierarchical Bayesian modelling. Further still, one may wish to partition the space into a number of loosely correlated regions, as a high-dimensional analogue to changepoint detection [15, 16].

A related concern is in assigning appropriate priors to any unknowns in the model, particularly kernel hyper-parameters. These should be informative enough to encode any prior knowledge we have about the structure of the objective, but flexible enough to make the necessary inferences from observed data. There are even opportunities for sharing this information across similar problem instances via hierarchical modelling.

Attention also needs to be paid to how these variables are inferred computationally. In particular, maximum likelihood methods — and to a lesser but still significant extent, maximum *a posteriori* methods — are prone to overfitting, particularly in the initial stages of the optimisation process when few datapoints have been observed. A solution to this may simply be to provide GPO with a moderately large sample of data from the beginning, but a more flexible approach would be to utilise more accurate inference methods, such as Monte Carlo integration.

Naive inference methods also lead to scalability problems, particularly the ability to tractably handle large numbers of function evaluations. A number of potential solutions to this have been proposed elsewhere, including sparse matrices and the corresponding Markov networks [19, 20], matrix decomposition and approximation methods [21], active sets [1, §8], and incremental updating of matrices.

There is also a great deal of room for improvement in the methods utilised for any subsidiary optimisation problems encountered, such as maximising expected improvement (EI). It would be of interest to more thoroughly investigate the structure of the EI surface (and likelihood surfaces too for that matter) in order to determine any structural information that could be exploited for improved accuracy and computational efficiency. By re-interpreting decision-making as a special case of inference [31], it may even be possible to apply many powerful inference algorithms directly to this problem.

At a slightly higher level, there are a number of possible variations to the problem set-up described in this report. To start with, the utility function being used here is quite simple, and does not necessarily capture the behaviour one would expect in a practical problem. In particular, it does not make any reference to computational resource usage, costs of function evaluations, the actual utility of different function values, or penalties on large discrepancies between predictions and reality, to name a few. Many of these details are necessary to consider in practice, so it would be helpful to analyse a variety of such utility functions, in order to determine their efficacy as well as the practical implications on implementing the model tractably.

The ability to handle different observation models, particularly additive noise, is another important practical requirement. A number of *ad hoc* modifications to the decision-making process have been suggested elsewhere, but these have trouble adapting to changes in other parts of the model. Therefore, it is helpful to reconsider the relevant implementation details from first principles (such as the brief derivation in §2.5) as this provides a greater amount of clarity as to why these modifications are necessary, as well as taking into account the relevant details of the model as a whole.

Casting GPO as a partially observable Markov decision process (POMDP) [37] is another potentially fruitful direction to pursue, as it allows general-purpose techniques and approximations to be applied without needing to re-invent the wheel, so to speak. In particular, multi-step lookahead approximations such as Monte Carlo tree search (MCTS) [35] could allow the optimiser to make much more intelligent and strategic decisions.

Looking at the problem of global optimisation more generally, there are a few potential areas to investigate. Firstly, it would be helpful to compare optimiser implementations on a variety of different benchmarks more reflective of the prior assumptions commonly encountered in real problems. Constructing such benchmarks is a difficult problem in itself, and one should not expect any single benchmark to be universally applicable. However, we should at least consider benchmarks that go beyond parametric models whose maximum can be derived analytically. Otherwise, it is unlikely that the benchmark will capture the full spectrum of difficulties that need to be handled in reality.

Having said that, there are many real problems — such as those generated by physical systems — in which the objective function could be accurately modelled by a relatively simple parametric model. In such cases, it would be desirable for an optimiser to exploit this fact. Conceptually at least, we simply need to replace the GP prior on the objective with a more suitable distribution. This would be relatively simple if we had prior knowledge of the exact form of the appropriate parametric model, but this need not be the case. In general, we could make progress by pairing a sufficiently expressive prior with clever computational techniques for searching the space of such models [42].

In addition to this, further analysis of existing state-of-the-art optimisation algorithms from a Bayesian perspective would be beneficial, focussing in particular on separating their (implicit) core assumptions from their implementation details. Doing so could allow these methods to be generalised and adapted to new problems more easily. For example, it may be illuminating to further investigate the connection between optimisers such as CMA-ES and variational methods for Bayesian inference [33, 34].

Finally, there are a number of general computational issues whose solution would be hugely beneficial to all problems involving decision-making under uncertainty. At the forefront of these is the problem of semi-automatically translating abstract models such as GPO into an efficient concrete implementation. A prerequisite to this is extending the utility function to penalise excessive computation, as well as any other costly actions such as sampling of the objective function, so that the appropriate level of approximation can be formally determined.

In order to generate a space of potential implementations for a given abstract model, we would need a library of automated implementation and approximation strategies, as well as the symbolic and analytic manipulation routines required to apply these methods to the model to derive a concrete implementation. Of course, it is a non-trivial matter to analyse the accuracy and runtime efficiency of any given implementation in advance of its execution. A brute-force approach would be to simply execute each implementation a number of times, simulating observations by sampling from the prior distribution specified by the model, and averaging utilities to approximate the expectation.

However, such an expensive procedure would likely be intractable, so a more sophisticated framework for selecting implementations would be required in practice. A potential approach to this problem is to note that the implementation process can be interpreted as a decision-making-under-uncertainty problem in itself. More specifically, we are required to select a number of implementation techniques to apply to the abstract model in order to eventually obtain a concrete implementation, but our limited

computational resources mean that can never be certain that we have selected the "best" implementation (in the sense of maximising expected utility). Perhaps if we can express this problem as a POMDP, then it may be possible to apply a range of existing techniques and approximations (such as MCTS) to derive a solution.

Further work would be required in order to make this tractable for as many practically useful models as possible. As a start in this direction, perhaps we can exploit the fact that a software system is usually comprised of a network of interacting sub-components, whose precise implementation details can often be isolated from one another. It may therefore be possible to confine the computational impact of specific approximations to individual components of the system. By constructing a (suitably simplified) model of how approximations affect the propagation of uncertainty throughout the system, it could be possible to aggregate all of these effects in order to roughly infer the overall accuracy of the implementation, without significant computational expense in doing so. A greater deal of research would be required to determine if these ideas prove useful in practice.

Bibliography

[1] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. University Press Group Limited, Jan 2006. [Online]. Available: http://gaussianprocess.org/gpml/

[2] E. Brochu, V. M. Cora, and N. de Freitas, "A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning," Dec 2010. [Online]. Available: http://arxiv.org/abs/1012.2599

[3] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *NIPS*, P. L. Bartlett, F. C. N. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., Aug 2012, pp. 2960–2968. [Online]. Available: http://papers.nips.cc/paper/4522-practical

[4] M. R. Frean and P. Boyle, "Using gaussian processes to optimize expensive functions," in *Australasian Conference on Artificial Intelligence*, ser. Lecture Notes in Computer Science, W. Wobcke and M. Zhang, Eds., vol. 5360. Springer Science + Business Media, Nov 2008, pp. 258–267. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-89378-3_25

[5] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, "Theano: a CPU and GPU math expression compiler," in *Proceedings of the Python for Scientific Computing Conference (SciPy)*, Jun. 2010, oral Presentation. [Online]. Available: http://deeplearning.net/software/theano/

[6] E. Jones, T. Oliphant, P. Peterson *et al.*, "SciPy: Open source scientific tools for Python," 2001–. [Online]. Available: http://www.scipy.org/

[7] M. Mudge, "Efficient glocal optimization for expensive functions," Honours thesis, Victoria University of Wellington, 2010.

[8] F. Hutter, H. Hoos, and K. Leyton-Brown, "An evaluation of sequential model-based optimization for expensive blackbox functions," in *GECCO (Companion)*, C. Blum and E. Alba, Eds. ACM, 2013, pp. 1209–1216. [Online]. Available: http://coco.gforge.inria.fr/lib/exe/fetch.php?media=pdf2013:w0311-hutter.pdf

[9] T. Robinson, "Evaluate and improve an optimization algorithm," Honours thesis, Victoria University of Wellington, 2011.

[10] D. K. Duvenaud, H. Nickisch, and C. E. Rasmussen, "Additive gaussian processes," in *NIPS*, J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. C. N. Pereira, and K. Q. Weinberger, Eds., Dec 2011, pp. 226–234. [Online]. Available: http://papers.nips.cc/paper/4221-additive-gaussian-processes

[11] D. K. Duvenaud, J. R. Lloyd, R. B. Grosse, J. B. Tenenbaum, and Z. Ghahramani, "Structure discovery in nonparametric regression through compositional kernel search," in *ICML (3)*, ser. JMLR Proceedings, vol. 28. JMLR.org, May 2013, pp. 1166–1174. [Online]. Available: http://jmlr.org/proceedings/papers/v28/duvenaud13.html

[12] Z. Wang, M. Zoghi, F. Hutter, D. Matheson, and N. de Freitas, "Bayesian optimization in a billion dimensions via random embeddings," Jan 2013. [Online]. Available: http://arxiv.org/abs/1301.1942

[13] J. Djolonga, A. Krause, and V. Cevher, "High-dimensional gaussian process bandits," in *NIPS*, C. J. C. Burges, L. Bottou, Z. Ghahramani, and K. Q. Weinberger, Eds., 2013, pp. 1025–1033. [Online]. Available: http://papers.nips.cc/paper/5152-high

[14] R. Garnett, M. A. Osborne, and P. Hennig, "Active learning of linear embeddings for gaussian processes," Oct 2013. [Online]. Available: http://arxiv.org/abs/1310.6740

[15] R. Adams and D. MacKay, "Bayesian online changepoint detection," Oct 2007. [Online]. Available: http://arxiv.org/abs/0710.3742

[16] R. Garnett, M. A. Osborne, and S. J. Roberts, "Sequential bayesian prediction in the presence of changepoints," vol. 382, pp. 1–8, Dec 2009. [Online]. Available: http://dx.doi.org/10.1145/1553374.1553418

[17] K. B. Petersen and M. S. Pedersen, *The matrix cookbook*, Nov 2012. [Online]. Available: http://www2.imm.dtu.dk/pubdb/views/publication_details.php?id=3274

[18] J. Friedman, T. Hastie, and R. Tibshirani, "Sparse inverse covariance estimation with the graphical lasso," *Biostatistics*, vol. 9, no. 3, pp. 432–441, Jul 2008. [Online]. Available: http://dx.doi.org/10.1093/biostatistics/kxm045

[19] F. Lindgren, H. Rue, and J. Lindström, "An explicit link between gaussian fields and gaussian markov random fields: the stochastic partial differential equation approach," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 73, no. 4, pp. 423–498, Sep 2011. [Online]. Available: http://dx.doi.org/10.1111/j.1467-9868.2011.00777.x

[20] M. Wytock and J. Z. Kolter, "Sparse gaussian conditional random fields: Algorithms, theory, and application to energy forecasting," in *ICML (3)*, ser. JMLR Proceedings, vol. 28. JMLR.org, 2013, pp. 1265–1273. [Online]. Available: http://jmlr.org/proceedings/papers/v28/wytock13.html

[21] S. Ambikasaran, D. Foreman-Mackey, L. Greengard, D. W. Hogg, and M. O'Neil, "Fast direct methods for gaussian processes and the analysis of nasa kepler mission data," Mar 2014. [Online]. Available: http://arxiv.org/abs/1403.6015

[22] J. R. Bunch and C. P. Nielsen, "Updating the singular value decomposition," *Numerische Mathematik*, vol. 31, no. 2, pp. 111–129, Jun 1978. [Online]. Available: http://dx.doi.org/10.1007/BF01397471

[23] M. Osborne, "Bayesian gaussian processes for sequential prediction, optimisation and quadrature," Ph.D. dissertation, 2010. [Online]. Available: http://www.robots.ox.ac.uk/~mosb/papers/full_thesis.pdf

[24] D. J. C. MacKay, "Introduction to gaussian processes," 1998. [Online]. Available: http://www.inference.phy.cam.ac.uk/mackay/gpB.pdf

[25] D. Kalman, "A singularly valuable decomposition: The SVD of a matrix," 2002. [Online]. Available: http://www.math.umn.edu/~lerman/math5467/svd.pdf

[26] S. Streltsov and P. Vakili, "A non-myopic utility function for statistical global optimization algorithms," *Journal of Global Optimization*, vol. 14, no. 3, pp. 283–298, 1999. [Online]. Available: http://dx.doi.org/10.1023/A:1008284229931

[27] J. von Neumann, O. Morgenstern, H. W. Kuhn, and A. Rubinstein, *Theory of Games and Economic Behavior (Commemorative Edition)*. Princeton University Press, Mar 2007. [Online]. Available: https://archive.org/details/theoryofgamesand030098mbp

[28] L. J. Savage, *The Foundations of Statistics*. Dover Publications, Jun 1972.

[29] J. B. Paris, *The Uncertain Reasoner's Companion: A Mathematical Perspective*. Cambridge University Press, Nov 2006.

[30] J. M. Alexander, "Decision theory meets the witch of agnesi," *Journal of Philosophy*, vol. 109, no. 12, pp. 712–727, Dec 2012. [Online]. Available: http://eprints.lse.ac.uk/45286/

[31] M. Toussaint, S. Harmeling, and A. Storkey, "Probabilistic inference for solving (PO)MDPs," Tech. Rep., Dec 2006. [Online]. Available: http://eprints.pascal-network.org/archive/00003924/

[32] N. D. Goodman, V. K. Mansinghka, D. M. Roy, K. Bonawitz, and J. B. Tenenbaum, "Church: a language for generative models," in *Proc. Uncertainty in Artificial Intelligence (UAI)*, D. A. McAllester and P. Myllymäki, Eds. AUAI Press, Jul 2008, pp. 220–229. [Online]. Available: http://uai.sis.pitt.edu/papers/08/p220-goodman.pdf

[33] D. Wierstra, T. Schaul, T. Glasmachers, Y. Sun, J. Peters, and J. Schmidhuber, "Natural evolution strategies," *Journal of Machine Learning Research*, vol. 15, pp. 949–980, 2014. [Online]. Available: http://jmlr.org/papers/v15/wierstra14a.html

[34] A. Honkela, M. Tornio, T. Raiko, and J. Karhunen, "Natural conjugate gradient in variational inference," in *ICONIP (2)*, ser. Lecture Notes in Computer Science, M. Ishikawa, K. Doya, H. Miyamoto, and T. Yamakawa, Eds., vol. 4985. Springer, Jun 2007, pp. 305–314. [Online]. Available: http://eprints.pascal-network.org/archive/00003358/

[35] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, 2012. [Online]. Available: http://dx.doi.org/10.1109/TCIAIG.2012.2186810

[36] N. Hay, S. J. Russell, D. Tolpin, and S. E. Shimony, "Selecting computations: Theory and applications," in *UAI*, N. de Freitas and K. P. Murphy, Eds. AUAI Press, Jul 2012, pp. 346–355. [Online]. Available: http://auai.org/uai2012/papers/123.pdf

[37] M. Toussaint, "The bayesian search game," in *Theory and Principled Methods for the Design of Metaheuristics*, ser. Natural Computing Series, Y. Borenstein and A. Moraglio, Eds. Springer Science + Business Media, 2014, ch. 7, pp. 129–144. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-33206-7_7

[38] D. J. C. MacKay, *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, Sep 2003. [Online]. Available: http://www.inference.phy.cam.ac.uk/itila/

[39] N. Hansen, A. Auger, S. Finck, and R. Ros, "Real-parameter black-box optimization benchmarking 2010: Experimental setup," INRIA, Tech. Rep. RR-7215, 2010. [Online]. Available: http://coco.gforge.inria.fr/bbob2010-downloads

[40] S. Finck, N. Hansen, R. Ros, and A. Auger, "Real-parameter black-box optimization benchmarking 2009: Presentation of the noiseless functions," Research Center PPE, Tech. Rep. 2009/20, 2009, updated February 2010. [Online]. Available: http://coco.gforge.inria.fr/bbob2010-downloads

[41] N. Hansen, S. Finck, R. Ros, and A. Auger, "Real-parameter black-box optimization benchmarking 2009: Noiseless functions definitions," INRIA, Tech. Rep. RR-6829, 2009, updated February 2010. [Online]. Available: http://coco.gforge.inria.fr/bbob2010-downloads

[42] M. Schmidt and H. Lipson, "Distilling free-form natural laws from experimental data," *Science*, vol. 324, no. 5923, pp. 81–85, Apr 2009. [Online]. Available: http://dx.doi.org/10.1126/science.1165893

[43] *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, ser. JMLR Proceedings, vol. 28. JMLR.org, 2013. [Online]. Available: http://jmlr.org/proceedings/papers/v28/

Appendix A

Sample Python Code

This appendix provides a simple Python implementation of GPO. It should be noted that this is not the final code that was used to generate the BBOB plots, and is merely provided for demonstration purposes.

A.1 Boilerplate

Imports and a few utility functions:

```
from pylab import *
from scipy.stats import norm
from scipy.spatial.distance import pdist, cdist, squareform
from scipy.optimize import anneal
from mpl_toolkits.mplot3d import Axes3D

def ascolumn(xs):
    '''Convert vectors to columns'''
    if rank(xs) < 2:
        return atleast_2d(xs).T
    return asarray(xs)

def grid2d(xmin, xmax, xnum, ymin, ymax, ynum):
    '''2-D grid as list of coordinates and meshgrid format'''
    xs = linspace(xmin,xmax,xnum)
    ys = linspace(ymin,ymax,ynum)
    # observation matrix (Nx2)
    D = array([[x,y] for x in xs for y in ys])
    X,Y = meshgrid(xs,ys)
    return D,X,Y

def eye_like(mat):
    '''Identity matrix with same dimensions as given matrix'''
    dims = mat.shape
    return eye(*dims)

def dqf(A,B):
    '''Calculate diag(A * B * A.T),
       see <stackoverflow.com/q/14758283>'''
    return einsum('ij,ij->i', dot(A,B), A)

def sameshape(a,b):
    '''Coerce a into the shape of b'''
    return asarray(a).reshape(shape(b))
```

```
def errbars(ts, mean, sd):
    '''Plot predictive distribution'''
    plot(ts, mean, 'k')
    plot(ts, mean + 2*sd, '--b')
    plot(ts, mean - 2*sd, '--b')

def norm01(Z):
    '''Normalise all entries into [0,1]'''
    zmin = Z.min(); zmax = Z.max()
    return (Z - zmin) / (zmax - zmin)

def fc_alpha(Z, A, cmap=cm.jet):
    '''Generate a colormap for Z with alpha values given by A'''
    C = cmap(Z)
    for i in xrange(C.shape[0]):
        for j in xrange(C.shape[1]):
            C[i,j][3] = A[i,j]
    return C

class GPmem(object):
    '''A function sampled from a GP, with values calculated as needed'''
    def __init__(self, dim):
        self.dim = dim
        self.d = {tuple([0]*dim) : randn()}
    def __call__(self, xs):
        if self.dim > 1: xs = atleast_2d(xs)
        else:            xs = atleast_1d(xs)
        mean,cov = mvg_cond(self.d.keys(), self.d.values(),
                            xs, full=True)
        ys = multivariate_normal(atleast_1d(mean), atleast_2d(cov))
        self.update(xs, ys)
        return ys
    def update(self, xs, ys):
        for x,y in zip(xs,ys):
            self.d[tuple(atleast_1d(x))] = y

class Fmem(object):
    '''Wraps a function to record any calls made to it'''
    def __init__(self, f, lower=-Inf, upper=Inf):
        self.xs = []
        self.ys = []
        self.f = f
        self.lower = lower
        self.upper = upper
    def __call__(self, x):
        if any(asarray(x) < asarray(self.lower)) or
           any(asarray(x) > asarray(self.upper)):
            y = -Inf
        else:
            y = self.f(x)
            self.xs.append(x)
            self.ys.append(y)
        return y
```

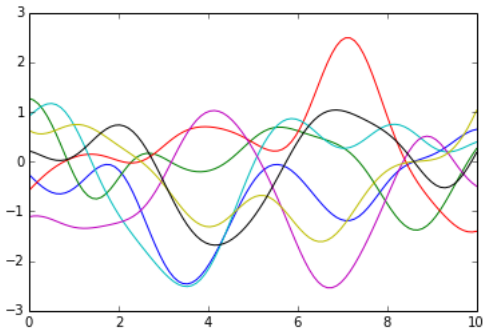
A.2 Squared Exponential Kernel

This function gives the covariance matrix $[K_{\mathrm{SE}}(\mathrm{xs}[i], \mathrm{ys}[j])]_{ij}$:

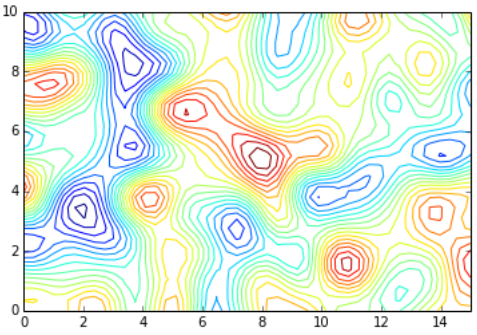
```
def K(xs, ys=None):
    if ys is None:
        d = squareform(pdist(ascolumn(xs), 'sqeuclidean'))
    else:
        d = cdist(ascolumn(xs), ascolumn(ys), 'sqeuclidean')
    return exp(-d/2)
```

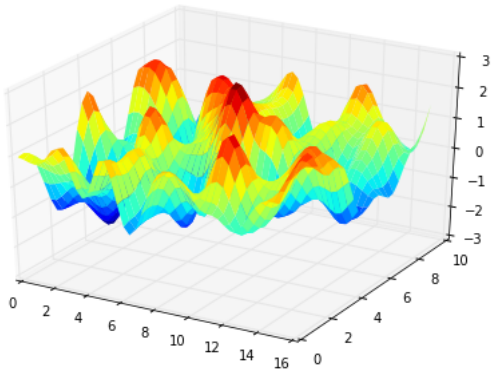
Generate some one- and two-dimensional samples from the GP with this kernel:

```
xs = linspace(0,10,100)
mean = zeros_like(xs)
cov = K(xs)
for i in range(7):
    plot(xs, multivariate_normal(mean, cov))
```




```
Nx = 45; Ny = 30
D,X,Y = grid2d(0,15,Nx, 0,10,Ny)
Z = multivariate_normal([0]*(Nx*Ny), K(D))
Z = Z.reshape(Nx,Ny).T # cols for x, rows for y
contour(X,Y,Z,20)
ax = Axes3D(figure())
ax.plot_surface(X,Y,Z, rstride=1, cstride=1, cmap=cm.jet, linewidth=0)
```





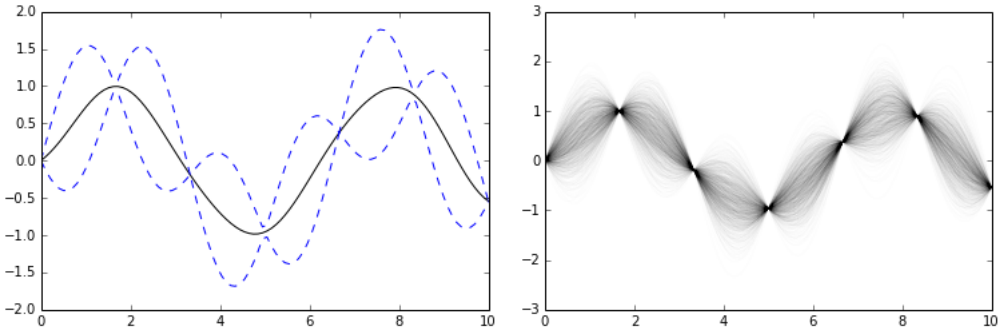
A.3 Multivariate Gaussian Conditional Distribution

This class represents the posterior distribution after observing data (xs, ys), and gives the predictive distribution of a set of test points ts:

```
class GaussCond(object):
    def __init__(self, xs, ys, noise=0.001):
        self.xs = xs
        Kxx = K(xs)
        self.KxxI = pinv(Kxx + (noise**2) * eye_like(Kxx))
        self.KxxI_ys = self.KxxI.dot(ys)

    def __call__(self, ts, full=False):
        Ktx = K(ts, self.xs)
        mean = Ktx.dot(self.KxxI_ys)
        if full: # full covariance matrix
            cov = K(ts) - Ktx.dot(self.KxxI).dot(Ktx.T)
            return mean, cov
        else: # predictive variance (diag cov)
            # assuming diag(K(ts)) is (approximately) [1,1,...]
            var = 1 - dqf(Ktx, self.KxxI)
            return squeeze(mean), squeeze(var)

def mvg_cond(xs, ys, ts, full=False, noise=0.001):
    return GaussCond(xs, ys, noise)(ts, full)
```

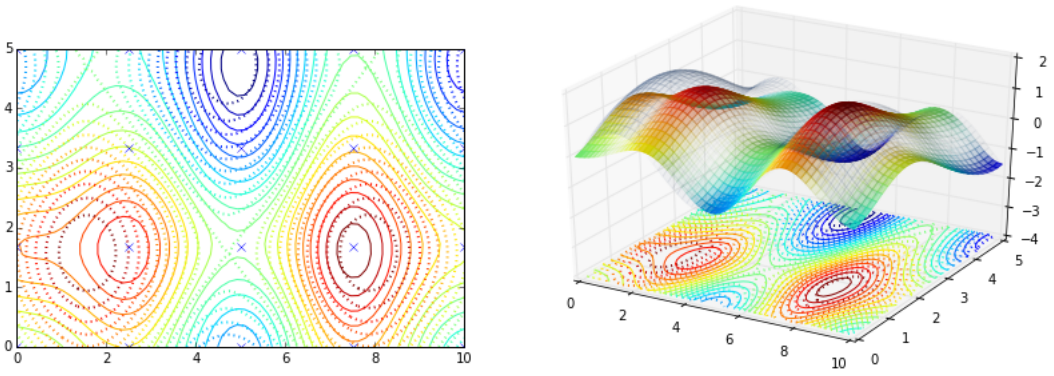
Visualise the GP predictive distribution given data generated from the 1-D function $y = \sin x$ and the 2-D function $z = \sin x + \sin y$:

```
xs = linspace(0,10,7)
ys = sin(xs)
ts = linspace(0,10,100)

mean,var = mvg_cond(xs, ys, ts)
errbars(ts, mean, sqrt(var)); show()

mean,cov = mvg_cond(xs, ys, ts, full=True)
for i in xrange(1000):
    plot(ts, multivariate_normal(mean, cov), 'k', alpha=.01)
```




```
def plot_2d_approx(X1, X2, Z, S, F):
    ax = Axes3D(figure())
    ax.plot_surface(X1,X2,Z, rstride=1, cstride=1,
                    facecolors=fc_alpha(norm01(Z), 1 - .975*(S / S.max())))
    E = absolute(F - Z)
```

```
    ax.plot_surface(X1,X2,F, rstride=1, cstride=1,
                    facecolors=fc_alpha(ones_like(F), .1*(E / E.max()), cm.Blues))
    zlo = 2 * ax.get_zlim()[0]; ax.set_zlim(bottom=zlo)
    ax.contour(X1,X2,Z,20, offset=zlo)
    ax.contour(X1,X2,F,20, offset=zlo, linestyles='dotted', linewidths=2)
    return ax


N1 = 60; N2 = 30
D,D1,D2 = grid2d(0,10,5, 0,5,4) # training data inputs
Y = sin(D[:,0]) + sin(D[:,1])   #     "       "   outputs
T,X1,X2 = grid2d(0,10,N1, 0,5,N2) # test points
mean,var = mvg_cond(D, Y, T)

Z = mean.reshape(N1,N2).T
V = var.reshape(N1,N2).T; S = sqrt(V)
F = sin(X1) + sin(X2)

plot(D[:,0], D[:,1], 'x')
contour(X1,X2,Z,20)
contour(X1,X2,F,20, linestyles='dotted', linewidths=2)

plot_2d_approx(X1, X2, Z, S, F)
```



The solid contours give the predictive mean and the dashed contours give the true objective function. On the right, the colourful surface gives the predictive mean, with transparent regions indicating large predictive variance (uncertainty). The ghosted blue surface shows the objective function, and is more opaque in regions where it differs significantly to the predictive mean.

A.4 Expected Improvement

```
def EI(mean, sd, maxpt):
    mean = asarray(mean); sd = asarray(sd); maxpt = asscalar(maxpt)
    Z = (mean - maxpt) / sd
    ei = (mean - maxpt) * norm.cdf(Z) + sd * norm.pdf(Z)
    return nan_to_num(ei)
```

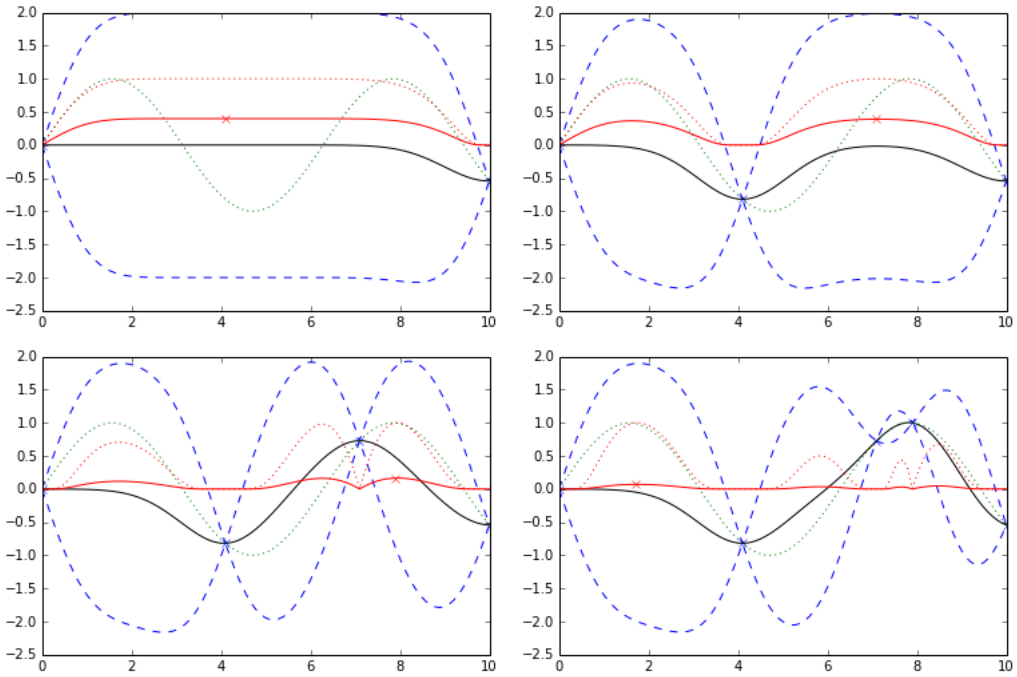
A.4.1 GPO-EI in 1-D

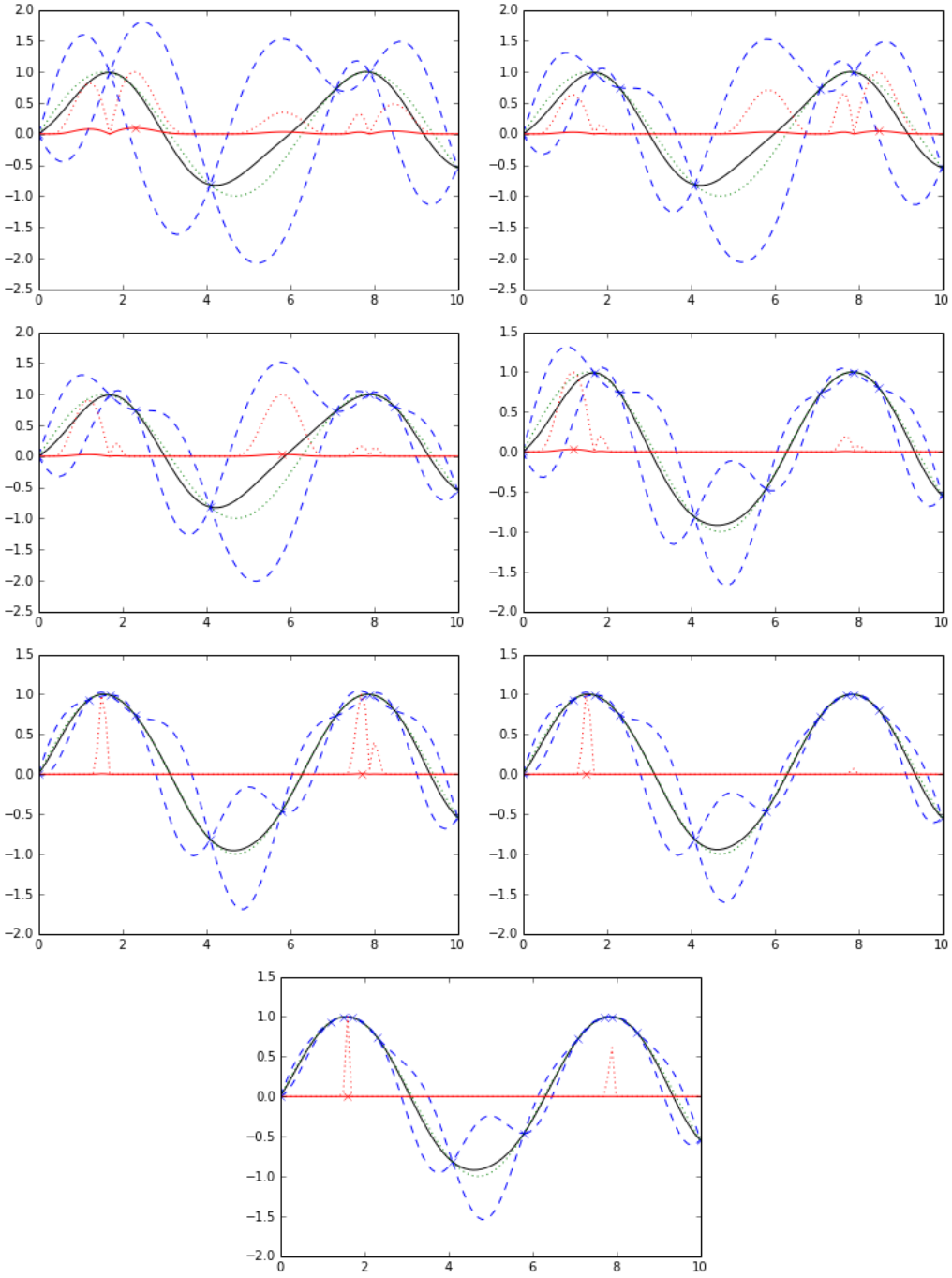
```
def gpgo_ei_1d(f, xs, ts, showit=True, noise=0.001):
    xs = list(xs); ys = map(f,xs)
    mei = Inf # maximum expected improvement
    while mei > noise:
        mean,var = mvg_cond(xs,ys,ts)
        sd = sqrt(var)
        ei = EI(mean, sd, max(ys))
        mei = max(ei); t = ts[argmax(ei)]
        if showit: # visualisation
            plot(xs,ys,'x')
            errbars(ts, mean, sd)
            plot(ts,f(ts),':')
            plot(ts, ei, 'r')
            plot(ts, ei/mei, 'r:')
            plot(t, mei, 'rx')
            show()
        xs.append(t); ys.append(f(t))
```

In the plots in this section, the solid black line gives the current predictive mean of the GP, and the dashed blue lines are two predictive standard deviations above and below. The dotted green line is the true objective function. The red lines give the current EI, with the solid line to scale, and the dotted line enlarged for greater visibility. The red cross shows the location of maximal EI, where the next sample will be taken.

A.4.1.1 Objective function $y = \sin x$

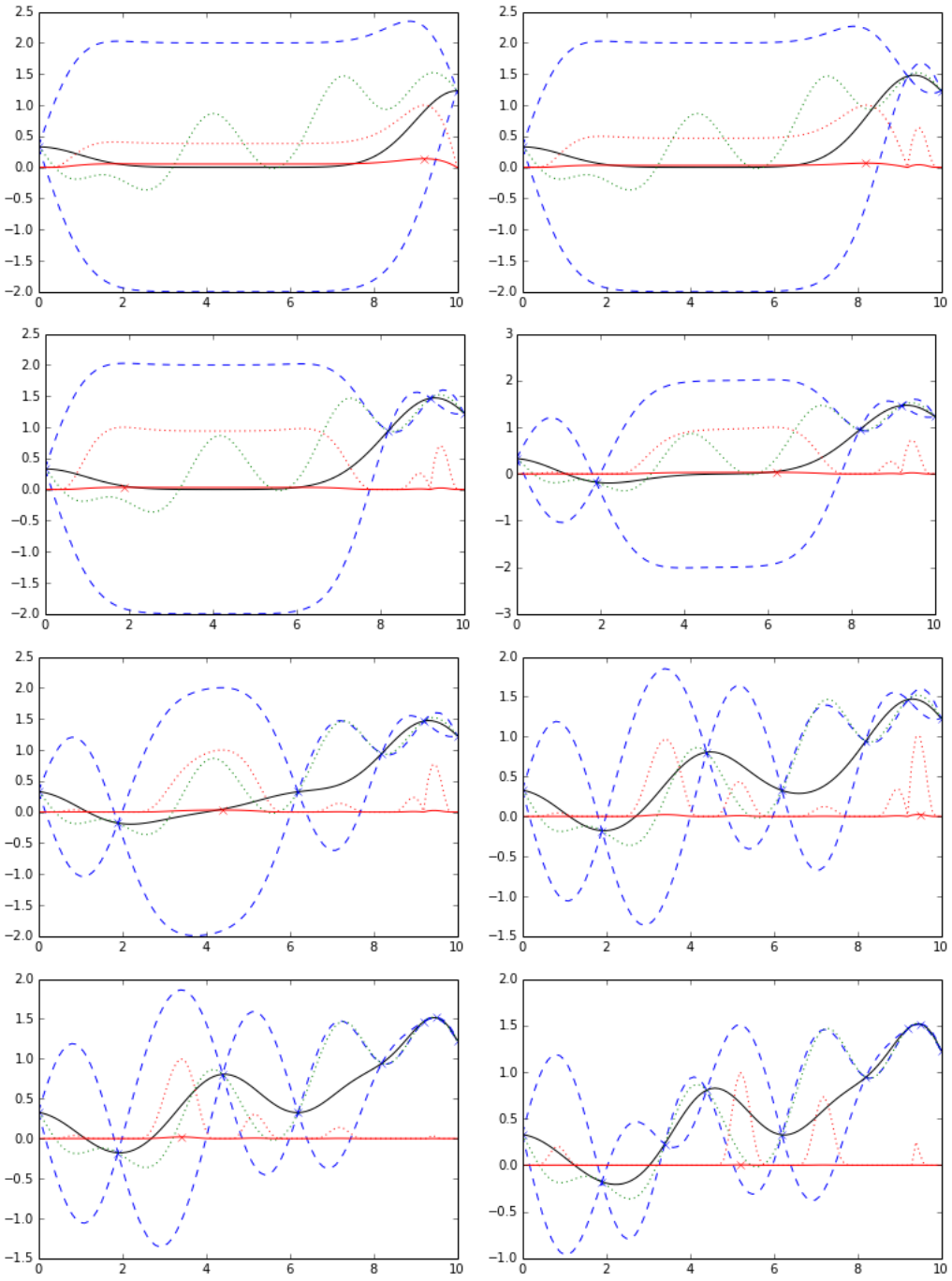
```
gpgo_ei_1d(sin, [0,10], linspace(0,10,101))
```

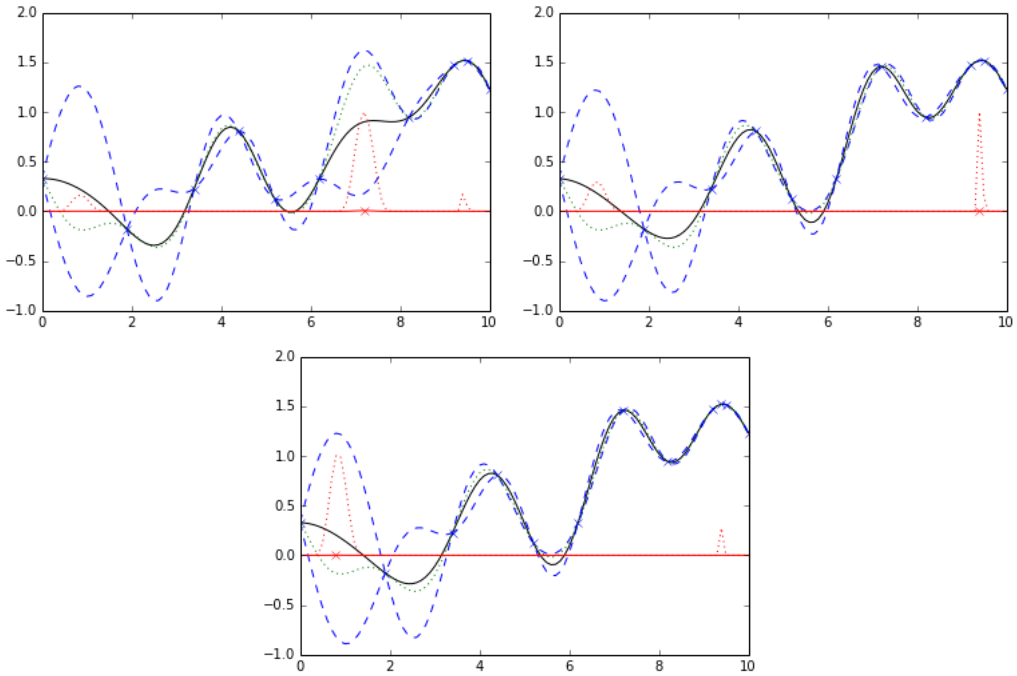




A.4.1.2 Objective function sampled from a GP

```
gpgo_ei_1d(GPmem(1), [0,10], linspace(0,10,101))
```





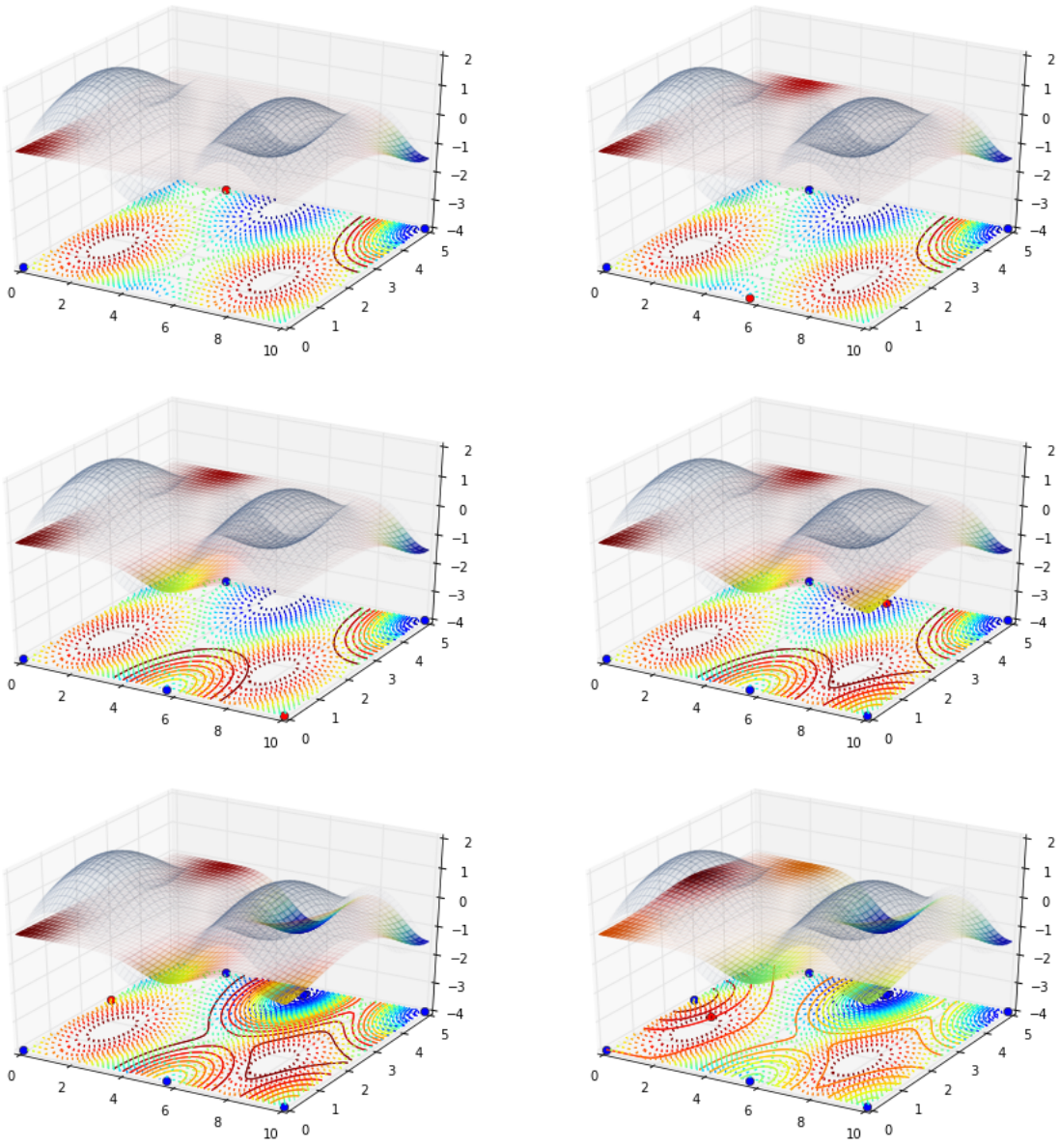
A.4.2 GPO-EI in 2-D

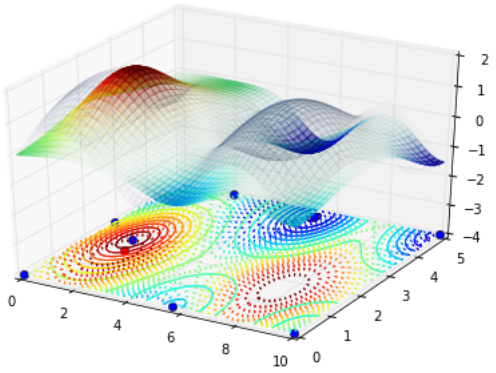
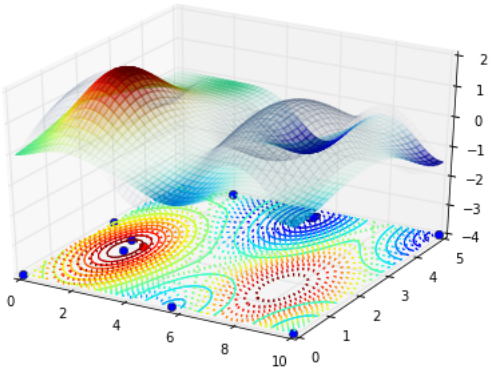
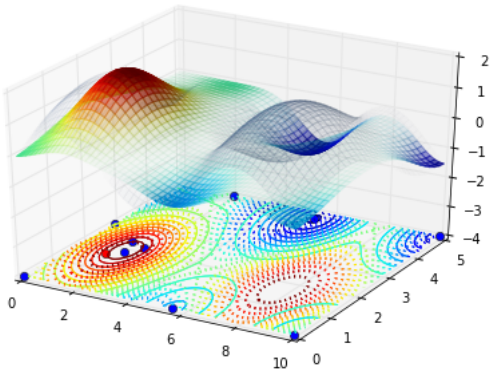
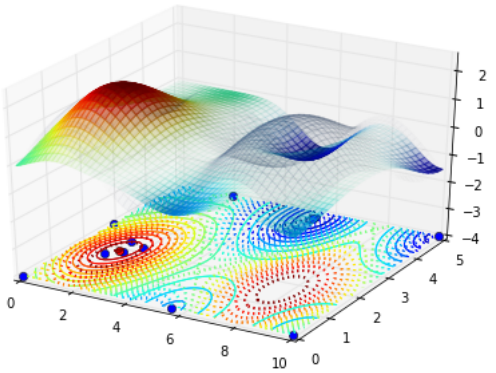
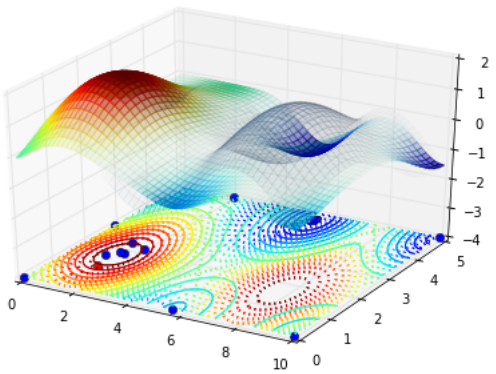
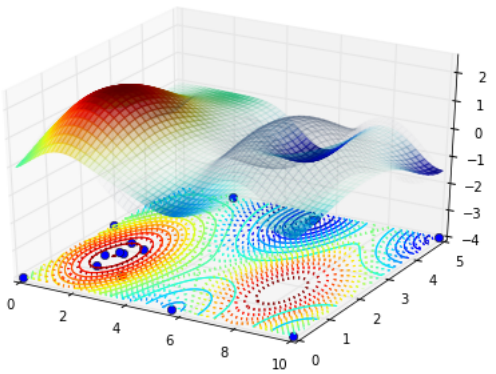
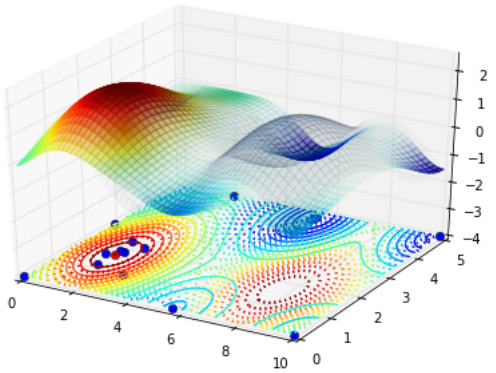
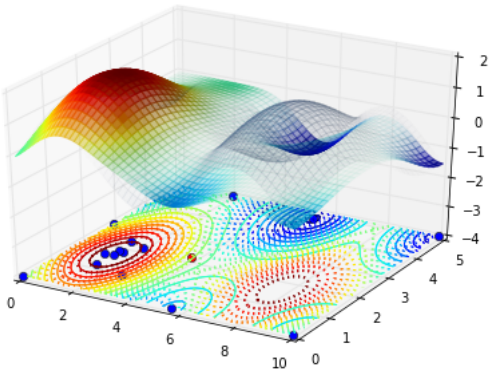
```
def gpgo_ei_2d(f, D, N1=60, N2=30, showit=True, noise=0.001):
    D = list(D); Y = list(f(D))
    mei = Inf # maximum expected improvement
    while mei > noise:
        T,X1,X2 = grid2d(0,D[1][0],N1, 0,D[1][1],N2) # test points
        mean,var = mvg_cond(asarray(D), ascolumn(Y), T)
        sd = sqrt(var)

        ei = EI(mean, sd, max(Y))
        mei = max(ei)
        t = T[argmax(ei)]

        if showit:
            Z = mean.reshape(N1,N2).T
            V = var.reshape(N1,N2).T; S = sqrt(V)
            F = f(T).reshape(N1,N2).T
            E = absolute(F - Z)

            ax = plot_2d_approx(X1, X2, Z, S, F)
            zlo = ax.get_zlim()[0]
            ax.plot(asarray(D)[:,0], asarray(D)[:,1], zlo, 'o')
            ax.plot([t[0]], [t[1]], zlo, 'or')
            ax.contour(X1,X2,Z,20, offset=zlo)
            ax.contour(X1,X2,F,20, offset=zlo, linestyles='dotted', linewidths=2)
            show()

        D.append(t); Y.append(f(atleast_2d(t)))
```

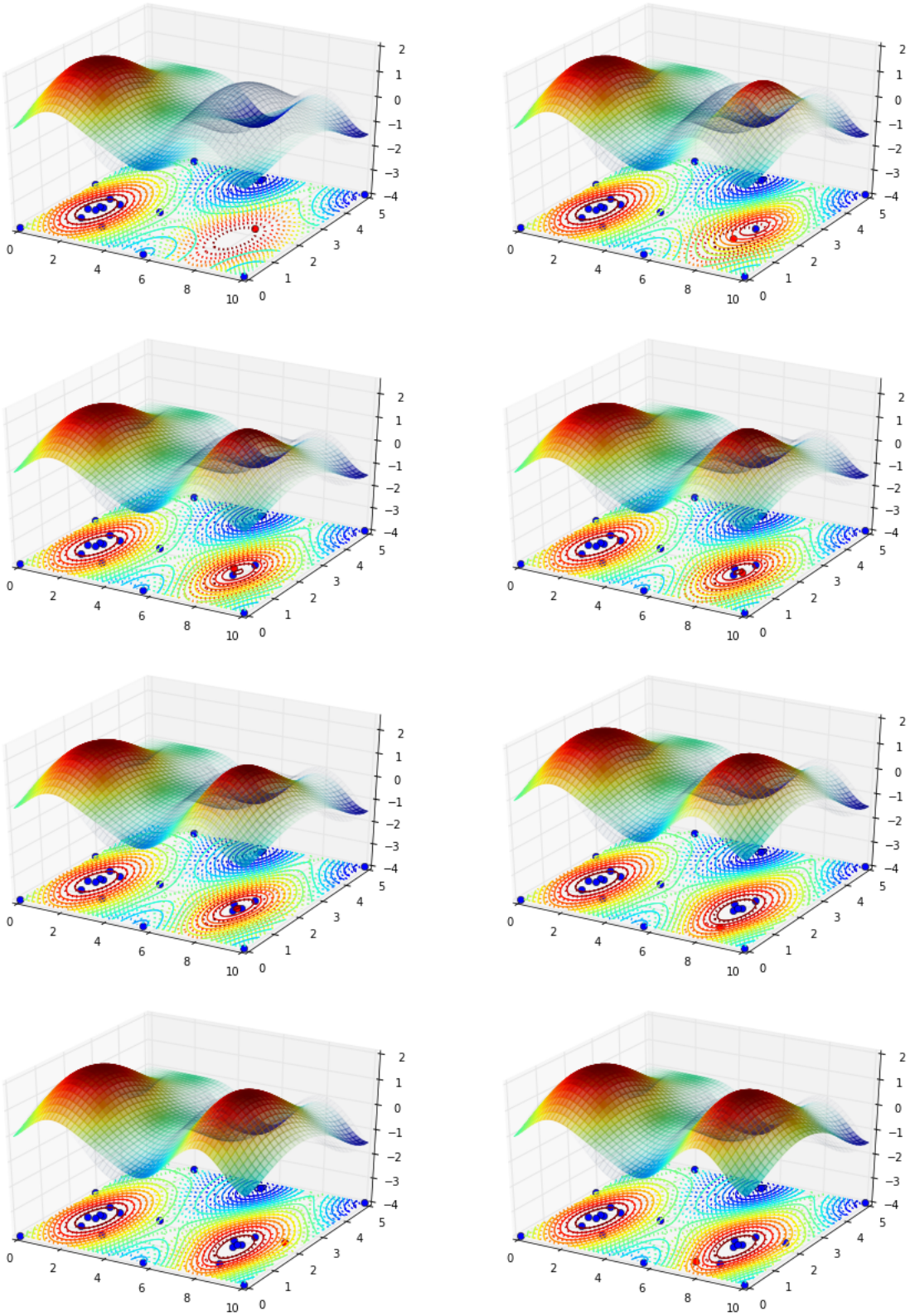
The plots in this section follow the same conventions as those shown earlier. Unfortunately, EI is not shown in order to reduce the amount of clutter, but one can draw an analogy to the 1-D plots given earlier — EI is zero immediately surrounding previous samples (blue circles), and large in regions that have large mean and/or high variance (transparent surface). The red circles show the location of the maximal EI.

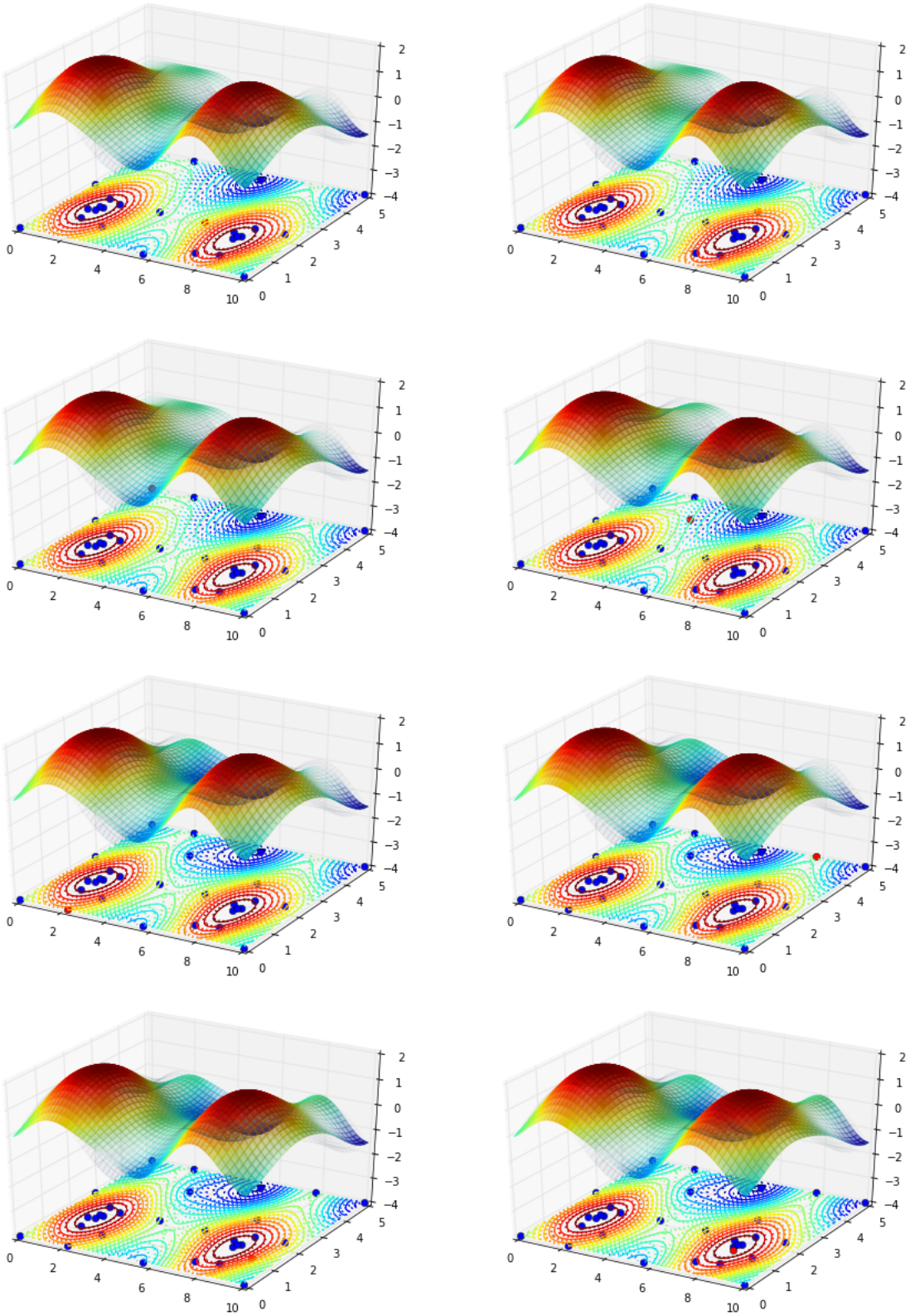
A.4.2.1 Objective function $z = \sin x + \sin y$

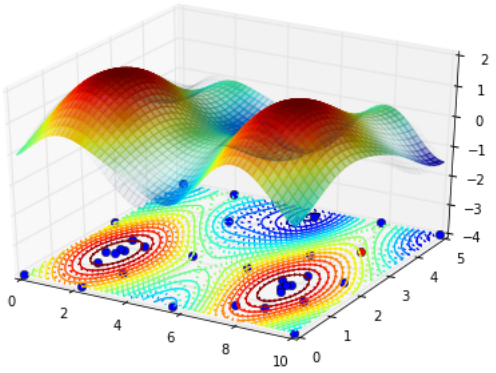
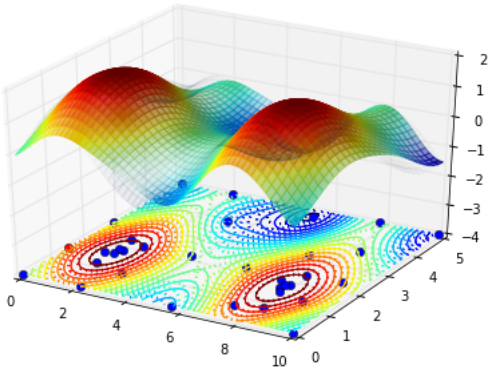
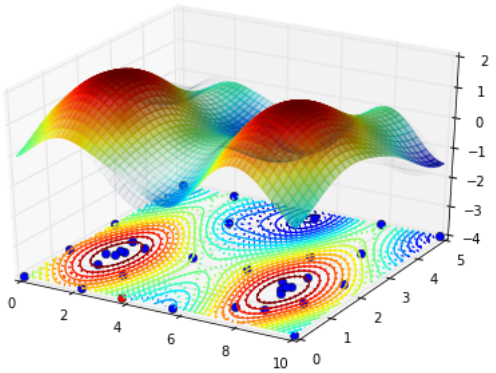
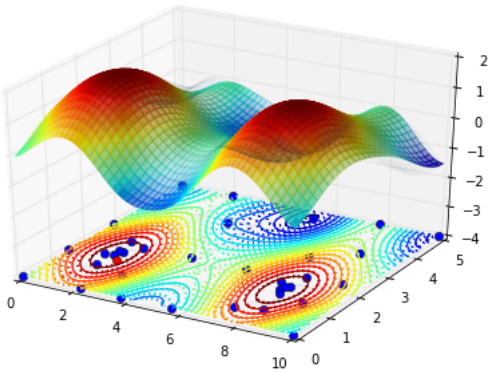
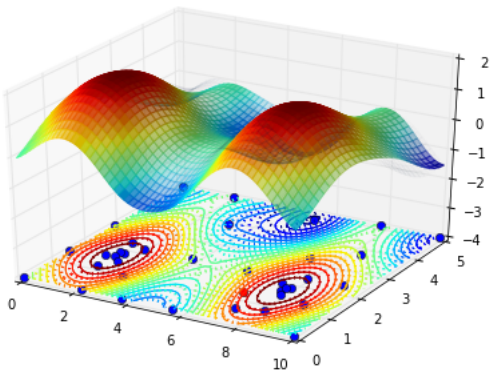
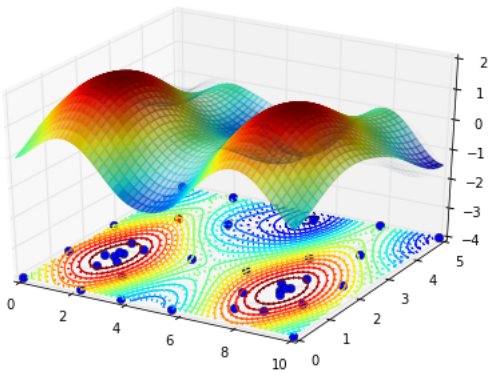
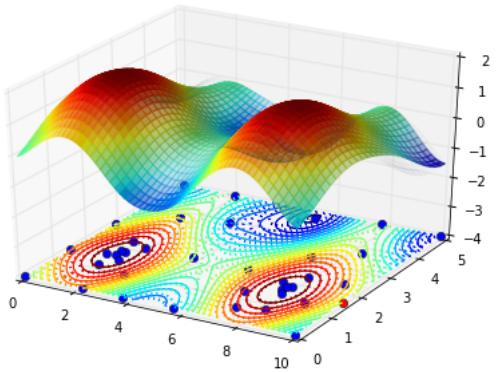
```
def sinsin(X):
    X = atleast_2d(X)
    return sin(X[:,0]) + sin(X[:,1])

gpgo_ei_2d(sinsin, [[0,0], [10,5]])
```



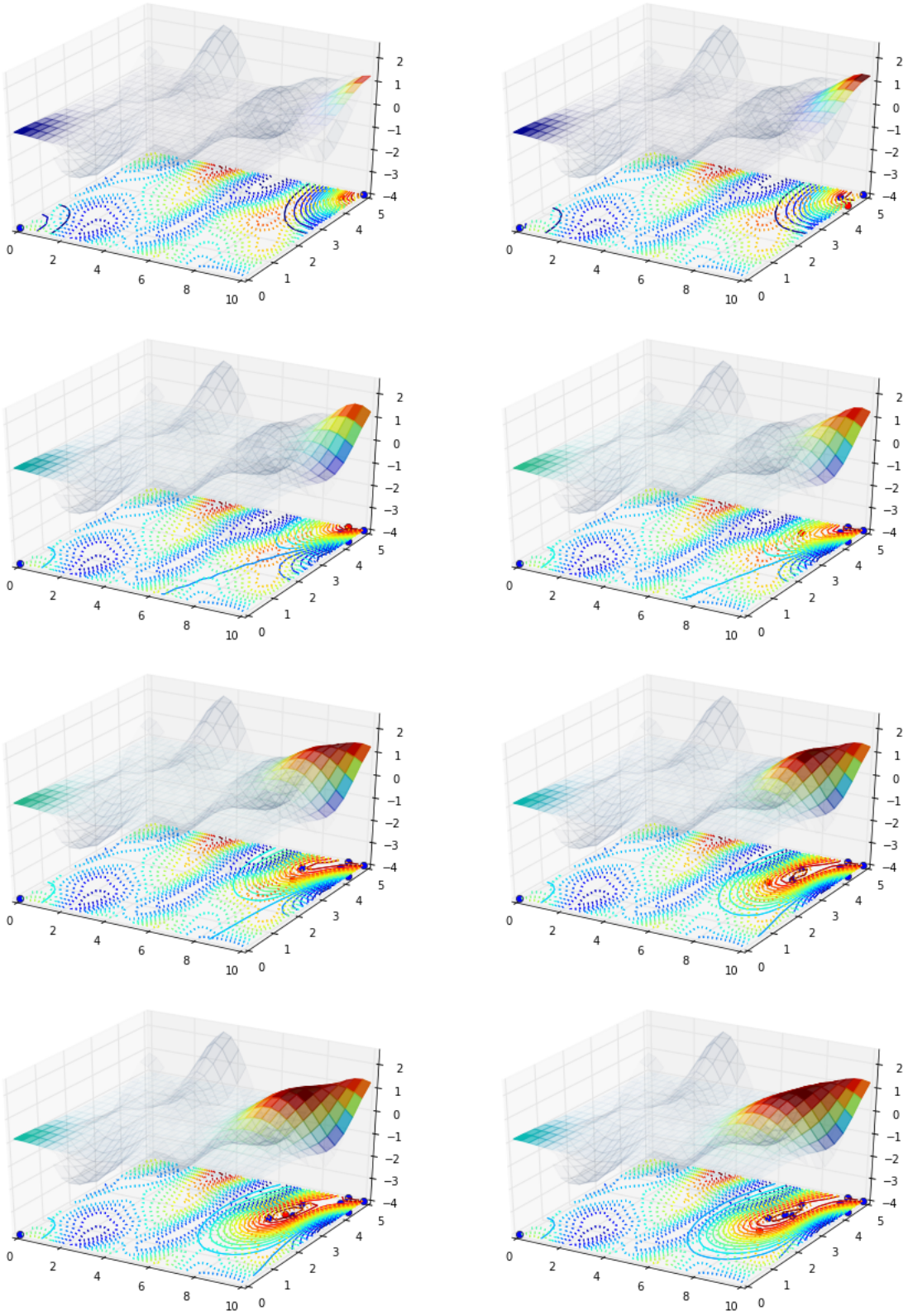


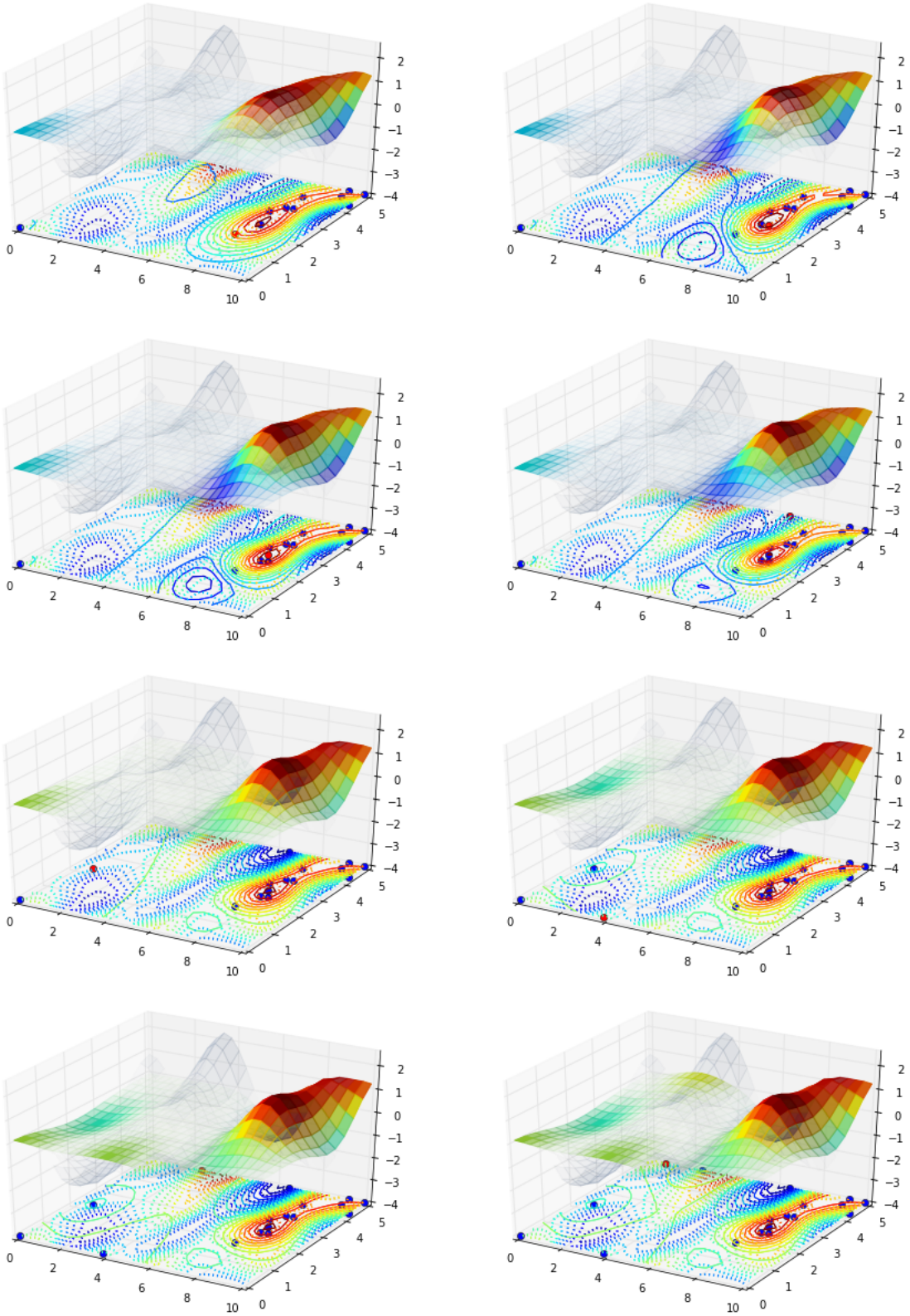


A.4.2.2 Objective function sampled from a GP

```
gpgo_ei_2d(GPmem(2), [[0,0], [10,5]], 30,15)
```




2
1
0
−1
−2
−3
−4
0
2
4
6
8
10
0
1
2
3
4
5