Acquisition Functions for Likelihood-Free Inference via Bayesian Optimization

Cole Zimmerman

4th Year Project Report
Artificial Intelligence
School of Informatics
University of Edinburgh
2018

Abstract

This report deals with the likelihood-free Bayesian inference of simulator-based statistical model parameters given some observed data. Our goal is to calculate the posterior distribution over parameter settings when direct evaluation of parameter likelihoods is impossible; thus, inference can only be done by assuming the likelihood is high when the 'discrepancy' between observed and simulated data is low. We assume the cost of running the simulator is high, so we want to find these low-discrepancy parameter settings in as few simulations as possible.

In this report, we explore an inference method that models the 'discrepancy function' (mapping parameters to discrepancies) using Gaussian processes (GPs). These models can be used to efficiently construct likelihood approximations when combined with Bayesian optimization, a tool which uses an *acquisition function* to iteratively choose new parameters to simulate and update the GP model with. We are also interested in *portfolios of acquisition functions*, higher-level strategies which choose the next parameter setting from a list of candidates each nominated by one standard acquisition function. The main benefit of portfolios is that, over time, they learn which acquisitions are best and shift their strategy to pick those well-performing acquisitions with high probability. Our goal is to determine which acquisitions and portfolios achieve the most accurate posterior approximation in the fewest simulations.

We ran hundreds of experiments comparing each acquisition and portfolio's performance, and found that the lower confidence bound (LCB) criterion was on average the best-performing standard acquisition. We did not find much success with any of our portfolio algorithms - on average, they were all outperformed by LCB, and often were unable to learn to pick the best acquisitions. Despite that, they did exhibit some desirable behaviour, and we believe that with some modifications they could rival or even outperform LCB in many situations.

Acknowledgements

I would like to thank my supervisor, Dr. Michael Gutmann, for guiding me through this process. I would also like to thank Marko Järvenpää for providing me with the code for the bacterial infections simulator.

Contents

1 Introduction

Contents

In the scientific world, simulator-based statistical models are commonly used tools for modeling complex processes. In practice, they are computer programs which take some parameters $\boldsymbol{\theta}$ and run simulations of some real-life process, producing simulated data $\boldsymbol{y_\theta}$ as output. They are especially appealing because they can be made as complex as necessary; no simplifications of the model have to be made to make the simulator analytically tractable. The downside to this, however, is that inference of the parameters $\boldsymbol{\theta}$ given observed data is made difficult by the lack of a tractable likelihood function $L(\boldsymbol{\theta}) = p(\boldsymbol{y}_o|\boldsymbol{\theta})$ (where $\boldsymbol{y}_o$ is the observed data). In this report, we explore in detail one Bayesian optimization-based method for efficient *likelihood-free* inference of $\boldsymbol{\theta}$. In the next section, we give a broad overview of our goals in exploring this method.

1.1 Project Goals

We are concerned with the Bayesian inference of model parameters $\boldsymbol{\theta}$ given observed data $\boldsymbol{y}_o$:

$$p(\boldsymbol{\theta}|\boldsymbol{y}_o) = \frac{p(\boldsymbol{y}_o|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\boldsymbol{y}_o)} \propto p(\boldsymbol{y}_o|\boldsymbol{\theta})p(\boldsymbol{\theta}). \tag{1}$$

We assume the use of a uniform prior, so the posterior probability of $\boldsymbol{\theta}$ is directly proportional to the likelihood of $\boldsymbol{\theta}$:

$$p(\boldsymbol{\theta}|\boldsymbol{y}_o) \propto p(\boldsymbol{y}_o|\boldsymbol{\theta}). \tag{2}$$

The specific models we are interested in are called *simulator-based statistical models*, a type of model which stochastically generates data $\boldsymbol{y_\theta}$ given parameter settings $\boldsymbol{\theta}$, where $\boldsymbol{y_\theta}$ is drawn from some latent probability density function $p_{\boldsymbol{y}|\boldsymbol{\theta}}$. Because $p_{\boldsymbol{y}|\boldsymbol{\theta}}$ is unknown, direct point-wise evaluation of the likelihood $L(\boldsymbol{\theta}) = p(\boldsymbol{y}_o|\boldsymbol{\theta})$ is not possible. Thus, an approximation of the posterior in Equation 2 must be created in order for any inference to occur. Inference methods that work without access to the true likelihood belong to a broad class of approximate inference methods referred to as *likelihood-free inference* (LFI).

LFI with the Discrepancy Function With the help of a function $D(\boldsymbol{y_\theta}, \boldsymbol{y}_o)$, which calculates the 'discrepancy' $\Delta_{\boldsymbol{\theta}}$ between observed and simulated data, likelihood-free inference of $\boldsymbol{\theta}$ can be done by assuming $p(\boldsymbol{y}_o|\boldsymbol{\theta})$ is high when $\Delta_{\boldsymbol{\theta}}$ is low. Several well-established LFI methods based on this assumption exist already, including the method of approximate Bayesian computation (ABC). Unfortunately, the simulators can be very computationally expensive to run, and these methods often require a large number of simulations to produce an accurate posterior approximation.

LFI via Bayesian Optimization In this report, we use a method proposed by Gutmann & Corander (2016) for efficient LFI via Bayesian optimization (BO) and Gaussian process regression. In a nutshell, it works by using a probabilistic model to repeatedly guess which $\boldsymbol{\theta}$ are likely to yield a low discrepancy, run the simulator at a chosen point $\boldsymbol{\theta}$ to sample a $\boldsymbol{y_\theta}$, calculate a corresponding discrepancy $\Delta_{\boldsymbol{\theta}}$, then update the model with the new data $(\boldsymbol{\theta}, \Delta_{\boldsymbol{\theta}})$ using Bayes' rule. As more data is acquired, the model becomes more accurate and better at estimating which $\boldsymbol{\theta}$ are likely to yield a low discrepancy (and thus have a high likelihood). After some number of iterations, the model should be accurate enough to be the basis for an approximation of the likelihood in Equation 2. Due to the expensiveness of running the simulator, we would like to construct a model sufficient for this purpose in as few iterations as possible.

Goals The main goal of this project is to assess which 'acquisition functions' (informally, the heuristics used to guess whether a $\boldsymbol{\theta}$ is likely to yield a low discrepancy) are best for the task of likelihood-free inference. In addition, we would like to assess the performance of so-called 'portfolios of acquisition functions', a higher-level type of BO algorithm which chooses the next $\boldsymbol{\theta}$ to evaluate from a list of candidate points each nominated by a standard acquisition functions. Over time, the portfolio strategy should learn to prioritize acquisitions that yield good points (i.e. low discrepancy/high information) over those who nominate bad points.

More specifically, the main goals of this project are to empirically demonstrate which acquisition functions and portfolios achieve better likelihoods in fewer iterations. Also, where possible, we attempt to characterize their behaviours with a mixture of qualitative and quantitative analysis.

1.2 Project Outline

Chapter 2 gives a broad overview of the background needed to understand our work. It is split into 5 sections:

- Section 2.1 explains simulator-based statistical models in greater depth, and gives an example of one such model.
- Section 2.2 outlines the basics of likelihood-free inference for simulator-based statistical models, approximate Bayesian computation, and kernel density estimation (KDE).
- Section 2.3 outlines the basics of Gaussian process regression (GPR), its use as a probabilistic model of the discrepancy, and how such a model can be used to construct an approximate likelihood via KDE.
- Section 2.4 gives an overview of Bayesian optimization with GPR, including descriptions of acquisition functions and key concepts such as the exploration/exploitation dilemma.
- Section 2.5 gives an introduction to portfolios of acquisition functions, a special family of BO algorithms meant to make data acquisition more efficient. It also delineates the four portfolio algorithms we use during our experiments (GP-Hedge, GP-Exp3, GP-Explorer, and Baseline).

Chapter 3 outlines the research questions we wish to answer, and Chapter 4 the experimental methodology we use to answer them.

Finally, Chapter 5 details the results of these experiments, while Chapter 6 offers some discussion of the results and ideas for future related work.

1.3 Contributions

- Wrote a moderately large Python library implementing everything required for our experiments (see 4.4), including the Bayesian optimization procedure, Gaussian processes, acquisition functions, portfolios of acquisition functions, kernel density estimation, and a simple program for aggregating and viewing all experiment results.
- Implemented and modified two existing portfolio algorithms, GP-Hedge and Exp3, for use in our experiments (see 2.5).
- Created two portfolio algorithms of our own, GP-Explorer and 'LCBEI', although we only had time to run experiments with GP-Explorer. See Sections 2.5.2.3 and 6.2 for their respective descriptions.
- Ran a large number of experiments comparing the LFI performance of 5 standard acquisition functions (see 5.1).
- Ran a large number of experiments comparing the LFI performance of 4 portfolio algorithms and the standard acquisition functions (see 5.2).
- Analyzed the behaviours of the standard acquisitions and the portfolios, and made some suggestions for improving the LFI performance of the portfolios (see 5.3).

2 Background

Contents

2.1 Simulator-Based Statistical Models

Simulator-based statistical models are stochastic generative models of real-life processes used often for scientific purposes. In this report, we consider those that take real-valued parameters and are stochastic in nature, i.e. repeatedly running a simulation with the same parameters will yield varying results. We denote a parameter setting as a vector $\boldsymbol{\theta} \in \mathbb{R}^d$ (where $d \in \mathbb{Z}$ is the number of parameters the simulator takes), and the data generated with those parameters as $\boldsymbol{y}_{\boldsymbol{\theta}}$.

Mathematically, a simulator-based statistical model is a family of latent probability density functions (pdfs), plus a function of $\boldsymbol{\theta}$ which samples data from the corresponding pdf:

$$S(\boldsymbol{\theta}) = \boldsymbol{y}_{\boldsymbol{\theta}}, \qquad \boldsymbol{y}_{\boldsymbol{\theta}} \sim \{p_{\boldsymbol{y}|\boldsymbol{\theta}}\}_{\boldsymbol{\theta}}. \tag{3}$$

We consider the task of inferring the parameters $\boldsymbol{\theta}$ of these simulators using observed data $\boldsymbol{y}_o$ and Bayes' rule (Equations 1 and 2).

Simulator-based statistical models are powerful tools for scientists who wish to model arbitrarily complex systems, as no compromises have to be made to make the simulators analytically tractable. The black-box nature of the simulators, however, comes with its own set of issues: namely, the lack of an analytical likelihood function $L(\boldsymbol{\theta}) = p_{\boldsymbol{y}|\boldsymbol{\theta}}(\boldsymbol{y}_o|\boldsymbol{\theta})$. For many simulators, it is impossible or impractical to even calculate the likelihood of a single point $\boldsymbol{\theta}$, making direct inference of $\boldsymbol{\theta}$ all but impossible. Therefore, inference of simulator parameters must be done via a class of approximate inference methods called *likelihood-free inference* (LFI). In Section 2.2, we give details of LFI for simulator-based statistical models.

2.1.1 An Example: Bacterial Infections Simulator

One example of a simulator-based statistical model was developed by Numminen et al. (2013) for modeling the spread of bacterial infections in daycare centers. Using a latent Markov chain, this simulator models the spread over time of a number of disease strains among individuals in day care centers. The variables (or states) of the Markov chain are binary variables I_{is}^t equaling 1 if individual i is infected with strain s at time t, and 0 otherwise. Three parametrized equations control the probabilities of each individual becoming infected with (or cured of) a new strain after some time h passes:

$$\begin{aligned} P(I_{is}^{t+h} = 0 | I_{is}^t = 1) &= h + o(h), \\ P(I_{is}^{t+h}) = 1 | I_{is'}^t = 0 \; \forall s') &= R_s(t)h + o(h), \\ P(I_{is}^{t+h} | I_{is}^t = 0, \exists I_{is'}^t = 1) &= \theta R_s(t)h + o(h). \end{aligned}$$

The rate of infection $R_s(t)$ factors in the probability an infection is introduced from outside the day care center (P_s), and the probability an infection is introduced from within the center ($E_s(t)$):

$$R_s(t) = \beta E_s(t) + \Lambda P_s.$$

The model parameters $\boldsymbol{\theta} = (\beta \in [0, 11], \Lambda \in [0, 2], \theta \in [0, 1])$ are called the internal infection parameter, the external infection parameter, and the co-infection parameter, respectively.

The data simulated by this model is an $S \times I$ matrix $\boldsymbol{y}_\theta$, where S and I are the number of strains and individuals, respectively, and the entry (s, i) is the variable I_{is}^t indicating whether individual i was infected with strain s at time t. The observed data $\boldsymbol{y}_o$ for this model are 29 such matrices sampled from participants at 29 day care centers.

Inference of this models parameters $\boldsymbol{\theta}$ is difficult as the observed data are snapshots of the states at just a single time point. In addition, "since the [data-generating] process evolves in continuous-time, the modeled system involves infinitely many correlated unobserved variables" (Gutmann & Corander, 2016), making direct evaluation of a true analytical likelihood completely impractical. Well-established LFI tools like Markov chain Monte Carlo (MCMC) methods struggle to deal with this kind of complex data (Green et al., 2015), especially when the cost of sampling from it (in our case, running the simulator) is very high.

2.2 Likelihood-Free Inference for Simulator-Based Statistical Models

Without a method for directly calculating the likelihood $L(\boldsymbol{\theta})$, we must find a way to calculate an approximation, $\mathcal{L}(\boldsymbol{\theta})$. To do this, we first need to formalize how we quantify the similarity between observed data $\boldsymbol{y_o}$ and simulated data $\boldsymbol{y_\theta}$. We do this with a carefully chosen discrepancy function

$$D(\boldsymbol{y_o}, \boldsymbol{y_\theta}) : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}_{\geqslant 0}, \tag{4}$$

which takes the observed and simulated data for parameter setting $\boldsymbol{\theta}$, and outputs a non-negative scalar "discrepancy" between the two: $\Delta_{\boldsymbol{\theta}}$. This is normally done by reducing the observed and simulated data to some choice of summary statistics, then calculating the distance between the reduced forms of the data. Reducing the data this way "helps to reduce dimensionality and filter out information deemed not relevant for the inference of $\boldsymbol{\theta}$" (Gutmann & Corander, 2016). The choice of discrepancy function and summary statistics is an ongoing area of research (Nunes & Balding, 2010) (Fearnhead & Prangle, 2012), but in this paper, we assume a suitable discrepancy function and choice of summary statistics are given in all cases.

For the sake of conciseness, we define a function f, which we simply call the *objective function*, as follows:

$$f(\boldsymbol{\theta}) = D(S(\boldsymbol{\theta}), \boldsymbol{y_o}). \tag{5}$$

As we assume suitable summary statistics and discrepancy function are given, we can ignore the functions S and D during our experiments, and instead assume we are directly given this objective f (see Section 4.1 for details). Moving forward, this objective function will be the basis for all likelihood approximations (and thus the basis for all inference of $\boldsymbol{\theta}$) in this paper.

Importantly, as the output of $S(\boldsymbol{\theta})$ is a random variable, $\Delta_{\boldsymbol{\theta}}$ is also a random variable:

$$\Delta_{\boldsymbol{\theta}} = f(\boldsymbol{\theta}), \qquad \Delta_{\boldsymbol{\theta}} \sim p_{\Delta|\boldsymbol{\theta}}, \tag{6}$$

where $p_{\Delta|\boldsymbol{\theta}}$ is the marginal distribution of f at $\boldsymbol{\theta}$, or in other terms, the distribution of discrepancies $\Delta_{\boldsymbol{\theta}}$ yielded by parameter settings $\boldsymbol{\theta}$. The form of $p_{\Delta|\boldsymbol{\theta}}$ is implicitly defined by S and the choice of D, and it is also of unknown analytical form, a property it inherits from $p_{\boldsymbol{y}|\boldsymbol{\theta}}$.

2.2.1 Approximate Bayesian Computation

This section offers a brief overview of approximate Bayesian computation (ABC) for the sake of contrasting it with the method used in our experiments. For a more thorough explanation of ABC (including some examples of its use on simulation-based models), see Turner & Zandt (2012).

Approximate Bayesian computation is a well-established method for approximately sampling from a posterior distribution when the likelihood is unavailable. The simplest form of it is the ABC rejection algorithm, which skips calculating an approximate likelihood and instead attempts to directly sample from the posterior. In this method, a set of points $\boldsymbol{\theta}^{(1)}, ..., \boldsymbol{\theta}^{(n)}$ is sampled from the prior distribution $p(\boldsymbol{\theta})$, then used to simulate data points $\boldsymbol{y}_{\boldsymbol{\theta}}^{(1)}, ..., \boldsymbol{y}_{\boldsymbol{\theta}}^{(n)}$. Each of these points are accepted if their corresponding discrepancy is less than some acceptance/rejection threshold ϵ and discarded otherwise: $f(\boldsymbol{\theta}) \leqslant \epsilon$. The result is a set of points which approximately follow the desired posterior distribution.

This approach is simple and makes few assumptions about the data, but nonetheless has some important drawbacks. The threshold ϵ is typically set to a low value to ensure the sampled $\boldsymbol{\theta}$ produce realistic data. With a low value for ϵ, the majority of $\boldsymbol{\theta}$ sampled from $p(\boldsymbol{\theta})$ are going to be rejected. Thus, a *very* high number of samples is needed before

enough are accepted to produce an accurate approximate posterior, and the majority of those samples are going to be thrown away. Therefore when the cost of running the simulator is high, posterior approximation by ABC can often require a prohibitively large amount of computational resources.

2.2.2 Likelihood Approximation by Kernel Density Estimation

In this report, the likelihood is approximated using kernel density estimation (KDE) (Parzen (1962), Rosenblatt (1956)), a nonparametric technique for estimating the probability distribution of a random variable using a finite number of realizations of it. While there exists a wide variety of choices for the kernel, we exclusively use the uniform (or boxcar) kernel:

$$L(\boldsymbol{\theta}) \approx \mathcal{L}(\boldsymbol{\theta}) = cP(\Delta_{\boldsymbol{\theta}} < h), \tag{7}$$

where c is a normalizing constant, h is the kernel bandwidth, which (similarly to ABC) acts as an acceptance/rejection threshold. The choice of h is crucial for approximating the likelihood - two different values can produce significantly different results. See, for instance, figure 1[1]. Letting h equal 2 produces a tight unimodal symmetric likelihood, while letting h equal 5 produces a wide multimodal asymmetric likelihood.

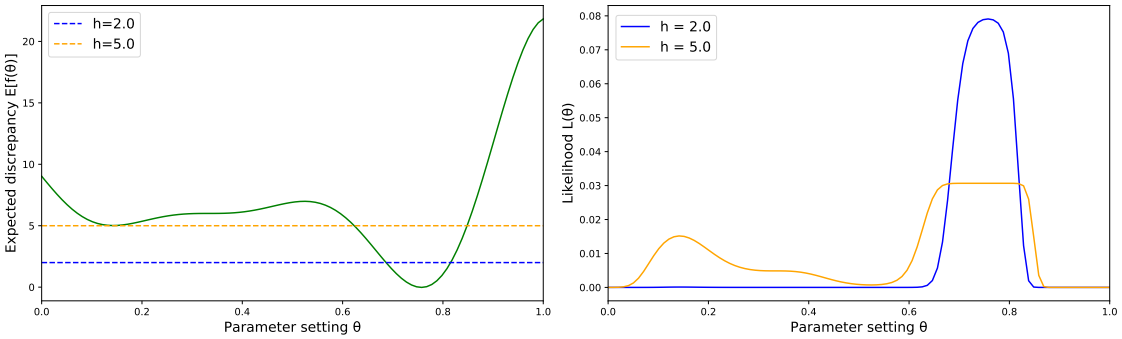


Figure 1: On the left, an example one-dimensional objective function with range (0, 22). On the right, two corresponding likelihood approximations: one calculated with a bandwidth of 2, the other with a bandwidth of 5.

There is no clear method for determining which bandwidth is better, so in our experiments, we simply set it equal to a sensibly small value, usually:

$$h = \Delta_{\boldsymbol{\theta}}^{min} + 0.05 \cdot (\Delta_{\boldsymbol{\theta}}^{max} - \Delta_{\boldsymbol{\theta}}^{min}), \tag{8}$$

where $\Delta_{\boldsymbol{\theta}}^{min}$ is the smallest observed discrepancy (obtained by sampling the objective f), and $\Delta_{\boldsymbol{\theta}}^{max}$ the largest. Setting the bandwidth to a small quantile of the observed data is a common convention in rejection ABC, so we felt it made sense to follow it. In addition, the small variations in bandwidths calculated this way can help to (very slightly) marginalize away the choice of bandwidth in calculating the likelihood.

As $p_{\Delta|\boldsymbol{\theta}}$ is of unknown analytical form, we are unable to directly calculate $P(\Delta_{\boldsymbol{\theta}} < h)$. Thus, we need to find a further approximation of $\mathcal{L}$, $\hat{\mathcal{L}}$, by estimating $P(\Delta_{\boldsymbol{\theta}} < h)$. Once we do that, we finally have a tractable approximation, $\hat{\mathcal{L}}$, of the true likelihood L.

[1]These likelihood approximations were calculated in a grid of 100 parameter values $\{0.01, 0.02, ..., 1.0\}$, using the assumption that for all $\boldsymbol{\theta}$, $f(\boldsymbol{\theta}) = \Delta_{\boldsymbol{\theta}} \sim \mathcal{N}(E[f(\boldsymbol{\theta})], 1.0)$.

2.2.3 Sample-Average Based Methods

Perhaps the simplest way to approximate $P(\Delta_{\boldsymbol{\theta}} < h)$ is by the use of sample averages: for each $\boldsymbol{\theta}$ you wish to calculate the likelihood of, sample N discrepancies $\Delta_{\boldsymbol{\theta}}^{(1)}, ..., \Delta_{\boldsymbol{\theta}}^{(N)}$ and let

$$P(\Delta_{\boldsymbol{\theta}} < h) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}[\Delta_{\boldsymbol{\theta}}^{(i)} < h],$$

where $\mathbb{I}$ equals one if $\Delta_{\boldsymbol{\theta}} < h$, and zero otherwise.

This method is extremely simple, and makes little to no assumptions about the data. On the other hand, it suffers from a number of disadvantages listed below (as presented in Gutmann & Corander (2016), Section 4). Choosing N forces the user to make the trade-off between high accuracy (large N) and computational efficiency (low N), and for finite N, the likelihood approximation $\hat{\mathcal{L}}$ is not necessarily smooth. Most importantly, sample average based methods are terribly inefficient when the simulator is expensive to run, requiring N simulations for every $\boldsymbol{\theta}$ to be evaluated. Regions of the parameter space ($\boldsymbol{\theta}$-space) that consistently yield high discrepancies, for example, should require far fewer simulations before realizing the likelihood there is almost certainly negligibly small (i.e. in that region, it is very unlikely any $\Delta_{\boldsymbol{\theta}} < h$).

2.3 Modeling the Discrepancy with Gaussian Process Regression

The following section details our motivations for using Gaussian processes (GPs), plus a brief introduction to how they work. For a more thorough overview, see Rasmussen & Williams (2006).

Various methods combining probabilistic modeling and optimization have been proposed to increase the computational efficiency of likelihood-free inference (e.g. Wilkinson (2014), Meeds & Welling (2014)). In this paper, we exclusively use the method proposed by Gutmann & Corander (2016) that combines Gaussian process regression with Bayesian optimisation.

To speed up inference, a few key goals must be met:

1. We need a framework for guessing which unexplored regions of $\boldsymbol{\theta}$-space are likely to yield small discrepancies, so that we may explore them more thoroughly.

2. This framework should also guess which regions of $\boldsymbol{\theta}$-space are likely to yield large discrepancies (and thus negligibly small likelihoods), so that we may ignore them in favor of more promising regions.

3. This framework should also be able to quantify its uncertainty of this knowledge, and act accordingly (e.g. "this region of $\boldsymbol{\theta}$-space doesn't look that promising, but the uncertainty there is high enough that it should be explored at least a little").

To achieve these ends, we create a probabilistic model $\hat{f}(\boldsymbol{\theta})$ of the objective function f using Gaussian process regression. By creating such a model $\hat{f}$ that can accurately emulate the distributions $f(\boldsymbol{\theta}) = \Delta_{\boldsymbol{\theta}} \sim p_{\Delta|\boldsymbol{\theta}}$, we can achieve these three goals and significantly speed up inference.

Building a regression function that directly models $L(\boldsymbol{\theta})$ is also possible; however, this requires choosing a bandwidth h before much data is gathered, which is difficult to do properly unless the range of possible discrepancies is known beforehand. For example, a small value for a simulator whose discrepancies are in the range $(0, 1000)$ is much different to that of a simulator with a discrepancy range of $(0, 10)$.

Building $\hat{f}$ requires training data in the form of tuples $(\boldsymbol{\theta}_t, \Delta_{\boldsymbol{\theta},t})$ obtained by sampling the objective f. We leave the issue of how this training data is gathered to the next section, assuming here we have a suitable evidence set, $\mathcal{E}_t = \{(\boldsymbol{\theta}_1, \Delta_{\boldsymbol{\theta},1}), ..., (\boldsymbol{\theta}_t, \Delta_{\boldsymbol{\theta},t})\}$ for use in constructing the model.

2.3.1 Assumptions

To construct $\hat{f}$ using Gaussian processes, we must make a few reasonable assumptions about f:

- f is a smooth function, i.e. sampling it at any point yields information about nearby points.

- Each distribution over discrepancies given a parameter settings $p_{\Delta|\boldsymbol{\theta}}$ follows a normal distribution.

- The joint distribution of f at any finite number of points $\{\boldsymbol{\theta}_1, ..., \boldsymbol{\theta}_t\}$ is Gaussian with some mean vector $\boldsymbol{m_t}$ and covariance matrix $\boldsymbol{K}_t$:

$$\boldsymbol{\Delta}_{\boldsymbol{\theta}}^{(1:t)} = (\Delta_{\boldsymbol{\theta},1}, ..., \Delta_{\boldsymbol{\theta},t}) \sim \mathcal{N}(\boldsymbol{\Delta}_{\boldsymbol{\theta}}; \boldsymbol{m_t}, \boldsymbol{K}_t). \tag{9}$$

Strictly speaking, these assumptions do not always perfectly hold. Real discrepancies are non-negative, and there is no easy way to formulate GPs to put positive probability only on non-negative $\Delta_{\boldsymbol{\theta}}$. Modeling the log discrepancy, $f(\boldsymbol{\theta}) = \log \Delta_{\boldsymbol{\theta}}$, (as proposed by Gutmann & Corander (2016)) instead can help alleviate this issue. In our experiments, however, we directly model the discrepancy, as Gutmann et al. found it works just as well as modeling $\log \Delta_{\boldsymbol{\theta}}$.

2.3.2 Gaussian Process Basics

A Gaussian process is specified by a prior mean function $m(\boldsymbol{\theta})$ and a covariance function $k(\boldsymbol{\theta}, \boldsymbol{\theta}^{'})$ subject to additive Gaussian observation noise with variance σ_n^2. Intuitively, the prior mean function takes a parameter value $\boldsymbol{\theta}$ and "guesses" the mean discrepancy it will yield, $E[\Delta_{\boldsymbol{\theta}}]$. It is often chosen to be zero, some other constant, or a sum of quadratic polynomials:

$$m_t(\boldsymbol{\theta}) = \sum_j a_j \theta_j^2 + b_j \theta_j + c, \qquad \boldsymbol{m_t} = (m_t(\boldsymbol{\theta}_1, ..., m_t(\boldsymbol{\theta}_t))^T \tag{10}$$

For our experiments, we use a Gaussian process with a mean function that returns the average over all observed discrepancies:

$$m_t(\boldsymbol{\theta}) = \frac{1}{t} \sum_{i=1}^{t} \Delta_{\boldsymbol{\theta},i}. \tag{11}$$

We discuss why we chose this prior mean over a constant zero-mean function in Section 2.4.2.

The covariance function, or *kernel*, is a 'guess' at how correlated two (normally distributed) discrepancies $\Delta_{\boldsymbol{\theta}}$ and $\Delta_{\boldsymbol{\theta}}^{'}$ are. The squared exponential covariance function is a common choice (and what we use for our experiments):

$$k(\boldsymbol{\theta}, \boldsymbol{\theta}^{'}) = \sigma_f^2 \exp\Big(-\sum_j \frac{1}{\lambda_j^2} (\theta_j - \theta_j^{'})^2 \Big), \qquad \boldsymbol{K}_t = \begin{pmatrix} k(\boldsymbol{\theta}_1, \boldsymbol{\theta}_1) & ... & k(\boldsymbol{\theta}_1, \boldsymbol{\theta}_t) \\ \vdots & \vdots & \vdots \\ k(\boldsymbol{\theta}_1, \boldsymbol{\theta}_1) & ... & k(\boldsymbol{\theta}_t, \boldsymbol{\theta}_t) \end{pmatrix} + \sigma_n^2 \boldsymbol{I}_t \tag{12}$$

This kernel makes the assumption that the closer two points $\boldsymbol{\theta}$ and $\boldsymbol{\theta}^{'}$ are, the more correlated their corresponding discrepancies are. Since the discrepancies are assumed to all be normally distributed with the same variance (σ_n^2), this is a reasonable further assumption to make. Section 2.3.4 gives an overview of the hyperparameters a_j, b_j, c, σ_n, σ_f, and λ_j.

With these equations, we can compute the posterior predictive distribution of f, a Gaussian with posterior mean $\mu_t(\boldsymbol{\theta})$ and posterior variance $\upsilon_t(\boldsymbol{\theta})$:

$$\Delta_{\boldsymbol{\theta}} | \mathcal{E}_t \sim \mathcal{N}(\Delta_{\boldsymbol{\theta}}; \mu_t(\boldsymbol{\theta}), \upsilon_t(\boldsymbol{\theta}) + \sigma_n^2), \tag{13}$$

The posterior mean $\mu_t(\boldsymbol{\theta})$ is the GP model's estimate of $f(\boldsymbol{\theta})$ given data $\mathcal{E}_t$. It can be used to predict discrepancies for unseen $\boldsymbol{\theta}$ and can be minimized efficiently with gradient-based techniques. The posterior variance $\upsilon_t(\boldsymbol{\theta})$ can be thought of as the level of uncertainty the model has about the distribution of discrepancies at $\boldsymbol{\theta}$:

$$\mu_t(\boldsymbol{\theta}) = m_t(\boldsymbol{\theta}) + k_t(\boldsymbol{\theta})^T \boldsymbol{K}_t^{-1} (\boldsymbol{\Delta}_{\boldsymbol{\theta}}^{(1:t)} - \boldsymbol{m}_t), \qquad \upsilon_t(\boldsymbol{\theta}) = k(\boldsymbol{\theta}, \boldsymbol{\theta}) - \boldsymbol{k_t}(\boldsymbol{\theta})^T \boldsymbol{K}_t^{-1} \boldsymbol{k}_t(\boldsymbol{\theta}), \tag{14}$$

$$\boldsymbol{\Delta}_{\boldsymbol{\theta}}^{(1:t)} = (\Delta_{\boldsymbol{\theta},1}, ..., \Delta_{\boldsymbol{\theta},t})^T, \qquad \boldsymbol{k}_t(\boldsymbol{\theta}) = (k(\boldsymbol{\theta}, \boldsymbol{\theta}_1), ..., k(\boldsymbol{\theta}, \boldsymbol{\theta}_t))^T.$$

As more data is gathered around a point $\boldsymbol{\theta}$, the subtracted term in $\upsilon_t(\boldsymbol{\theta})$ grows, causing the uncertainty to shrink. See Figure 2 below for an example. Notice how the error bars "pinch" inwards around the evidence, and the corresponding pdfs $p(\Delta_{\boldsymbol{\theta}} | \mathcal{E}_t, \theta)$ get tighter.

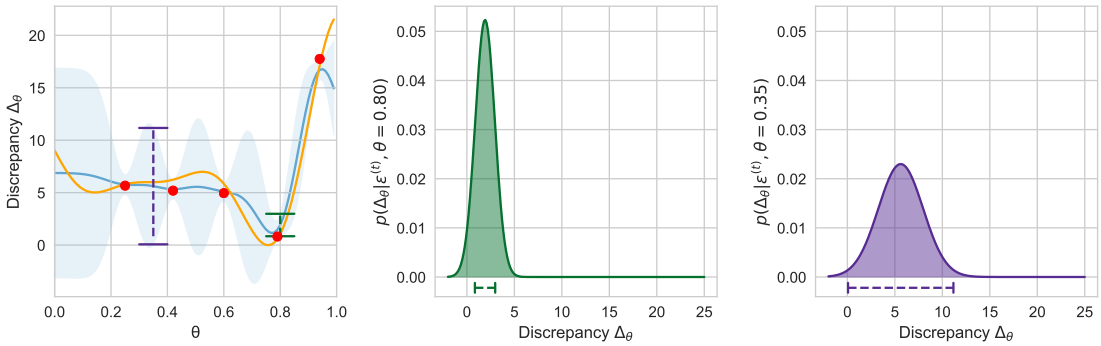


Figure 2: A Gaussian process model of an example 1D objective function f_{forr}. Evidence $(\theta, \Delta_{\boldsymbol{\theta}})$ is marked in red. Posterior means and variances are marked in blue. The two figures on the right show predictive pdfs $p(\Delta_{\boldsymbol{\theta}}|\mathcal{E}^{(t)}, \theta)$ for $\theta = 0.8$, 0.35.

2.3.3 Likelihood Estimation via Gaussian Processes

These discrepancy models can be used to estimate parameter likelihoods:

$$\hat{\mathcal{L}}(\boldsymbol{\theta}) = \Phi\Big(\frac{h - \mu_t(\boldsymbol{\theta})}{\sqrt{\upsilon_t(\boldsymbol{\theta}) + \sigma_n^2}}\Big) \tag{15}$$

$$\hat{p}(\boldsymbol{\theta}|\boldsymbol{y}_o) \propto \hat{\mathcal{L}}(\boldsymbol{\theta}) p(\boldsymbol{\theta}) \propto \hat{\mathcal{L}}(\boldsymbol{\theta}) \tag{16}$$

where $p(\boldsymbol{\theta})$ is the prior distribution over parameters (assumed to be uniform), and Φ is the standard normal cumulative density function (cdf). This approximate posterior can be compared to the true posterior with metrics like the Kullback-Leibler divergence (see Sections 4.1 and 4.2 for details).

Construction of the approximate likelihood using Gaussian processes is much more efficient than through using sample-averaged based methods or ABC rejection:

- Because μ_t and υ_t are smooth functions, the resulting likelihood will also be smooth.
- Because this likelihood is smooth, we can gain information about likelihoods of an entire region of $\boldsymbol{\theta}$-space by sampling the objective just one or two times in that region.
- Unlike ABC rejection, using Gaussian process modeling allows us to intelligently select which points $\boldsymbol{\theta}$ to use in evaluating the objective:
  - We can choose not to bother running simulations with $\boldsymbol{\theta}$ likely to yield high discrepancies (e.g. $\theta \approx 0.95$ in our example above).
  - We can choose to run simulations more frequently in regions of $\boldsymbol{\theta}$-space likely to yield a low discrepancy (e.g. $\theta \approx 0.79$ in our example above).

Importantly, using Gaussian processes to predict discrepancies for unseen $\boldsymbol{\theta}$ allows us to use a powerful tool called *Bayesian optimization*, which we go over in Section 2.4.

2.3.4 Gaussian Process Hyperparameters

The constants a_j, b_j, c, σ_n, σ_f, and λ_j shown in Equations 10 to 14 are hyperparameters. In this section, we briefly go over each (excluding those for the prior mean) and explain their role.

σ_n^2 is called the observation noise variance. It corresponds to the variance of $f(\boldsymbol{\theta}) = \Delta_{\boldsymbol{\theta}} \sim p_{\Delta|\boldsymbol{\theta}}$ (Equation 6). It is important to note that our Gaussian process model $\hat{f}$ assumes this observation noise is constant with respect to $\boldsymbol{\theta}$ (or 'homoscedastic', as opposed to 'heteroscedastic'), i.e. that $p_{\Delta|\boldsymbol{\theta}}$ has the same variance regardless of $\boldsymbol{\theta}$. This property can not always assumed to be true; for instance, the bacterial infections simulator (Section 2.1.1) has heteroscedastic observation noise. This assumption of homoscedasticity can be relaxed by using an 'input-dependent GP model' (Järvenpää et al., 2017a), but we do not explore this option in this report.

Formally, the 'signal variance' σ_f^2 is the marginal variance of $\hat{f}$ if the observation noise is zero. Intuitively, it can be thought of as the maximum level of uncertainty the model can have about a point $\boldsymbol{\theta}$. To illustrate this, consider the posterior variance (Equation 14) of a point $\boldsymbol{\theta}$ when no data has been observed. The subtracted term will equal 0, and the first term $k(\boldsymbol{\theta}, \boldsymbol{\theta})$ will equal σ_f^2 (as the exponential term becomes 1), leaving $\upsilon_t(\boldsymbol{\theta}) = \sigma_f^2$.

Finally, the length scales λ_j control the amount of correlation between $f(\boldsymbol{\theta})$ and $f(\boldsymbol{\theta}')$. Setting them very low places high probability on very smooth functions, and setting them high on very 'wiggly' ones.

The hyperparameter settings of a Gaussian process can significantly affect data acquisition during Bayesian optimization (see Section 2.4.2) and so they must be set carefully. The hyperparameters $\boldsymbol{\phi}$ can be inferred by maximizing the log marginal likelihood of the observed discrepancies $\boldsymbol{\Delta}_{\boldsymbol{\theta}}$, or alternatively through cross-validation (Rasmussen & Williams (2006), 5.4.2):

$$\log p(\boldsymbol{\Delta}_{\boldsymbol{\theta}} \mid \mathcal{E}_t, \boldsymbol{\phi}) = -\frac{1}{2}\boldsymbol{\Delta}_{\boldsymbol{\theta}}^T \boldsymbol{K}_t^{-1} \boldsymbol{\Delta}_{\boldsymbol{\theta}} - \frac{1}{2}\log|\boldsymbol{K}_t| - \frac{t}{2}\log 2\pi. \tag{17}$$

In addition, the hyperparameters can be continually updated as more data is gathered (as in Gutmann & Corander (2016)), or they can be precomputed by gathering a large amount of data and maximizing one of these two equations offline. For our experiments, we infer the hyperparameters offline by maximizing L_{LOO}. In doing so, we make the assumption that the GP hyperparameters are fixed and known during Bayesian optimization. Discussion of the implications of this assumption can be found later on in Section 6.1.

2.4 Building the Model with Bayesian Optimization

This section offers an overview of Bayesian optimization sufficient enough to guide the reader through the rest of the paper. For a more thorough overview of Bayesian optimization, see any of (Jones (2001); Shahriari et al. (2016); Brochu et al. (2010); Snoek et al. (2012)).

Bayesian optimization (BO) is a strategy for minimizing black-box functions (functions with unknown form and gradients). It iteratively builds a probabilistic model of the objective function f with the ultimate goal of finding its minimizer in as few iterations as possible. At each iteration $t = 1, ..., t_{max}$, it selects a new point $\boldsymbol{\theta}_t$ using the model and some decision rule (called an 'acquisition function', or alternatively an acquisition *rule*). After, it samples the objective to obtain the new data point $(\boldsymbol{\theta}_t, \Delta_{\boldsymbol{\theta},t})$, updates the evidence set $\mathcal{E}^{(t)}$, and then updates the model via Bayes' rule. In order to find low-discrepancy $\boldsymbol{\theta}$ in as few objective evaluations as possible, a balance must be struck between *exploring* regions of $\boldsymbol{\theta}$-space that we have very little information on, and *exploiting* promising looking areas (i.e. repeatedly evaluating areas for which the model predicts low discrepancies).

Central to the process of Bayesian optimization is the acquisition function α, which is optimized to choose the next point $\boldsymbol{\theta}^{(t)}$ to evaluate. Much of the literature on Bayesian optimization is framed with the goal of maximizing the objective, and thus also maximizes the acquisition (e.g. Hoffman et al. (2011) and Shahriari et al. (2016)). In this paper, however, the goal is to minimize the objective (discrepancy), so we formulate the acquisitions such that they are minimized. Thus, we select the next point $\boldsymbol{\theta}^*$ to evaluate as the minimizer of α.

A common choice is the lower confidence bound (LCB) criterion:

$$\alpha_{LCB}(\boldsymbol{\theta}) = \mu_t(\boldsymbol{\theta}) - \sqrt{\eta_t^2 \upsilon_t(\boldsymbol{\theta})}, \tag{18}$$

where $\eta_t^2 = 2log[t^{d/2+2}\pi^2/(3\epsilon_\eta)]$ and $\epsilon_\eta = 0.1$. The exact form of the function and the choice of η_t may be tweaked to fit each individual problem; here, we present it exactly as in Gutmann & Corander (2016). The balance between exploration and exploitation is readily visible in the two terms of the equation: the first term, $\mu_t(\boldsymbol{\theta})$, incentivizes picking $\boldsymbol{\theta}$ the model believes will yield a low discrepancy. The second term incentivizes picking $\boldsymbol{\theta}$ for which the model is uncertain what discrepancy it will yield. η_t controls the balance between these two objectives. Here, it grows larger as t increases, meaning exploration is increasingly favored as more evidence is gathered. An example of the data acquisition process can be seen below in Figure 3.

This combination of Bayesian optimization and GPR has been empirically shown to work well for likelihood-free inference (Gutmann & Corander (2016); Järvenpää et al. (2017b); Järvenpää et al. (2017a)). By modeling f as a Gaussian process, often only a single evaluation of $f(\boldsymbol{\theta})$ is required to verify with high probability whether $\boldsymbol{\theta}$ will have a negligibly small likelihood. This way, more focus can be given to high-likelihood regions of $\boldsymbol{\theta}$-space during optimization, thus building an accurate model of the likelihood in fewer simulations.

2.4.1 Acquisition Functions

Many well-established acquisition functions for Bayesian optimization exist, each with their own strengths and weaknesses. Some, like LCB, tend to favor exploration compared to other acquisitions. Others, like probability of improvement (PI) and expected improvement (EI) focus more on exploitation. The equation for probability of improvement is:

$$\alpha_{PI}(\boldsymbol{\theta}) = -\mathcal{P}[\Delta_{\boldsymbol{\theta}} < \tau] = -\Phi\Big(\frac{\tau - \mu_t(\boldsymbol{\theta})}{\sqrt{\upsilon_t(\boldsymbol{\theta})}}\Big), \tag{19}$$

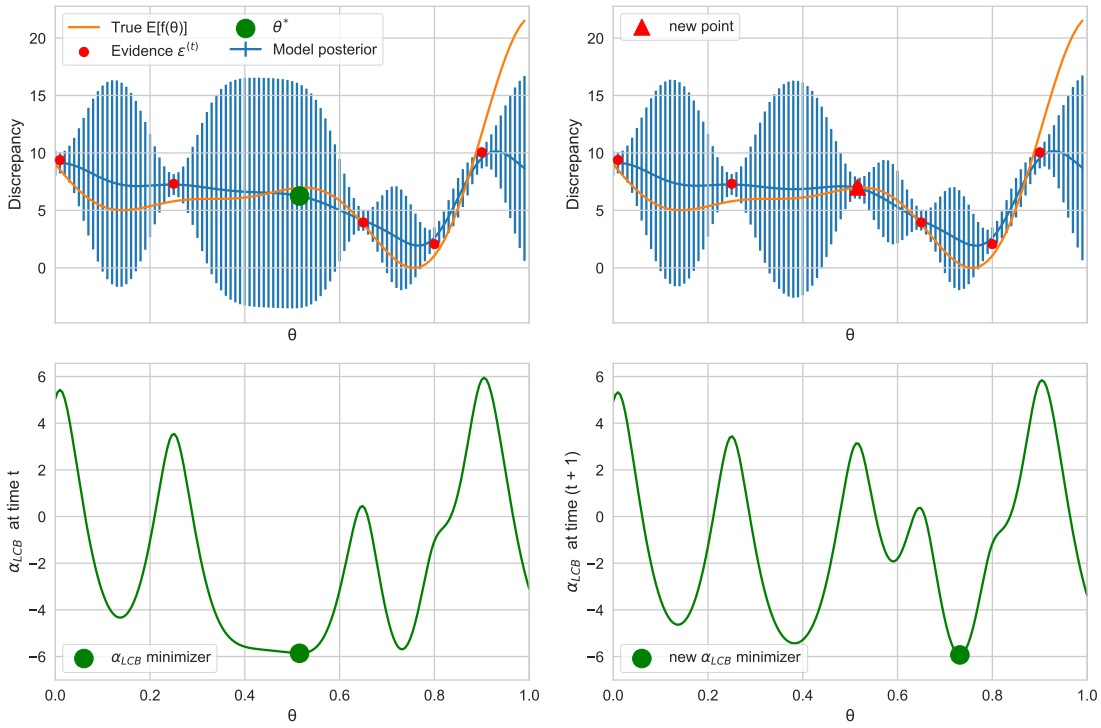


Figure 3: The top two figures show an example objective f_{forr} and a GP model being updated after acquiring a new data point. The bottom two figures the values of the LCB acquisition being used to select the new data point.

where Φ is the standard normal cumulative distribution function and the negative sign is added for minimization. The idea is simple: return the point the model believes is most likely to yield a lower discrepancy than some incumbent discrepancy. The incumbent is commonly set equal to the best observed outcome so far, $\tau = \min_{\Delta_{\boldsymbol{\theta},j} \in \boldsymbol{\Delta}_{\boldsymbol{\theta}}^{(1:t)}} \Delta_{\boldsymbol{\theta},j}$. Compared to other acquisitions, however, PI has been shown to be heavily exploitative (and thus ineffective) when the incumbent is set this way (Järvenpää et al. (2017b); Shahriari et al. (2016); Jones (2001)).

While PI considers only the probability of improving over τ, EI also factors in the expected *amount* of improvement under the model:

$$u(x) = \max(0, \tau - f(x)), \tag{20}$$

$$\alpha_{EI}(\boldsymbol{\theta}) = \mathbb{E}[u(\boldsymbol{\theta})|\boldsymbol{\theta}, \mathcal{E}_t], \tag{21}$$

$$= -(\tau - \mu_t(\boldsymbol{\theta}))\Phi\Big(\frac{\tau - \mu_t(\boldsymbol{\theta})}{\sqrt{v_t(\boldsymbol{\theta})}}\Big) - \sqrt{v_t(\boldsymbol{\theta})}\phi\Big(\frac{\tau - \mu_t(\boldsymbol{\theta})}{\sqrt{v_t(\boldsymbol{\theta})}}\Big), \tag{22}$$

where ϕ is the standard normal probability density function, and the minus signs here are added for minimization.

For our experiments, we also used three additional acquisition functions. The first, which we call 'Rand', selects the next point to evaluate uniformly at random (u.a.r.) from the set of possible parameter settings Θ:

$$\alpha_{Rand} : \boldsymbol{\theta}_t \underset{u.a.r.}{\sim} \Theta. \tag{23}$$

The second acquisition function is intentionally bad (thus the name), seeking out $\boldsymbol{\theta}$

with low uncertainty and high expected discrepancy:

$$\alpha_{Bad}(\boldsymbol{\theta}) = \upsilon_t(\boldsymbol{\theta}) - \mu_t(\boldsymbol{\theta}). \tag{24}$$

This acquisition function is included for the purposes of testing the ability of 'portfolios of acquisition functions' (Section 2.5) to 'weed out' bad acquisition functions.

The final acquisition, which we call 'PostVar', seeks out areas of high uncertainty, ignoring $\mu_t(\boldsymbol{\theta})$ altogether:

$$\alpha_{PostVar}(\boldsymbol{\theta}) = -\upsilon_t(\boldsymbol{\theta}). \tag{25}$$

2.4.2 How GP Hyperparameters Affect Data Acquisition

In Section 2.3, we gave our choice of prior mean function for our Gaussian process model:

$$m(\boldsymbol{\theta}) = \frac{1}{t}\sum_{i=1}^{t} \Delta_{\boldsymbol{\theta},i},$$

and said we used it over the constant zero-mean function.

The reason we chose this prior mean was that using the zero-mean function significantly changes data acquisition behaviour. By using a constant prior mean of zero, we are effectively saying our prior belief is that the discrepancy is zero *everywhere*. During optimization, this translates into a posterior belief that the discrepancy is zero everywhere that hasn't been explored yet. Figure 4 shows an example of this phenomenon.

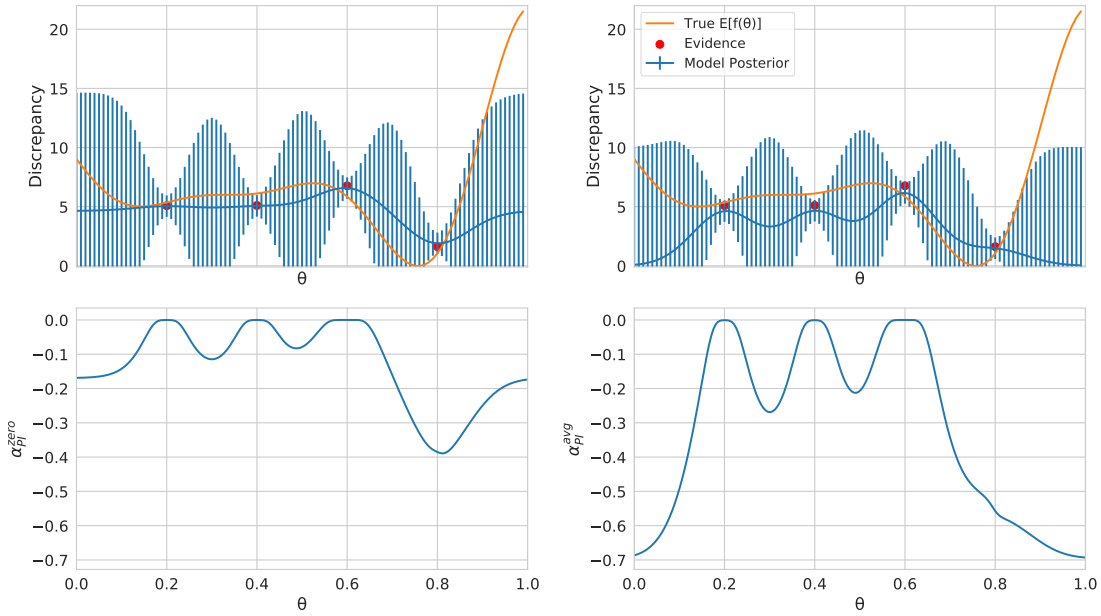


Figure 4: The top two plots show two GP models of f_{forr} (see 4.1): the left with an average-of-observed-discrepancies prior mean, and the right with a prior mean of zero. The bottom shows PI acquisition values for each of these models (which are minimized).

As this situation demonstrates, using a zero mean prior causes even the most exploitation-oriented acquisitions (e.g. α_{PI}) to prioritize exploration. This zero-prior phenomenon has been observed in our experiments and in Järvenpää et al. (2017b), where it was welcomed as an increased incentive for exploration. In this report, however, one of our main goals is to see if 'portfolios of acquisition functions' (2.5) can learn which acquisitions perform best

over time. For this to be possible, we need these acquisitions to have varying behaviours, not for them all to be continually picking unexplored regions of $\boldsymbol{\theta}$-space. Therefore, we use the average-based prior mean for all of our experiments.

2.5 Portfolios of Acquisition Functions

In practice, no single acquisition function is superior to others in all situations. In fact, it "has been empirically observed that the preferred strategy can change at various stages of the sequential optimization process" (Shahriari et al., 2016). To address this issue, Hoffman et al. (2011) proposed the use of a hierarchically-structured portfolio of acquisition functions. With this approach, a number of acquisition functions $k = 1, ..., K$ each nominate a point $\boldsymbol{\theta}_t^k$, and a higher-level strategy decides which of these K points to select. The base acquisitions 'nominate' points by minimizing their respective acquisition function (e.g. LCB would nominate the minimizer of Equation 18). The portfolio strategy keeps track of each acquisition's past performance and attempts to learn which acquisitions are likely to yield 'good' $\boldsymbol{\theta}$ (i.e. likely to yield a low discrepancy) and which are likely to nominate less useful $\boldsymbol{\theta}$.

Portfolio strategies solve a fundamentally different problem than standard acquisitions. Standard acquisitions do no learning and have no memory of past selected points. Portfolio strategies, on the other hand, are tasked with learning based on past data, and choose the next point from only K candidate points (as opposed to choosing from the entire continuous parameter space). In addition, the "data" portfolios use to learn (i.e. how well each acquisition is doing) evolves with time, and older data may become irrelevant to later stages of optimization. Thus, a balance must be struck between quick learning and filtering of older data no longer deemed to be relevant.

2.5.1 Overview

All of the portfolios we go over follow the same underlying mechanism: they track the cumulative gains (or regrets) g_k that each acquisition $k = 1, ..., K$ has yielded throughout the optimization process. At each iteration t, the portfolio has to choose between K candidate points $\{\boldsymbol{\theta}_t^1, ..., \boldsymbol{\theta}_t^K\}$, where $\boldsymbol{\theta}_t^k$ is the candidate nominated by acquisition k at iteration t.

To make this choice, the portfolio keeps a probability distribution over acquisitions, $p_t(k)$. This probability distribution is typically initialized uniformly. At each iteration of optimization, the portfolio:

1. Selects point $\boldsymbol{\theta}_t = \boldsymbol{\theta}_t^k$ with probability $p_t(k)$.
2. Samples the objective to get $(\boldsymbol{\theta}_t, \Delta_{\boldsymbol{\theta},t})$, and updates its GP model accordingly.
3. Calculates a *reward* r_t^k for each acquisition based on the candidate it nominated, and updates the gains $g_t^k = g_{t-1}^k + r_t^k$ accordingly.
4. Updates the probability distribution p_{t+1} based on the new rewards.

Each portfolio algorithm uses a different system for calculating rewards, all based on the usefulness of the candidate point. For instance, if the probability of improvement (PI) acquisition picks a very high-discrepancy point, the portfolio gives α_{PI} a low reward and lowers $p_{t+1}(\alpha_{PI})$.

Over many iterations, the portfolio will hopefully learn which acquisitions are likely to yield unhelpful points, and which are likely to yield useful (i.e. low-discrepancy and/or high information gain) points. We aim to characterize the typical behaviours of a few portfolio strategies (e.g. does it prefer exploration or exploitation; does it heavily prefer certain acquisitions over others?), and make an argument as to which fare better for the task of likelihood-free inference.

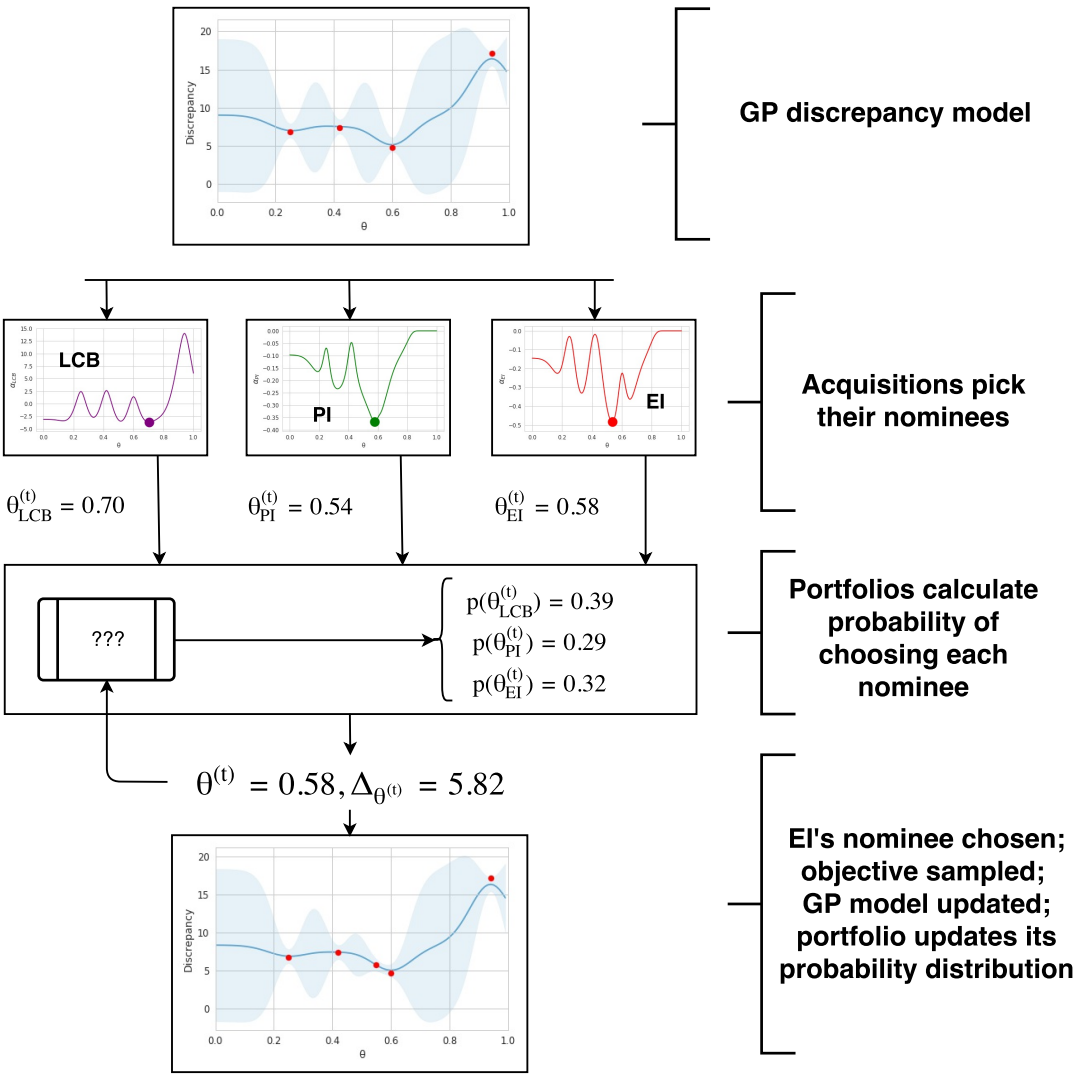


Figure 5: A diagram showing the basic structure of our portfolio algorithms, and example of it in use.

2.5.2 Portfolio Algorithms

Hoffman et al. (2011) adapted three existing algorithms for use as portfolio strategies: Hedge (Auer et al., 2000), Exp3 (Auer et al., 2000), and NormalHedge (Chaudhuri et al., 2009). In the next couple sections, we will go over the first two portfolio strategies as presented by Hoffman et al., then explain how we modified them for use in our experiments. We also go over one portfolio strategy we created specifically for the task of likelihood-free inference, *GP-Explorer*.

2.5.2.1 GP-Hedge The first portfolio strategy combines a Gaussian process model of the objective with the Hedge algorithm proposed by Auer et. al. The algorithm, which was designed for the goal of *maximizing* the objective, is given below:

At each time step t, each base acquisition k nominates a candidate point $\boldsymbol{\theta}_t^k$. Hedge then selects point $\boldsymbol{\theta}_t^k$ with probability $p_t(k)$, which is based on the cumulative rewards

Algorithm 1 GP-Hedge (for maximization)

1: Select parameter $\eta \in \mathbb{R}^+$.
2: Set $g_0^k = 0$ for $k = 1, ..., K$.
3: **for** t=1, 2, ... **do**
4: Nominate points $\boldsymbol{\theta}_t^k$ from each acquisition function.
5: Select nominee $\boldsymbol{\theta}_t = \boldsymbol{\theta}_t^k$ with probability $p_t(k) = \exp(\eta g_{t-1}^k) / \sum_{l=1}^K \exp(\eta g_{t-1}^l)$.
6: Sample the objective function $\Delta_{\boldsymbol{\theta},t} = f(\boldsymbol{\theta}_t)$.
7: Augment the data $\mathcal{E}_t = \{\mathcal{E}_{t-1}, (\boldsymbol{\theta}_t, \Delta_{\boldsymbol{\theta},t})\}$ and update the GP model accordingly.
8: Receive rewards $r_t^k = \mu_t(\boldsymbol{\theta}_t^k)$ from the updated GP. $\{r_t^k = -\mu_t(\boldsymbol{\theta}_t^k)$ for minimization.$\}$
9: Update gains $g_t^k = g_{t-1}^k + r_t^k$.
10: **end for**

(or gain) acquisition k received over the previous $t-1$ iterations. Intuitively, the Hedge strategy becomes more likely to pick points proposed by acquisitions that have 'done well' in the past (as defined by the objective function), and less likely to pick acquisitions that have performed poorly.

To adapt GP-Hedge for the task of minimization, we made an adjustment to line 8 of the algorithm: the sign of the reward is flipped. With this change, the higher the expected discrepancy at nominee point $\boldsymbol{\theta}_t^k$, the lower the corresponding acquisition's probability $p_t(k)$ becomes.

GP-Hedge's Full-Information Problem

Hedge was originally designed for a problem very similar to ours: the full-information adversarial multi-armed bandit problem (see Robbins (1952); Freund & Schapire (1997)). In this problem, an agent attempts to maximize his cumulative reward over a number of trials $t = 1, ..., T$. In each trial, the agent must choose between K of actions $k = 1, ..., K$ with unknown rewards $x_1(t), ..., x_K(t)$. Auer et al. (2000) specifies one trial as follows:

1. The adversary selects a vector $\boldsymbol{x}(t) \in [0, 1]^K$ of current rewards. The kth component $x_k(t)$ is interpreted as the reward associated with action k at trial t.

2. Without knowledge of the adversary's choice, the agent chooses an action by picking a number $k_t \in \{1, 2, ..., K\}$ and scores the corresponding reward $x_{k_t}(t)$.

3. The agent observes the entire vector of $\boldsymbol{x}(t)$ of rewards.

The parallels between this problem and ours are clear: over T iterations of Bayesian optimization, GP-Hedge wishes to pick nominees $\boldsymbol{\theta}_t^k$ that yield low expected discrepancies.

The key difference is the assumption of full information, or that the agent 'observes the entire vector ... of rewards' after each iteration. In the case of Bayesian optimization, the full reward vector can not be observed at every iteration (in our case, this would mean evaluating the objective at every nominee point at every iteration). Hoffman et al. (2011) fulfill this full-information requirement by giving each acquisition a reward equal to the updated model's estimate of the objective, $\mu_t(\boldsymbol{\theta}_t^k)$. For this setup to work properly, the rewards must be reasonably accurate - for instance, if an acquisition α_k nominates point $\boldsymbol{\theta}_t^k$, and the corresponding *true* expected discrepancy $E[f(\boldsymbol{\theta}_t^k)]$ is *low*, then the reward r_t^k should reflect that achievement (and vice versa: a high expected discrepancy should merit a low reward). In other words, the GP discrepancy model should be able to accurately estimate the expected discrepancy at each of the nominee points $\boldsymbol{\theta}_t^1, ..., \boldsymbol{\theta}_t^K$.

For the acquisition whose nominee was chosen and evaluated, this estimate can be trusted to be accurate. Hoffman et al. posit that because the objective function is assumed

to be smooth, the rewards for the other acquisitions should be reasonably accurate, as well. In practice, we have found this is often not true. As an example, consider the situation shown in Figure 6.

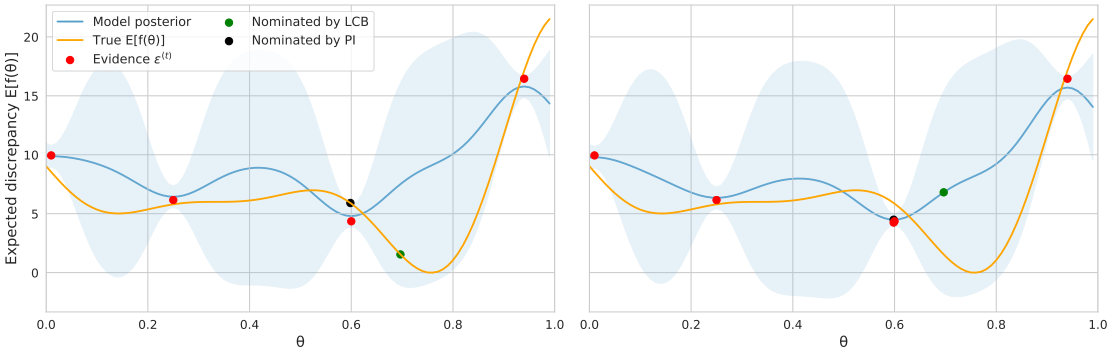


Figure 6: On the left - a GP model with 4 pieces of evidence and nominees of PI and LCB. On the right - the updated model after the point nominated by PI is chosen.

In this situation, α_{PI} and α_{LCB} nominate $\theta^{(1)} = 0.6$ and $\theta^{(2)} = 0.69$ respectively. GP-Hedge chooses the point nominated by α_{PI}, the objective is sampled at $\theta = 0.21$, and the model is updated (as shown on the right). Under the updated model, α_{PI} receives a reward of -4.48, and α_{LCB} receives a lower reward of -6.81. In reality, α_{LCB} picked a better point: its choice of θ yields an expected discrepancy of 1.54, and α_{PI}'s choice yields an expected discrepancy of 5.90. Because the GP model is inaccurate over $\theta \in (0.6, 0.85)$ (roughly), α_{LCB} is effectively punished for picking a better point.

It is unclear exactly how often these situations occur during the optimization process, but if they occur frequently, however, they could cause GP-Hedge to learn to favor a non-optimal acquisition.

2.5.2.2 GP-Exp3 Auer et al. (2000) also proposed a partial-information variant of Hedge called Exp3. Unlike Hedge, this algorithm gives a non-zero reward $\hat{r}_t^k$ *only* for the selected acquisition k at iteration t. In this case, assuming only positive rewards are given and that the objective is strictly positive, acquisitions become more likely to be selected purely by virtue of having been selected in the past. This can cause problems wherein acquisitions happen not to be selected for the first few iterations and consequently become very unlikely to be selected in all future iterations. Auer et al. address this issue by selecting acquisitions from a mixture of $p_t(k)$ and a uniform distribution, and also by giving larger awards to acquisitions it was unlikely to select:

$$\hat{p}_t(k) = (1-\gamma)p_t(k) + \frac{\gamma}{K}, \qquad \hat{r}_t^k = \frac{\mu_t(\boldsymbol{\theta}_t^k)}{\hat{p}_t(k)}. \tag{26}$$

As with Hedge, we adapted Exp3 for minimization (see Algorithm 2). Unlike the version of Exp3 proposed by Auer et al. (2000), GP-Exp3 skips the uniform distribution and directly uses $p_t(k)$, and also does not give more reward to acquisitions less likely to be selected. These mechanisms are unnecessary in our case because the 'rewards' are negative and the objective function non-negative, so selected acquisitions actually see their future selection probability temporarily *decrease*. This ensures GP-Exp3 periodically revisits historically under-performing acquisitions, as a decrease in the selected acquisition's probability of selection means an increase in the other acquisitions' probabilities. While this slows down GP-Exp3's ability to filter out bad acquisitions, it does prevent GP-Exp3

Algorithm 2 GP-Exp3 (for minimization)

Select parameter $\eta \in \mathbb{R}^+$.
Set $g_0^k = 0$ for $k = 1, ..., K$.
for t=1, 2, ... **do**
  Nominate points $\boldsymbol{\theta}_t^k$ from each acquisition function.
  Select nominee $\boldsymbol{\theta}_t = \boldsymbol{\theta}_t^k$ with probability $p_t(k) = \exp(\eta g_{t-1}^k) / \sum_{l=1}^K \exp(\eta g_{t-1}^l)$.
  Sample the objective function $\Delta_{\boldsymbol{\theta},t} = f(\boldsymbol{\theta}_t)$.
  Augment the data $\mathcal{E}_t = \{\mathcal{E}_{t-1}, (\boldsymbol{\theta}_t, \Delta_{\boldsymbol{\theta},t})\}$ and update the GP model accordingly.
  Receive reward $r_t = -\mu_t(\boldsymbol{\theta}_t)$ from the updated GP.
  Update gain for selected acquisition (all others receive a reward of 0): $g_t^k = g_{t-1}^k + r_t$.
end for

from preemptively ruling out acquisitions that do poorly initially but still may do well in the future.

That said, poorly-performing acquisitions should see much lower (i.e. more negative) rewards than those who do well, so their probability of selection should still see a much larger decrease. Our version of GP-Exp3 is thus still able to learn which acquisitions to choose as time goes on.

2.5.2.3 GP-Explorer We designed this last portfolio strategy specifically for the task of likelihood-free inference through Bayesian optimization, with inspiration from GP-Hedge and GP-Exp3. It works similarly to the others in that it selects nominee points based on a continually updated probability distribution, but the way it calculates rewards is completely different. Instead of setting the rewards equal to $-\mu_t(\boldsymbol{\theta})$ (which in our experience leads to Hedge and Exp3 overly favoring exploitative acquisitions like α_{PI} and α_{EI}), we attempt to reward each acquisition based on how much information it contributes to the final likelihood approximation. At each iteration, GP-Explorer estimates a bandwidth using the same procedure as Equation 8. After choosing point $\boldsymbol{\theta}_t = \boldsymbol{\theta}_t^k$ with probability $p_t(k)$ and sampling the objective to get $\Delta_{\boldsymbol{\theta},t}$, it estimates the likelihood of that point under the un-updated model: $\hat{\mathcal{L}}_{t-1}(\boldsymbol{\theta}_t)$. After updating the model, it calculates the new likelihood $\hat{\mathcal{L}}_t(\boldsymbol{\theta}_t)$, and gives a reward to acquisition k equal to the amount of change brought by that point: $\mid \hat{\mathcal{L}}_t(\boldsymbol{\theta}_t) - \hat{\mathcal{L}}_{t-1}(\boldsymbol{\theta}_t) \mid$. Unlike GP-Hedge and GP-Exp3, which give rewards based only on the posterior mean, this reward function makes use of the posterior variance, and is geared specifically towards LFI.

Also unlike GP-Hedge and GP-Exp3, this algorithm uses positive rewards. Since GP-Explorer only gives rewards to the selected acquisition, the probabilities $p_t(k)$ are mixed with a uniform distribution for the first t_{max} iterations. The uniform distribution, weighted strongly at first, is weighted less and less with each iteration until eventually GP-Explorer is only using $p_t(k)$.

The parameter h_{mult} corresponds to the '0.05' in Equation 8. In our experience, setting it equal to a low value incentivizes GP-Explorer to pick more exploitative acquisitions (e.g. α_{PI}, α_{EI}), and a high value to pick acquisitions which explore.

This portfolio strategy has three parameters to set (as opposed to just one with GP-Hedge and GP-Exp3). This makes it more flexible, but also more complicated to use. During our experiments, we set $t_{max} = 5K$ and $h_{mult} = 0.05$.

Because h is calculated on the fly, the likelihoods estimated using it may not be very accurate - especially during the early iterations, when limited observed data is available. This problem is mitigated, in part, by the use of the uniform distribution during early iterations.

Note that the algorithm above assumes that the final likelihood is calculated using

Algorithm 3 GP-Explorer

1: Select parameter $\eta \in \mathbb{R}^+$.
2: Select parameter $h_{mult} \in (0, 1)$.
3: Select parameter $t_{max} \in \mathbb{Z}^+$.
4: Set $g_0^k = 0$ for $k = 1, ..., K$.
5: **for** t=1, 2, ... **do**
6: Nominate points $\boldsymbol{\theta}_t^k$ from each acquisition function.
7: Calculate $p_t(k) = \exp(\eta g_{t-1}^k) / \sum_{l=1}^K \exp(\eta g_{t-1}^l)$ for each candidate k.
8: Calculate $u_t(k) = \frac{1}{K}(\min(t_{max} - t, 0)/t_{max})$.
9: Select nominee $\boldsymbol{\theta}_t = \boldsymbol{\theta}_t^k$ with probability $\hat{p}_t(k) = \frac{1}{2}(p_t(k) + u_t(k))$.
10: Sample the objective function $\Delta_{\boldsymbol{\theta},t} = f(\boldsymbol{\theta}_t)$.
11: Estimate a suitable bandwidth $\hat{h} = h_{mult}\big(\max \boldsymbol{\Delta}_{\boldsymbol{\theta}} - \min \boldsymbol{\Delta}_{\boldsymbol{\theta}}\big)$.
12: Estimate the likelihood of $\boldsymbol{\theta}_t$ under the un-updated model: $\hat{\mathcal{L}}_{t-1}(\boldsymbol{\theta}_t) = \Phi\Big(\frac{\hat{h} - \mu_{t-1}(\boldsymbol{\theta}_t)}{\sqrt{\upsilon_{t-1}(\boldsymbol{\theta}_t)}}\Big)$.
13: Augment the data $\mathcal{E}_t = \{\mathcal{E}_{t-1}, (\boldsymbol{\theta}_t, \Delta_{\boldsymbol{\theta},t})\}$ and update the model accordingly.
14: Estimate the likelihood of $\boldsymbol{\theta}_t$ under the updated model: $\hat{\mathcal{L}}_t(\boldsymbol{\theta}_t) = \Phi\Big(\frac{\hat{h} - \mu_t(\boldsymbol{\theta}_t)}{\sqrt{\upsilon_t(\boldsymbol{\theta}_t)}}\Big)$.
15: Receive reward $r_t = \mid \hat{\mathcal{L}}_t(\boldsymbol{\theta}_t) - \hat{\mathcal{L}}_{t-1}(\boldsymbol{\theta}_t) \mid$ from the updated GP.
16: Update gain for selected acquisition (all others receive a reward of 0): $g_t^k = g_{t-1}^k + r_t$.
17: **end for**

kernel density estimation. If another method for likelihood approximation is used, lines 2, 11, 12, 14, and 15 can be changed to use that method instead.

2.5.2.4 Baseline Portfolio For experimental purposes, we also include the use of the 'Baseline' portfolio, which selects new points by choosing uniformly at random from its nominees.

3 Research Objectives

Contents

The broad objective of this project is to investigate which acquisitions and portfolios perform best in likelihood-free inference via Bayesian optimization.

To make a rigorous comparison of each acquisition and portfolio's performance, we undergo a number of independent runs of Bayesian optimization, each using one acquisition or portfolio, on a number of objective functions. Over $t = 1, ..., t_{max}$ iterations of Bayesian optimization, each acquisition and portfolio sequentially acquires data according to its strategy, and updates its own Gaussian process model as more data is gathered. At each iteration t of optimization, we can calculate the approximate posterior for each acquisition and portfolio, using their GP models (Equation 15). We denote the approximate posterior constructed using acquisition/portfolio α's GP model at iteration t as $\hat{p}_t^{\alpha}(\boldsymbol{\theta}|\boldsymbol{y}_o)$. For each acquisition/portfolio α, we can compare the accuracy of the $\{\hat{p}_t^{\alpha}\}_{t=1}^{t_{max}}$ compared to the true posterior $p(\boldsymbol{\theta}|\boldsymbol{y}_o)$, using some metric for comparing probability distributions like the total variation (TV) distance (see Section 4.2).

In our experiments, we will deem the best-performing acquisitions and portfolios to be the ones which achieve a lower TV distance in the fewest iterations, or in other words, the ones with the best trade-off between posterior accuracy and number of iterations required to achieve it. We can visualize acquisition/portfolio performance by showing trade-off curves like those below:

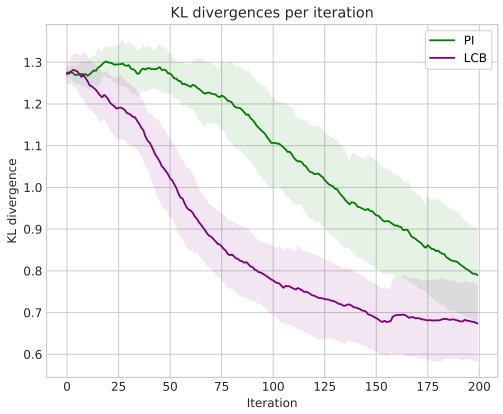


Figure 7: KL divergences over 200 iterations for PI and LCB acquisitions on the 'Six-Hump-Camel' (4.1) objective function.

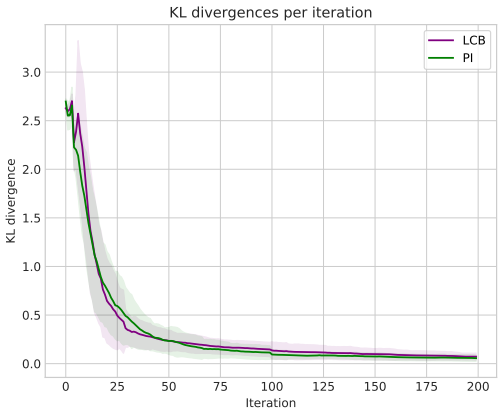


Figure 8: KL divergences over 200 iterations for PI and LCB acquisitions on the 'Sum-Of-Sinusoids' (4.1) objective function.

The left figure shows a situation where α_{LCB} clearly outclasses α_{PI}, achieving a more accurate approximate posterior in far fewer iterations. The right figure shows a situation where the two perform equally. Error bars represent standard deviations calculated using over 200 independent runs.

3.1 LFI Performance of Standard Acquisition Functions

We aim to compare the LFI performance of five standard acquisition functions (see 2.4.1): α_{LCB}, α_{EI}, α_{PI}, α_{Rand}, and $\alpha_{PostVar}$.

Comparison of their individual performances will be made using plots like in Figures 7 and 8 above. For each objective, we attempt to pick the acquisition with the best trade-off curve, and at the end, determine which one performs the best *overall* (if any). In addition, where possible, we attempt to explain why the acquisitions are successful (or unsuccessful) at various stages of the optimization process. For instance, we might find that exploration-oriented acquisitions (e.g. α_{LCB} and $\alpha_{PostVar}$) consistently perform the best during the early stages. These insights into how acquisition behaviour relates to LFI performance can help us better understand how to improve the standard acquisitions, and how to better understand (and potentially improve) portfolio algorithms.

3.2 LFI Performance of Portfolios of Acquisition Functions

We aim to compare the LFI performances of four portfolios of acquisition functions: GP-Hedge, GP-Exp3, GP-Explorer, and the Baseline portfolio (which selects the next point uniformly at random from the set of its nominees). See Section 2.5 for their explanations. All of the portfolios are set up with five 'base' acquisitions: α_{LCB}, α_{Rand}, α_{EI}, α_{PI}, and $\alpha_{PostVar}$.

Comparison of their individual performances can also be done with plots like in Figures 7 and 8. For each objective, we attempt to pick the portfolio(s) with the best trade-off curve(s), and also compare their performances against the best standard acquisition functions. Where possible, we attempt to explain portfolio behaviour (i.e. how they learn which acquisitions to pick) and how it relates to their LFI performance. Near the end of this report, we will also suggest modifications for improving their LFI performance.

3.3 Learning Which Acquisition Functions to Use

One of our main research objectives is to evaluate each portfolio's ability to learn which acquisitions to pick. We briefly go over each portfolio's success at doing so in Section 5.2. In the next section (5.3), we explore this issue in more detail by examining how and *why* each portfolio learns the way it does.

Reminder As a reminder, GP-Hedge, GP-Exp3, and GP-Explorer (see 2.5) work by tracking the cumulative *rewards* each acquisition has yielded throughout the optimization process. Acquisitions with higher (less negative) rewards see a higher probability of being chosen. Each of these three portfolios has a different strategy for giving rewards. Our goal is to explain how effective each portfolio's reward strategy is at facilitating learning to pick 'good' acquisitions, and learning not to pick 'bad' acquisitions.

Procedure We test this by setting up these three portfolios with five 'base' acquisitions (2.4.1): α_{LCB}, α_{EI}, α_{PI}, α_{Rand}, and α_{Bad}. The first three acquisitions are well-established in the literature on Bayesian optimization (Shahriari et al. (2016); Gutmann & Corander (2016); Jones (2001)) and have been shown to generally perform well during our experiments (5.1). The fourth, α_{Rand}, occasionally does well during the early stages of optimization (in our experiments), but generally under-performs compared to the first three. Finally, α_{Bad} is intentionally terrible, constantly seeking out the point of highest expected discrepancy and lowest uncertainty.

We run experiments and show how quickly each portfolio learns to pick α_{Bad} and α_{Rand} with low probability, and also how quickly each learns to pick the other three with high

probability. We can visualize these results with plots like Figure 9 below, which show the probabilities that a portfolio chooses each base acquisition's nominee over time:

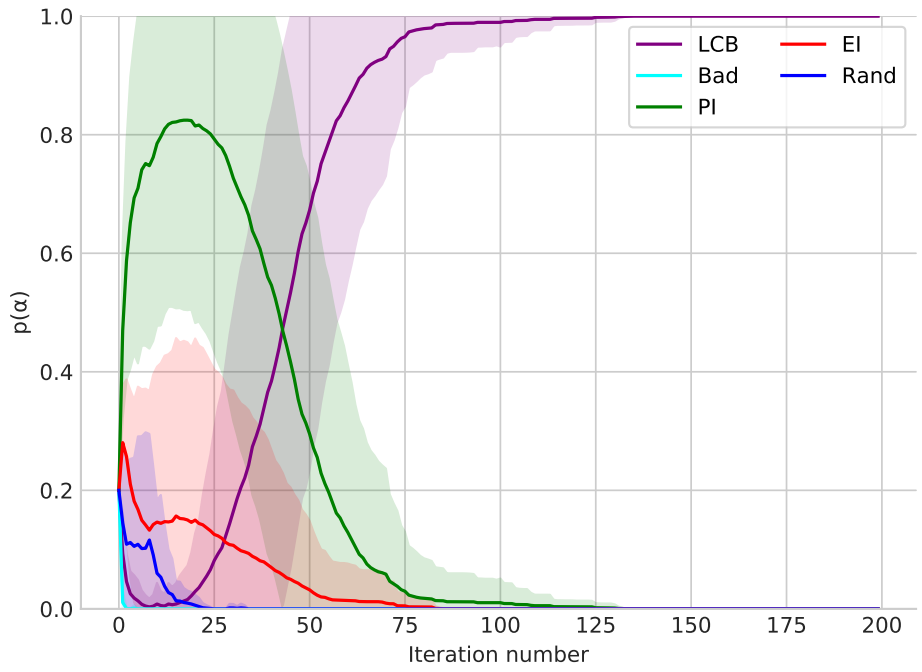


Figure 9: Depicted here are the probabilities that GP-Hedge chooses each base acquisition's nominee over time. Probabilities and error bars are calculated using about 200 runs of experiments on the 'Forrester' objective (4.1), and show GP-Hedge initially favoring α_{PI}, then strongly favoring α_{LCB}.

4 Experimental Methodology

Contents

4.1 Objective Functions used for Experiments

Overview: Synthetic Objective Functions

In earlier sections, we expressed the importance of the objective function $f : \boldsymbol{\theta} \to \Delta_{\boldsymbol{\theta}}$, stochastically mapping parameter settings to their associated discrepancies. In practice, this function is calculated as in Equation 5: $f(\boldsymbol{\theta}) = D(S(\boldsymbol{\theta}), \boldsymbol{y}_o)$. To avoid the hassle of dealing with a real simulator, we skip this process and instead choose several deterministic parametrized curves (with added Gaussian noise) to act as objective function:

$$f(\boldsymbol{\theta}) = g(\boldsymbol{\theta}) + \omega, \qquad \omega \sim \mathcal{N}(0, \sigma_n^2), \tag{27}$$

where g is the underlying deterministic function. The true likelihood (and posterior) can then be calculated as:

$$L(\boldsymbol{\theta}) = \Phi\Big(\frac{h - g(\boldsymbol{\theta})}{\sigma_n}\Big), \qquad p(\boldsymbol{\theta}|\boldsymbol{y}_o) \propto L(\boldsymbol{\theta})p(\boldsymbol{\theta}) \propto L(\boldsymbol{\theta}), \tag{28}$$

where $p(\boldsymbol{\theta})$ is the prior distribution over parameters (assumed to be uniform), h is the bandwidth chosen using 8, and Φ is the standard normal cumulative density function (cdf). The true posteriors here can then be compared to the approximate posteriors from Equation 15:

$$\hat{\mathcal{L}}_t^{\alpha}(\boldsymbol{\theta}) = \Phi\Big(\frac{h - \mu_t(\boldsymbol{\theta})}{\sqrt{\upsilon_t(\boldsymbol{\theta}) + \sigma_n^2}}\Big), \qquad \hat{p}_t^{\alpha}(\boldsymbol{\theta}|\boldsymbol{y}_o) \propto \hat{\mathcal{L}}_t^{\alpha}(\boldsymbol{\theta})p(\boldsymbol{\theta}) \propto \hat{\mathcal{L}}_t^{\alpha}(\boldsymbol{\theta})$$

using the methods outlined in Section 4.2 below. $\hat{\mathcal{L}}_t^{\alpha}$ here denotes the approximate likelihood created using a GP model f_t^{α} constructed with t iterations of Bayesian optimization with acquisition rule/portfolio α.

Forrester

The first and simplest objective is the Forrester function, which was already shown in Figure 1 in Section 2.2.2. It is a commonly used test function for Bayesian optimization, and it takes only one parameter. The code we used was adapted from the Forrester code from GPyOpt (The GPyOpt authors, 2016–). The Forrester function is defined as:

$$\begin{aligned} g_{forr}(\theta) &= (6 \cdot \theta)^2 \cdot \sin(12 \cdot \theta - 4) \\ f_{forr}(\theta) &= g_{forr}(\theta) + \omega, \qquad \omega \sim \mathcal{N}(0, 1.0). \end{aligned}$$

The function has a domain of [0, 1] and a minimizer of $\theta \approx 0.76$.

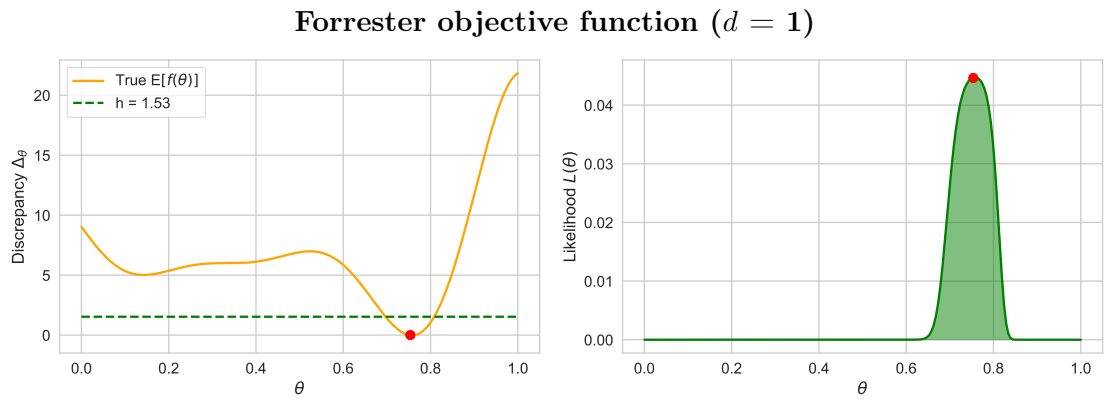


Figure 10: Forrester objective function and corresponding likelihood calculated with bandwidth 1.53. Maximum likelihood θ marked in red.

Sum-Of-Sinusoids

This is a slightly more complex synthetic objective we created specifically for this project. It also takes just one parameter, but it has many local minima and two global minima at $\theta \approx 0.18$ and $\theta \approx 0.81$. The Sum-Of-Sinusoids function is defined as:

$$g_{sumsin}(\theta) = \sin(30 \cdot \theta) - \cos(20 \cdot (\theta + 30)) - \sin(40 \cdot \theta)$$
$$f_{sumsin})\theta) = g_{sumsin}(\theta) + \omega, \qquad \omega \sim \mathcal{N}(0, 0.5).$$

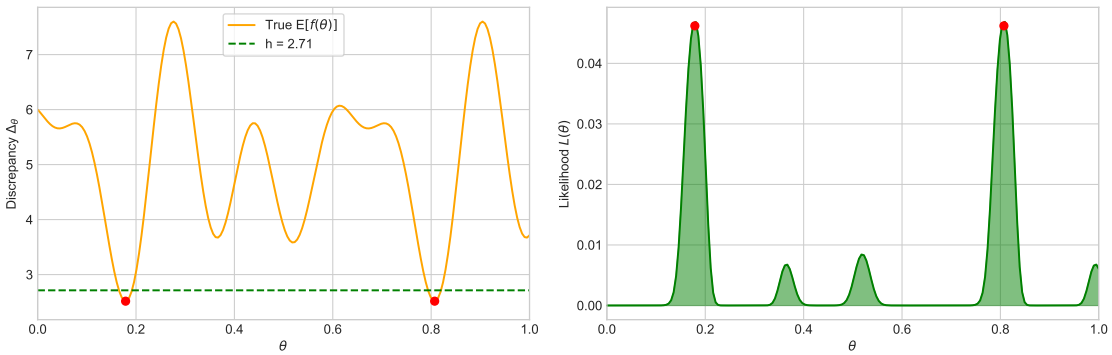


Figure 11: Sum-Of-Sinusoids objective function and corresponding likelihood calculated with bandwidth 2.71. Maximum likelihood θ marked in red.

Six-Hump-Camel

The next synthetic objective function is called *Six-Hump-Camel*, a common test function for Bayesian optimization (Marcin & Smutnicki, 2005). The code we used for it was also adapted from GPyOpt (The GPyOpt authors, 2016–). It takes two parameters as inputs, and has six local minima and two global minima at $\boldsymbol{\theta} = (0.0898, -0.7126)$ and $(-0.0898, 0.7126)$. Its equation is:

$$g_{sixhump}(\boldsymbol{\theta}) = (4 - 2.1 \cdot \theta_1^2 + \frac{\theta_1^4}{3} \cdot \theta_1^2) + (\theta_1 \cdot \theta_2) + \theta_2^2 \cdot (4 \cdot \theta_2^2 - r),$$
$$f_{sixhump}(\boldsymbol{\theta}) = g_{sixhump}(\boldsymbol{\theta}) + \omega, \qquad \omega \sim \mathcal{N}(0, 1.0).$$

Six-Hump-Camel objective function ($d = 2$)

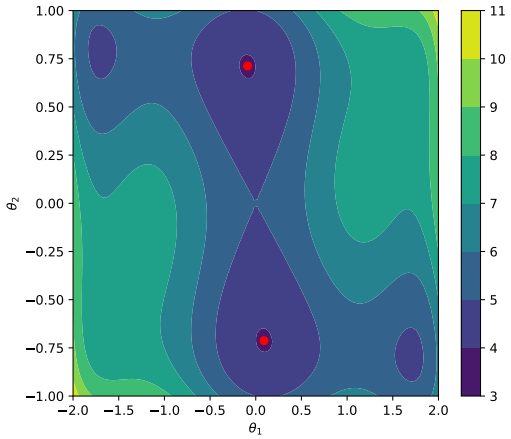


Figure 12: The true expected discrepancy given θ. Minimizers are marked in red.

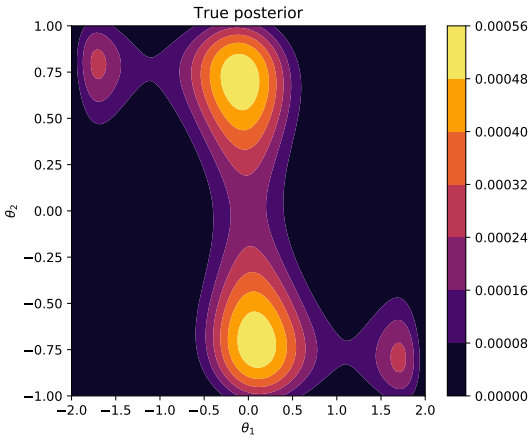


Figure 13: The true likelihood for Six-Hump-Camel, calculated with bandwidth $h = 4.17$. Points of maximum likelihood are marked in red.

Compared to the other objective functions, a very large proportion of Six-Hump-Camel's parameter space has a high likelihood. This should make it a challenging objective, as in order to construct a fully accurate posterior approximation, the acquisitions and portfolios must build an accurate model over a much larger space.

2D Gaussian

Next, we experiment on a Gaussian objective with 2 input dimensions. It is set up with bounds $[0, 20]$ for each parameter, a mean $\boldsymbol{\mu} = (10, 10)$, and a diagonal covariance matrix with diagonal elements $\boldsymbol{s}$ chosen independently and uniformly at random from $[1, 6]$. The underlying function is the standardized Euclidean distance between $\boldsymbol{\theta}$ and $\boldsymbol{\mu}$:

$$g_{gauss}(\boldsymbol{\theta}) = \sqrt{\sum_{i=1}^{d} \frac{(\theta_i - \mu_i)}{s_i^2}},$$
$$f_{gauss}(\boldsymbol{\theta}) = g_{gauss}(\boldsymbol{\theta}) + \omega, \qquad \omega \sim \mathcal{N}(0, 1.0).$$

The common minimizer of these Gaussian objectives is, of course, $\boldsymbol{\theta} = \boldsymbol{\mu}$.

Bacterial-Infections-2D

Finally, we run experiments on a modified version of the bacterial infections simulator we went over in Section 2.1.1. That simulator takes three parameters: the internal infection parameter $\theta_1 \in [0, 11]$, the external infection parameter $\theta_2 \in [0, 2]$, and the co-infection parameter $\theta_3 \in [0, 1]$. As this simulator is expensive to run, we only consider optimization of θ_2 and θ_3, assuming that the optimal value for θ_1 (3.589) is known. The range of discrepancies in this two-dimensional parameter space is approximately $(0, 35)$.

As the expected discrepancy function $E[f(\boldsymbol{\theta})]$ is unknown for this simulator, and non-noisy evaluation of f is impossible, we had to get creative to formulate the ground truth data for this simulator. First, we gathered over ten thousand evaluations of the simulator across the 2D parameter space, with a combination of random search, coarse grid search and Bayesian optimization. The resulting set of points can be seen in the right-hand section of Figure 14 below. We then used SciPy's (Jones et al., 2001–) `griddata` function to interpolate this data to get a 1000×1000 grid of points, shown on the left side of Figure 14. This data is what we use as the ground truth for our BacterialInfections2D experiments. To turn this discrete data into a continuous function, the discrepancy returned for any real-valued $\boldsymbol{\theta}$ is the average discrepancy of the 4 nearest points in the grid, weighted by distance:

$$g_{bactInf2D}(\boldsymbol{\theta}) = \frac{\sum_{\boldsymbol{\theta}' \in \Theta_{close}} \exp(-d(\boldsymbol{\theta}, \boldsymbol{\theta}'))\Delta'_{\boldsymbol{\theta}}}{\sum_{\boldsymbol{\theta}' \in \Theta_{close}} \exp(-d(\boldsymbol{\theta}, \boldsymbol{\theta}'))},$$
$$f_{bactInf2D}(\boldsymbol{\theta}) = g_{bactInf2D}(\boldsymbol{\theta}) + \omega, \qquad \omega \sim \mathcal{N}(0, 0.5),$$

where Θ_{close} is the set of the 4 closest points to $\boldsymbol{\theta}$ and $d(\boldsymbol{\theta}, \boldsymbol{\theta}') = \|\boldsymbol{\theta} - \boldsymbol{\theta}'\|$ is the Euclidean distance.

The resulting objective function is essentially a noisy snapshot of the simulator's true inner workings. It captures the complexity of the original simulator while still allowing us to have a solid ground truth objective for our later experiments. The large amount of local minima should also prove an interesting challenge for our acquisition rules and portfolios.

4.2 Assessing the Quality of the Approximate Posteriors

We use two main metrics for assessing the quality of the approximate posteriors $\hat{p}_t$ (as calculated in Equation 15): the total variation (TV) distance and the Kullback-Leibler (KL) divergence. To calculate these, we must calculate discrete approximations of the true and approximate posteriors, denoted as P and Q, respectively.

Recall that we are interested in simulators that take a finite number of real-valued parameters $\boldsymbol{\theta} \in \mathbb{R}^d$, and that the possible parameter settings are within some bounds

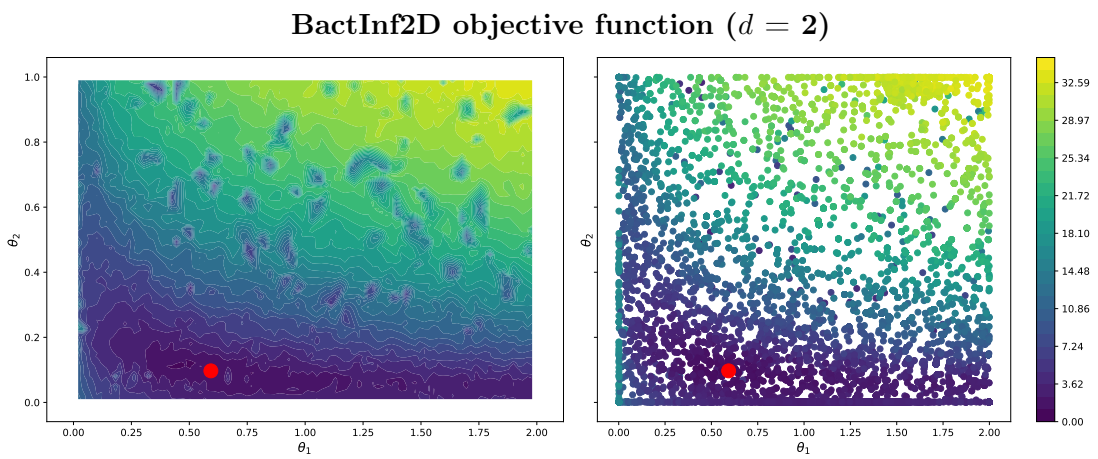


Figure 14: On the right: over 10 000 data points $(\theta_1, \theta_2, \Delta_{\boldsymbol{\theta}})$ gathered from the bacterial infections simulator (2.1.1). On the left: the data interpolated from these points which serves as the basis for $f_{bactInf2D}$. The minimizer of the true bacterial infections discrepancy function - $(0.593, 0.097)$ - is marked in red.

$[(\theta_1^{min}, \theta_1^{max}), ..., (\theta_d^{min}, \theta_d^{max})]$. For such an objective $f : \boldsymbol{\theta} \to \Delta_{\boldsymbol{\theta}}$, the set of its possible parameter settings Θ within those bounds, and a Gaussian process model $\hat{f}_t$, these discrete approximations can be calculated as follows:

1. Choose a finite number of parameter settings $\Omega \subset \Theta$, where Θ is the (infinite but bounded) set of all possible parameter settings.
2. Calculate unnormalized probabilities: $P = \{L(\boldsymbol{\theta}) \mid \boldsymbol{\theta} \in \Omega\}$ (using Equation 28).
3. Calculate unnormalized probabilities $Q = \{\hat{\mathcal{L}}_t(\boldsymbol{\theta}) \mid \boldsymbol{\theta} \in \Omega\}$ (using Equation 15).
4. Normalize by dividing P and Q element-wise by their respective sums.

In all cases, we choose Ω as a grid over possible parameter values.

Note that because we assume the use of a uniform prior, $p(\boldsymbol{\theta}|\boldsymbol{y}_o) \propto L(\boldsymbol{\theta})$. The elements of P are thus actually exactly proportional to their corresponding $p(\boldsymbol{\theta}|\boldsymbol{y}_o)$ (and those of Q to their corresponding $\hat{p}_t(\boldsymbol{\theta}|\boldsymbol{y}_o)$). The process of approximating a continuous pdf with a discrete grid was first proposed by Diggle & Gratton (1984), and is a commonly used method when the analytical form of p is unknown or intractable.

With P and Q, we can calculate the total variation (TV) distance:

$$D_{TV}(P, Q) = \frac{1}{2} \sum_{\omega \in \Omega} \mid P(\omega) - Q(\omega) \mid, \tag{29}$$

and the KL divergence:

$$D_{KL}(P, Q) = \sum_{\omega \in \Omega} P(\omega) \frac{P(\omega)}{Q(\omega)}. \tag{30}$$

These metrics are the main measure of success we use in our experiments: the quicker Bayesian optimization with acquisition α constructs a GP model which minimizes these metrics, the better α is.

4.3 Experimental Procedure

For each acquisition and portfolio, we ran a large number of independent Bayesian optimization trials on a number of different objective functions (listed in Section 4.1). The procedure for one trial is as follows:

1. Choose a simulator/objective function f.
2. Sample three points uniformly at random from within the simulator's parameter bounds, and create an initial Gaussian process model $\hat{f}$ with them (as in Section 2.3).
3. For each acquisition/portfolio α, and for 200 iterations each:
   (a) If α is a portfolio, save the probabilities $p_t(k)$ for choosing each nominee.
   (b) Select $\boldsymbol{\theta}_t$ using α.
   (c) Evaluate $f(\boldsymbol{\theta}_t)$ to get $\Delta_{\boldsymbol{\theta},t}$.
   (d) Update the Gaussian process model with $(\boldsymbol{\theta}_t, \Delta_{\boldsymbol{\theta},t})$, and save a copy of it for later.
4. Select a bandwidth h using Equation 8.
5. Calculate P as in Section 4.2.
6. For each acquisition/portfolio α and iteration $t = 1, ..., 200$:
   (a) Calculate Q_t^α as in Section 4.2 (using the saved model from before).
   (b) Calculate the accuracy of Q_t^α using the 4 metrics described in Section 4.2.

Q_t^α here denotes the discrete version of the approximate posterior constructed using t iterations of Bayesian optimization with acquisition rule α. For each simulator/objective function, we run this procedure tens or hundreds of times, and the results we show are averaged over these runs.

4.4 Software Library

For this project, I wrote a moderately sized library (about 3000 lines of Python code) implementing Gaussian processes, Bayesian optimization, acquisition functions, portfolios, KDE, objective functions, and a simple program for gathering, processing and visualizing experimental data. We also considered using existing libraries: specifically, GPyOpt (The GPyOpt authors, 2016–) and/or Spearmint (Harvard Intelligent Probabilistic Systems Group (HIPS), 2014–). In the end, we decided that implementing a small version of our own would work better for the following reasons:

- Spearmint does not implement the squared exponential covariance kernel (12), the closest they do implement being Matern52. Their Matern52 kernel only takes a single length scale (as opposed to one for each input dimension) as a hyperparameter, with no option for setting the signal variance, σ_f^2. Being able to set multiple length scales is helpful for modeling the bacterial infections simulator, especially.
- We found implementing our own acquisitions and portfolios within their frameworks was logistically difficult.
- Finally, we found it much simpler to track necessary information (i.e. portfolio base probabilities, model states) in our own pared-down library.

The code for the original bacterial infections simulator (including simulation, summary statistics and discrepancy function calculations) was provided to us by Marko Järvenpää.

5 Experiment Results

Contents

In running these experiments, our main goal is to show which acquisitions and portfolios perform best at likelihood-free inference via Bayesian optimization. Below is a brief review of how we measure performance at this task:

- Each acquisition/portfolio α is used to iteratively build a Gaussian process model $\hat{f}_t^\alpha$ of the objective function f, over iterations $t = 1, ..., t_{max}$ of Bayesian optimization (see Sections 2.3 and 2.4).
- These GP models are used to build a list of approximate posterior distributions $\{\hat{p}_t^\alpha(\boldsymbol{\theta}|\boldsymbol{y}_o)\}_{t=1}^{t_{max}}$ (see Section 2.3.3).
- For each iteration t, we calculate the total variation (TV) distance between the approximate posterior $\hat{p}_t^\alpha(\boldsymbol{\theta}|\boldsymbol{y}_o)$ and the true posterior $p(\boldsymbol{\theta}|\boldsymbol{y}_o)$.
- These TV distances are plotted versus iteration number t to show which acquisition rules α achieve the most accurate approximate posterior in the fewest iterations of Bayesian optimization.

These curves show a trade-off between approximate posterior accuracy and number of iterations required to achieve such a posterior. Example trade-off curves for α_{LCB} and α_{PI} are shown in Figure 9.

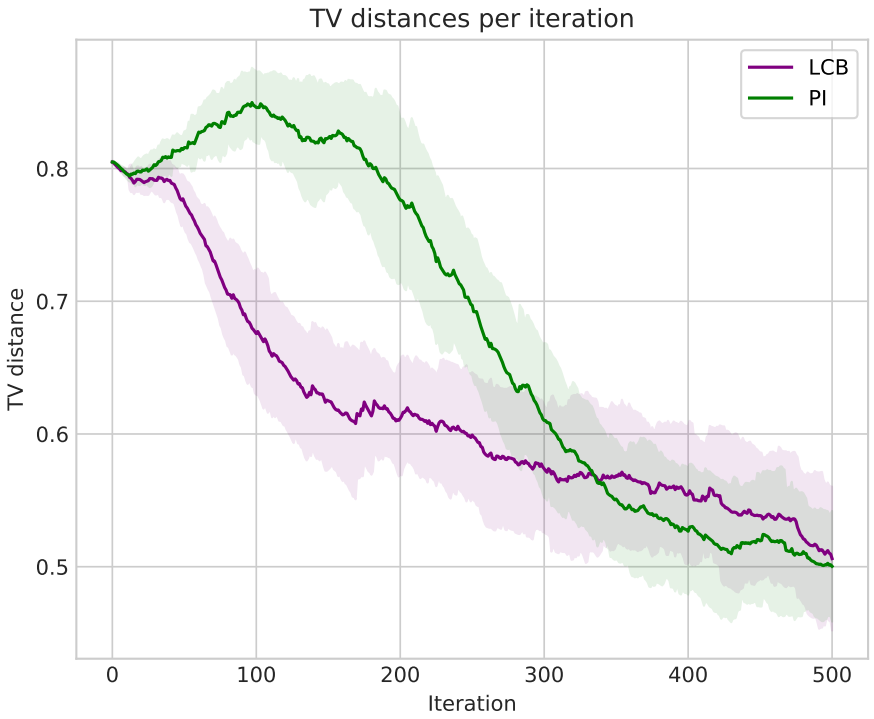


Figure 15: TV distances versus iteration number on the SixHumpCamel objective function (4.1). α_{LCB} is clearly better for the first 320 iterations, after which α_{PI} overtakes it.

5.1 LFI Performance of Standard Acquisition Functions

In this section, we present the results of experiments comparing the LFI performance of 5 standard acquisition functions: α_{PI}, α_{EI}, α_{LCB}, α_{Rand}, and $\alpha_{PostVar}$ (see Section 2.4.1 for how they work). On each objective function, we argue which acquisition function performed the best by examining trade-off curves like those in Figure 15. At the end, we evaluate which acquisition(s) performed the best overall, and give some comments on some standard acquisitions' behaviours.

5.1.1 Results on One Dimensional Simulators

On the one dimensional objective functions f_{forr} and f_{sumsin} (4.1), all of the standard acquisitions were able to achieve a low TV distance relatively quickly:

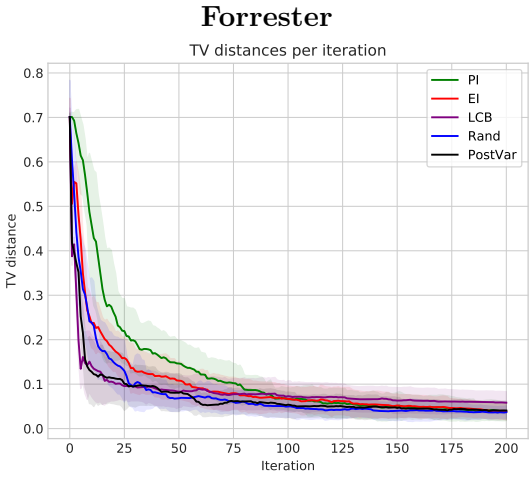


Figure 16: TV distances over time on the Forrester objective.

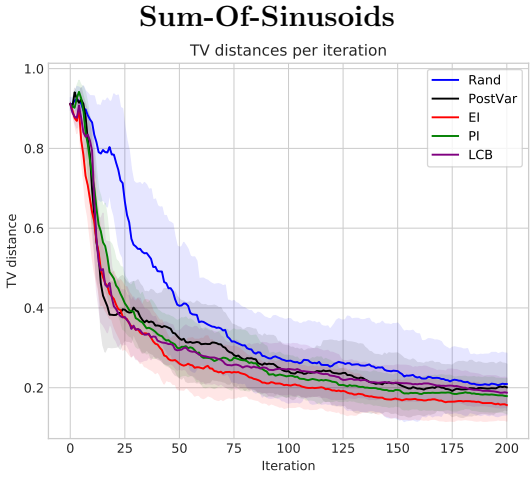


Figure 17: TV distances over time on the Sum-Of-Sinusoids objective.

On the Forrester objective, however, the exploitation-oriented acquisitions α_{EI} and α_{PI} showed a clear disadvantage - even worse than α_{Rand} during the first 75 or so iterations. The exploration-oriented acquisitions α_{LCB} and $\alpha_{PostVar}$ were easily the best during the early iterations, but the rest caught up by iteration 100, after which no acquisition had any discernible advantage.

On the Sum-Of-Sinusoids objective, most acquisitions performed roughly equally - apart from α_{Rand}, which was consistently the worst across all 200 iterations. No acquisition had any real advantage for the first 50 or so iterations, but after that α_{EI} took a small but consistent lead.

Overall, the one dimensional simulators are probably too simple for any significant differences in acquisition performance to become apparent.

5.1.2 Results on the 2D Gaussian

The 2D Gaussian (4.1) objective has a minimizer at $\boldsymbol{\theta} = (10, 10)$ (its mean) and a diagonal covariance matrix with diagonal elements chosen uniformly at random from $[1, 6]$. We ran experiments with multiple instances of this objective. The true posterior for one such instance is shown below in Figure 19, calculated with bandwidth $h = 0.5$.

On this objective, α_{LCB} performed the strongest, overall. During the first 20 or so iterations, however, $\alpha_{PostVar}$ was actually the best, clearly outclassing even α_{LCB}. Overall, though, α_{LCB} is the clear winner here, achieving the consistently lowest TV distances after

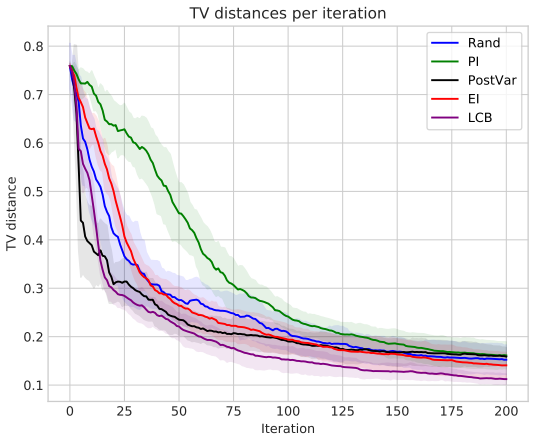


Figure 18: TV distances over time on the 2D Gaussian objective.

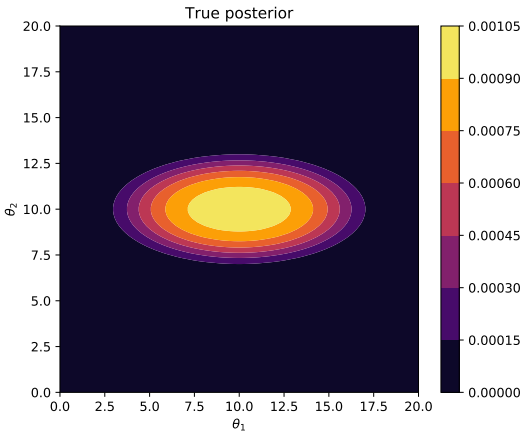


Figure 19: The true posterior for one instance of the 2D Gaussian objective.

that initial period. Again, α_{PI} was the worst, taking nearly 175 iterations to catch up to the rest.

5.1.3 Results on BactInf2D

The BactInf2D objective (4.1) has a single minimizer at $\boldsymbol{\theta} = (0.593, 0.097)$ and discrepancies roughly in the range $(0, 35)$. Figure 21 shows its true posterior calculated with bandwidth $h = 2.5$. On this objective, α_{LCB} again performed the best during the early iterations:

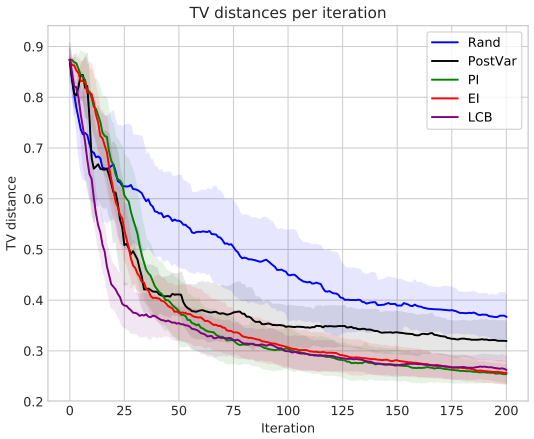


Figure 20: TV distances over time on the BactInf2D objective.

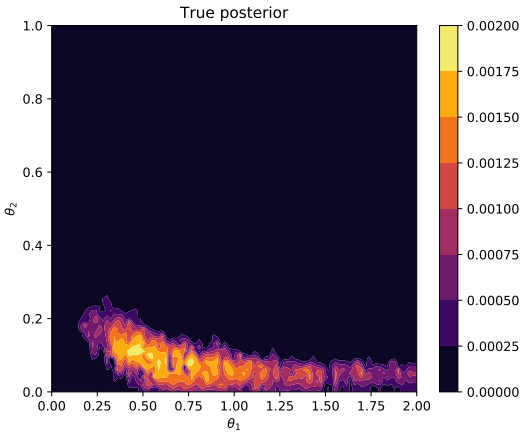


Figure 21: The true posterior for the BactInf2D objective.

α_{LCB} achieved the lowest TV distances over the first 50 iterations, and stayed on par with α_{EI} and α_{PI} thereon. $\alpha_{PostVar}$ fell far behind these three acquisitions after 50 iterations, presumably because it was still exploring areas of $\boldsymbol{\theta}$-space with low or almost-zero likelihoods. As might be expected on a complicated objective like this, α_{Rand} fell extremely far behind after only 25 iterations.

5.1.4 Results on Six-Hump-Camel

The Six-Hump-Camel objective (4.1) has two minimizers at $\boldsymbol{\theta} = (0.0898, -0.7126)$, $(-0.0898, 0.7126)$ and discrepancies roughly in the range $(3.96, 10.71)$. Figure 23 shows its

true posterior calculated with bandwidth $h = 4.17$.

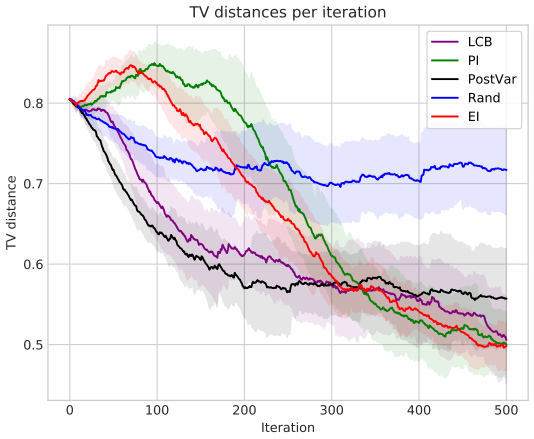


Figure 22: TV distances over time on the Six-Hump-Camel objective.

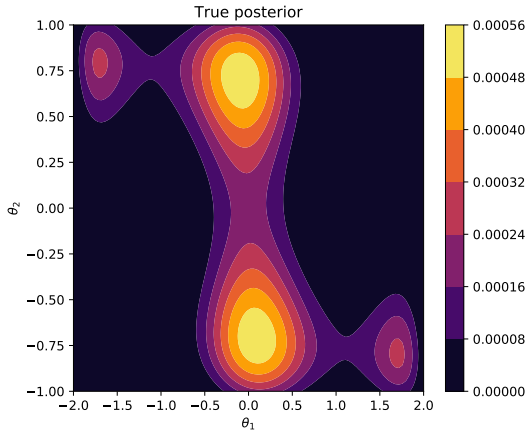


Figure 23: The true posterior for the Six-Hump-Camel objective.

With six local minima and two global minima, Six-Hump-Camel is arguably the most complex objective we experiment with. Because of this, we decided more iterations (500) were needed to gain a better understanding of the entire optimization process.

Unusually, the TV curves for α_{PI} and α_{EI} show a significant upward 'bump' early on. We have investigated why this occurs, and find that the following commonly occurring phenomenon causes this:

- When optimization is started, only three data points $(\boldsymbol{\theta}, \Delta_{\boldsymbol{\theta}})$ are available.
- α_{PI} and α_{EI} each repeatedly nominate points near the point of minimum observed discrepancy, until they find a local minimum.
- The acquisition(s) continues to nominate points in and around that local minimum.
  - As this is happening, the value of the constant prior mean $m_t(\boldsymbol{\theta})$ (which returns the average of observed discrepancies) slowly begins to become approximately equal to that local minimum.
  - As $m_t(\boldsymbol{\theta})$ lowers, the values of $\mu_t(\boldsymbol{\theta})$ for yet-unexplored $\boldsymbol{\theta}$ fall, also becoming approximately equal to that local minimum. In reality, most of the rest of the parameter space does not yield such low discrepancies, so the TV distance rises.
- Eventually, the values of $\mu_t(\boldsymbol{\theta})$ for the rest of the parameter space fall far enough that α_{PI} and α_{EI} begin to choose points outside that local minimum.

This phenomenon very clearly illustrates the importance of exploration during the early stages of optimization. Early on, α_{LCB} and (especially) $\alpha_{PostVar}$ were highly efficient due to their exploration of the whole parameter space. The other acquisitions were very slow to start, lagging behind α_{LCB} and $\alpha_{PostVar}$ for hundreds of iterations. This is almost certainly due to their tendency to preemptively exploit before they have a decent idea of where the regions of high likelihood might be.

During the later stages of optimization, however, it is clear that the exploitation-oriented acquisitions were superior. By iteration 300 or so, the acquisitions had all explored the parameter space enough to know, in broad strokes, where the regions of high likelihood are. At this point, α_{EI} and α_{PI} overtook α_{LCB} and $\alpha_{PostVar}$, presumably because they prioritized building their models around these high-likelihood regions, while the others were still exploring areas of $\boldsymbol{\theta}$-space with comparatively low likelihoods.

5.1.5 Discussion and Performance Rankings

Below, we show a table ranking each acquisition's performance across the five objectives. There is some inherent subjectivity in these rankings, so they are meant only to be a general indicator of acquisition performance. As a guideline, we generally ranked acquisitions who performed best early over those who did best later on.

Rank/Objective	Forrester	Sum-Of-Sinusoids	2D-Gaussian	Bact-Inf-2D	Six-Hump-Camel
1	$\alpha_{PostVar}$	α_{EI}	α_{LCB}	α_{LCB}	$\alpha_{PostVar}$
2	α_{LCB}	α_{LCB}/α_{PI}	$\alpha_{PostVar}$	α_{EI}	α_{LCB}
3	α_{Rand}	$\alpha_{PostVar}$	$\alpha_{EI}/\alpha_{Rand}$	α_{PI}	α_{EI}
4	α_{EI}	α_{Rand}	α_{PI}	$\alpha_{PostVar}$	α_{PI}
5	α_{PI}	–	–	α_{Rand}	α_{Rand}

Table 1: Rankings of standard acquisition LFI performances on each objective function.

Overall, α_{LCB} proved to be the best or second best standard acquisition on all of the objectives. While it was outperformed by $\alpha_{PostVar}$ on the Forrester and Six-Hump-Camel objectives, and by α_{EI} on Sum-Of-Sinusoids, it was still close behind. On the other objectives, α_{LCB} clearly showed the best trade-off curves. Even on Six-Hump-Camel, $\alpha_{PostVar}$ eventually falls behind α_{LCB} after iteration 320 (approximately), showing that α_{LCB} is better able to adapt to the requirements of each stage of the optimization process.

5.2 LFI Performance of Portfolios of Acquisition Functions

Portfolio Performance In this section, we show the results of experiments determining which portfolio algorithms - GP-Hedge, GP-Exp3, GP-Explorer, or Baseline (see 2.5) - have the best LFI performance. We also to compare their performance against the standard acquisition functions discussed in the last section. Each of these four portfolios were set up with the same five 'base' acquisitions from the last section: α_{PI}, α_{EI}, α_{LCB}, α_{Rand}, and $\alpha_{PostVar}$ (see 2.4.1). We measure performance by examining trade-off curves, like in the last chapter.

Portfolio Behaviour In the last section, we showed that none of our five standard acquisitions consistently perform the best on every objective and/or at every stage of optimization. Some, for instance α_{LCB} and $\alpha_{PostVar}$, are generally strongest during the early stages, and others like α_{EI} during the later stages. Because of this, we also examine each portfolio's probabilities $p_t(k)$ of selecting the nominee of each base acquisition over time, via plots like the one in Figure 9. In doing so, we are looking to see how well the portfolios learn to pick well-performing acquisitions with high probability, and under-performing acquisitions with low probability (e.g. α_{Rand} typically under-performs compared to the others).

Each of the following sections lists experiment results for one objective. At the beginning of each, we list the standard acquisitions α_{LCB}, α_{EI}, α_{PI}, α_{Rand}, and $\alpha_{PostVar}$ in order of their performance on that objective. This will help us keep track of which acquisitions the portfolios should learn to favor, and which ones they should learn to disfavor.

5.2.1 Results on Forrester

Best acquisitions: 1. $\alpha_{PostVar}$ 2. α_{LCB} 3. α_{Rand} 4. α_{EI} 5. α_{PI}

On the Forrester objective, we found that GP-Explorer and Baseline outperform GP-Hedge and GP-Exp3 by a small but significant amount:

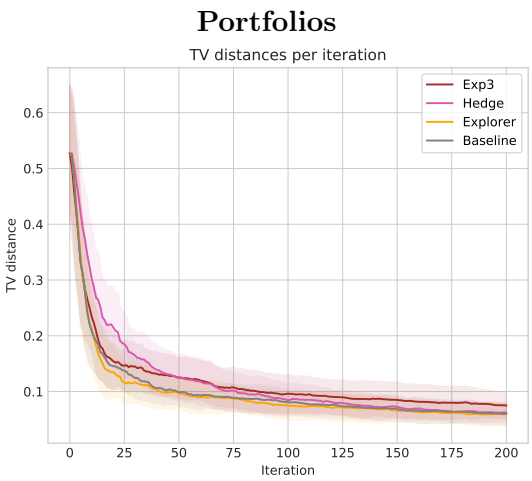


Figure 24: TV distances over time for the portfolios.

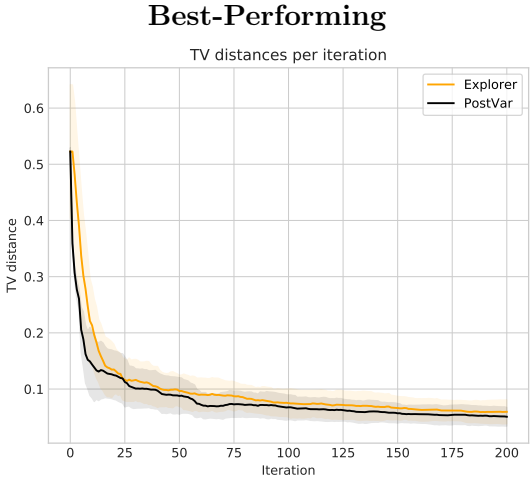


Figure 25: TV distances over time for the best-performing acquisition and portfolio: $\alpha_{PostVar}$ and GP-Explorer.

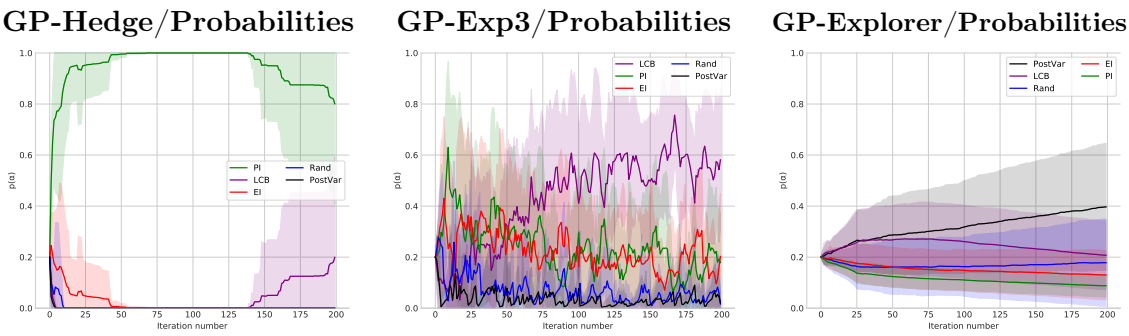


Figure 26: Probabilities each portfolio algorithm picks each base acquisition over time - on the Forrester objective.

GP-Hedge fell behind the others during the first 50 iterations, which makes sense as it strongly favored α_{PI}, the worst-performing acquisition for Forrester (see Figure 16 in the last section). GP-Exp3 performed on-par with GP-Explorer and Baseline during the early iterations, but later steadily learned to favor α_{LCB} over time, even though α_{LCB} actually performed the worst during later iterations on Forrester (albeit by a very thin margin; see Figure 16 in the last section). GP-Explorer, on the other hand, was the only portfolio to learn to favor the best-performing acquisition, $\alpha_{PostVar}$, and disfavor the two worst-performing acquisitions, α_{EI} and α_{PI}. Compared to the others, however, it learned very slowly - the probability of choosing $\alpha_{PostVar}$ only reaches 0.4 (from 0.2) after 200 whole iterations.

5.2.2 Results on Sum-Of-Sinusoids

Best acquisitions: 1. α_{EI} 2. α_{LCB}/α_{PI} 3. $\alpha_{PostVar}$ 4. α_{Rand}

On the Sum-Of-Sinusoids objective, each portfolio performs roughly equally:

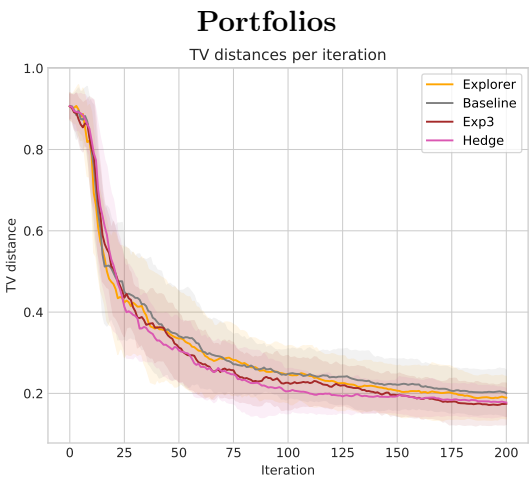


Figure 27: TV distances over time for the portfolio algorithms.

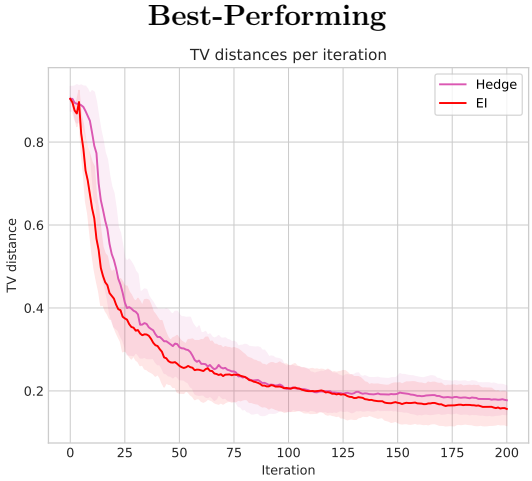


Figure 28: TV distances over time for the best acquisition α_{EI} and portfolio GP-Hedge.

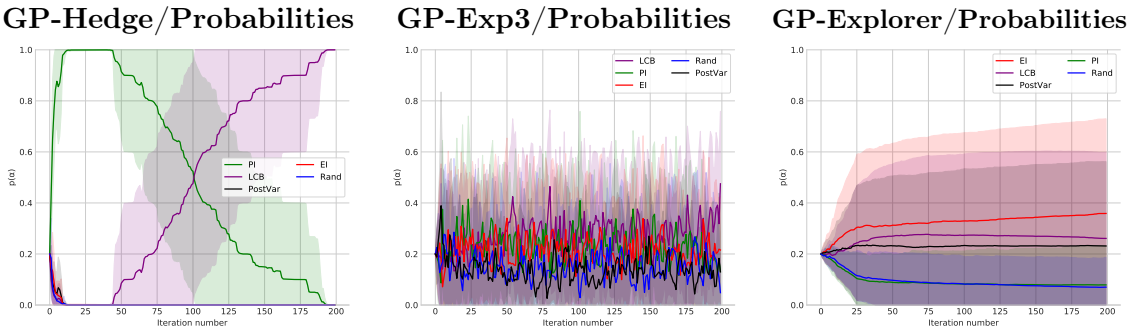


Figure 29: Probabilities each portfolio algorithm picks each base acquisition over time - on the Sum-Of-Sinusoids objective.

While performances are neck-and-neck during the first 25 iterations, GP-Hedge and GP-Exp3 do take a slight lead over GP-Explorer and Baseline afterwards - interesting, seeing as GP-Explorer was the only portfolio to place high probability on α_{EI}, the best-performing standard acquisition. Overall, we decided that GP-Hedge achieved the best trade-off curve, as GP-Exp3's curve is worse over iterations 30-50 and 80-150 (approximately).

Overall, Sum-Of-Sinusoids also appears to be too simple to draw any real conclusions from - although it is interesting to see the portfolios' drastically different acquisition-choosing strategies.

5.2.3 Results on the 2D Gaussian

Best acquisitions: 1. α_{LCB} 2. $\alpha_{PostVar}$ 3. $\alpha_{EI}/\alpha_{Rand}$ 4. α_{PI}

On the 2D Gaussian, GP-Explorer and Baseline showed a clear advantage over GP-Hedge and GP-Exp3:

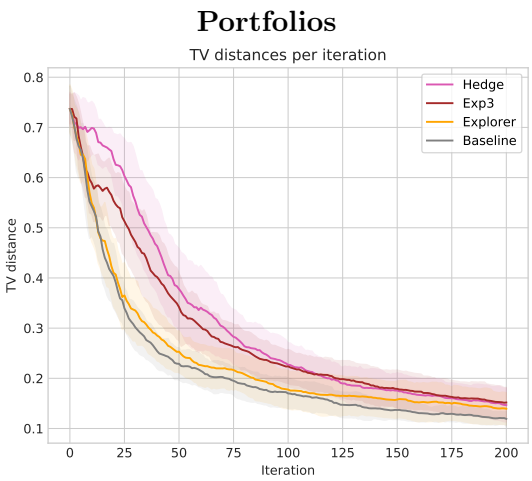


Figure 30: TV distances over time for the portfolio algorithms.

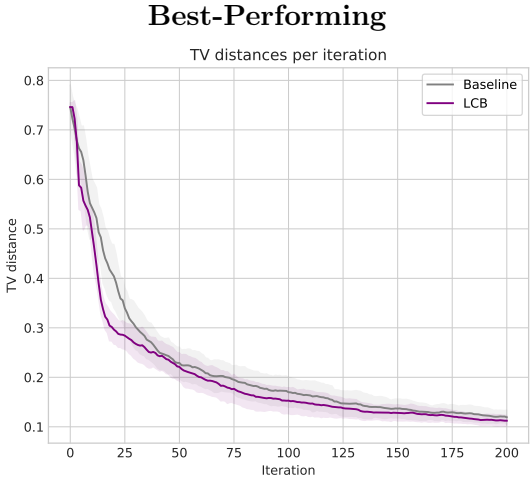


Figure 31: TV distances over time for the best acquisition and portfolio, α_{LCB} and Baseline.

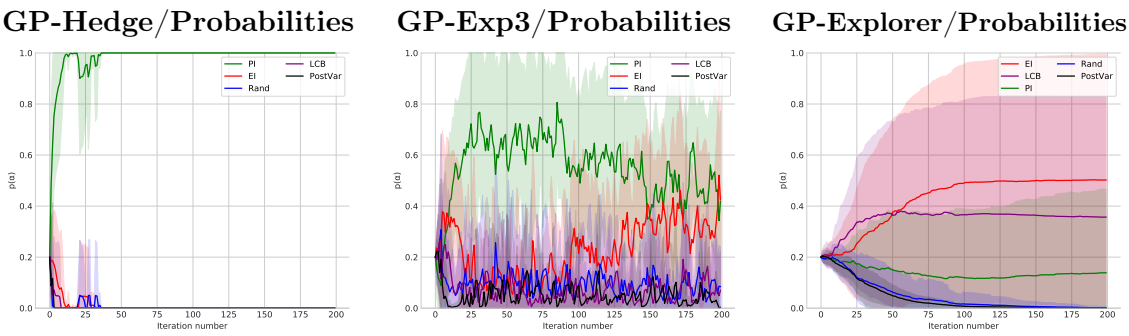


Figure 32: Probabilities each portfolio algorithm picks each base acquisition over time - on the 2D Gaussian objective.

GP-Hedge and GP-Exp3 tended to heavily favor the worst basic acquisition, α_{PI}, while GP-Explorer struck a balance between two better acquisitions: α_{EI} and α_{LCB}. Again, it is curious that even though GP-Explorer placed high probability on two of the best-performing acquisitions, it still lagged slightly behind Baseline throughout most of the optimization process.

Here, we again observe GP-Hedge learning to overwhelmingly favor α_{PI}, even though α_{PI} is by far the worst standard acquisition on this objective (see Figure 18 in the last section).

5.2.4 Results on BactInf2D

Best acquisitions: 1. α_{LCB} 2. α_{EI} 3. α_{PI} 4. $\alpha_{PostVar}$ 5. α_{Rand}

On the BactInf2D objective, no single portfolio shows a clear advantage:

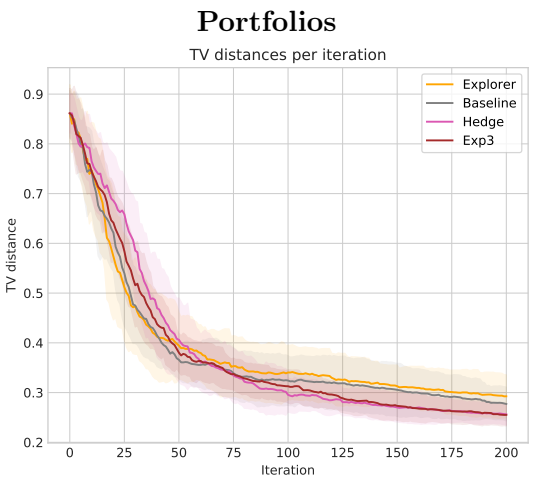


Figure 33: TV distances over time for the portfolio algorithms.

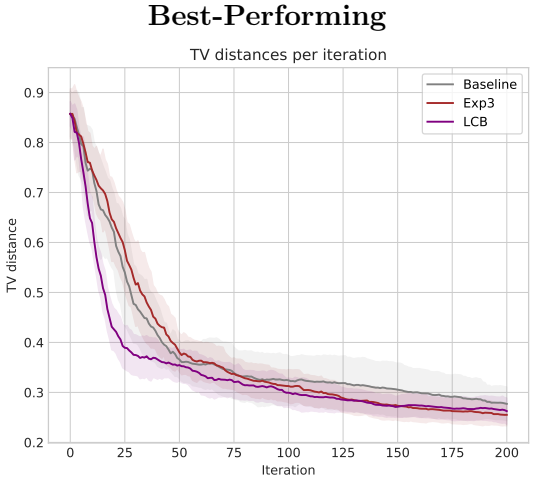


Figure 34: TV distances over time for the best acquisition α_{LCB} and portfolios GP-Exp3 and Baseline.

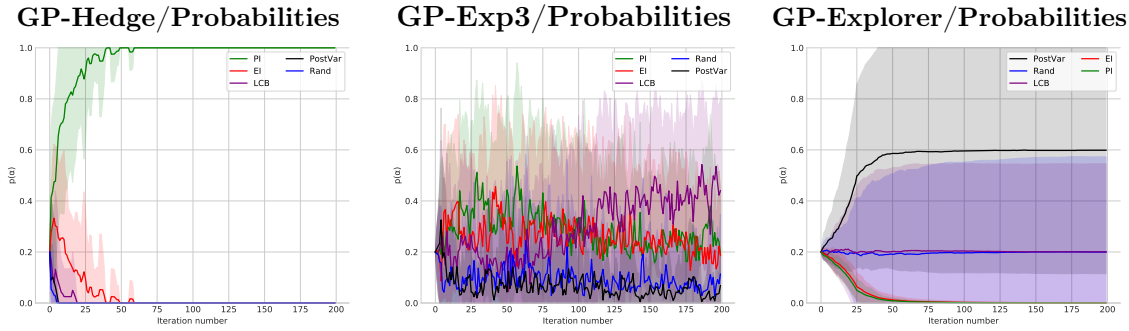


Figure 35: Probabilities each portfolio algorithm picks each base acquisition over time - on the BactInf2D objective.

GP-Explorer and Baseline show a clear advantage during the early iterations, but get overtaken by GP-Hedge and GP-Exp3 by iteration 55 or so.

Here, we again see the issue of GP-Hedge learning to pick α_{PI} with probability one - despite the fact that α_{PI}'s performance was mediocre at best on BactInf2D. GP-Explorer was also unsuccessful at learning to favor the best acquisitions, instead placing high probability on $\alpha_{PostVar}$. GP-Exp3 was the only portfolio that successfully learned to favor α_{LCB}, although not until 100 iterations had already passed.

5.2.5 Results on Six-Hump-Camel

Best acquisitions: 1. $\alpha_{PostVar}$ 2. α_{LCB} 3. α_{EI} 4. α_{PI} 5. α_{Rand}

On the Six-Hump-Camel objective, we ran five hundred iterations of Bayesian optimization instead of the typical two hundred. The results show that all of the portfolios except GP-Explorer fell into the trap of premature exploitation (see 5.1):

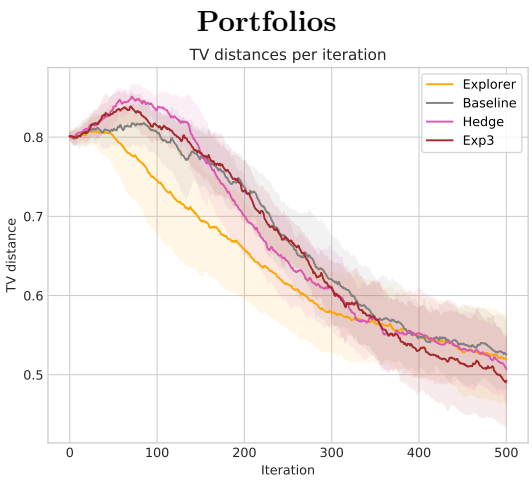


Figure 36: TV distances over time for the portfolio algorithms.

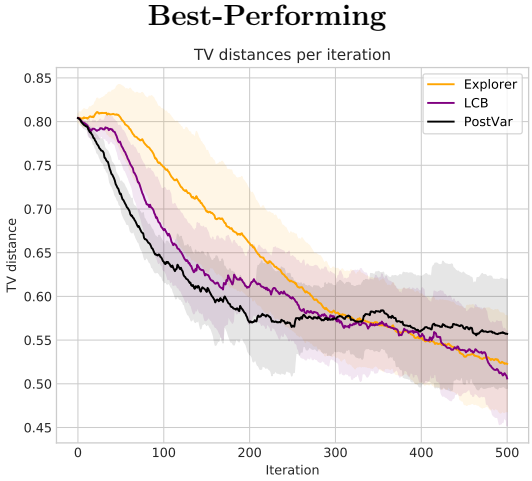


Figure 37: TV distances over time for the best acquisitions and portfolios.

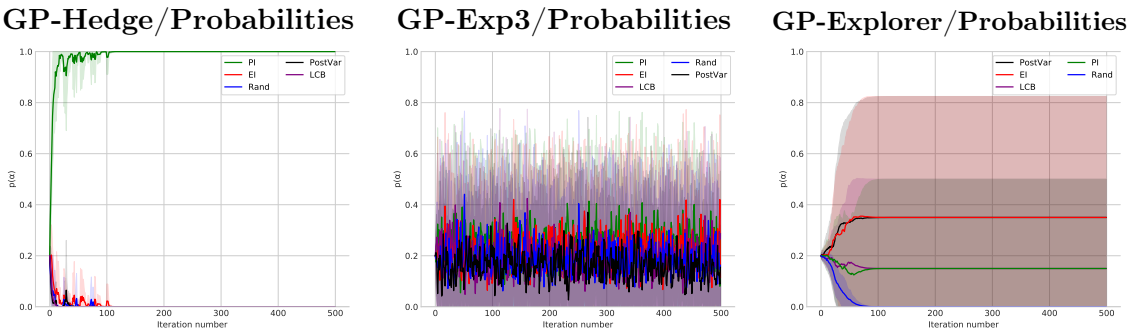


Figure 38: Probabilities each portfolio algorithm picks each base acquisition over time - on the Six-Hump-Camel objective.

GP-Hedge, again, learned to pick α_{PI} with probability one, leading its trade-off curve to closely resemble that of α_{PI}, the second worst-performing standard acquisition. Conversely, GP-Exp3 did not significantly raise or lower any acquisition's probability. All of its probabilities stayed roughly in the same range (about $[0.05, 0.4]$), although it did learn to slightly favor α_{EI} and α_{PI} over the long term. GP-Explorer was the only portfolio to place high probability on $\alpha_{PostVar}$, which helped it partially avoid the premature exploitation trap that the other portfolios seem to have fallen for.

5.2.6 Discussion and Performance Rankings

Below, we show a table ranking each portfolio's performance across the five objectives. As the acquisition we determined to be the best overall in the last section, we also include α_{LCB} in the rankings. Again, there is some inherent subjectivity in ranking LFI performance via these curves, so these rankings are meant only as a general indicator of performance.

Rank/Objective	Forrester	Sum-Of-Sinusoids	2D Gaussian	Bact-Inf-2D	Six-Hump-Camel
1	α_{LCB}	α_{LCB}	α_{LCB}	α_{LCB}	α_{LCB}
2	GP-Explorer	GP-Hedge, GP-Exp3	Baseline	GP-Explorer, Baseline	GP-Explorer
3	Baseline	GP-Explorer	GP-Explorer	GP-Hedge, GP-Exp3	GP-Hedge, GP-Exp3, Baseline
4	GP-Exp3	Baseline	GP-Exp3	–	–
5	GP-Hedge	–	GP-Hedge	–	–

Table 2: Rankings of portfolio LFI performances on each objective function.

The results are clear: no portfolio ever achieved a better trade-off curve than α_{LCB} in any of our experiments. The portfolios were able to beat other standard acquisitions on some experiments - for instance, α_{Rand}, $\alpha_{PostVar}$, and α_{PI} were frequently outperformed by the portfolios. Overall, however, the portfolio algorithms were consistently outperformed by α_{LCB}, especially during the early iterations.

5.3 Learning Which Acquisition Rules to Use

Reminder Three of our portfolio algorithms (GP-Hedge, GP-Exp3, GP-Explorer; see 2.5) work by tracking the cumulative rewards each base acquisition has yielded throughout the optimization process. Acquisitions with higher (less negative) cumulative rewards see a higher probability of being chosen. Each portfolio algorithm has its own strategy for determining what rewards to give to each base acquisition.

Objective In the last section, we explored these three portfolios' abilities to learn to pick well-performing acquisitions with high probability, and ill-performing ones with low probability. In this section, we explore this issue in more detail. In each sub-section, we briefly review how each of these portfolios gives rewards, and analyze why that portfolio behaves the way it does (e.g. for GP-Hedge, we would like to know why it always favors α_{PI}).

Procedure To test this, we set up GP-Hedge, GP-Exp3, and GP-Explorer portfolios with base acquisitions α_{PI}, α_{EI}, α_{LCB}, α_{Rand}, α_{Bad} (see 2.4.1). In particular, we would like to see these portfolio algorithms learn to pick α_{Bad} (our intentionally bad acquisition) with low or almost-zero probability as quickly as possible. In addition, we would like to see the portfolios learn to pick the first three acquisitions with high probability.

5.3.1 GP-Hedge

At each iteration t, GP-Hedge gives *every* acquisition $k = 1, ..., K$ a reward based on the model posterior mean:

$$r_t^k = -\mu_t(\boldsymbol{\theta}_t^k), \quad k = 1, ..., K$$

where $\boldsymbol{\theta}_t^k$ is candidate point nominated by acquisition k. This reward system causes GP-Hedge to overwhelmingly favor α_{PI} on most of the objectives:

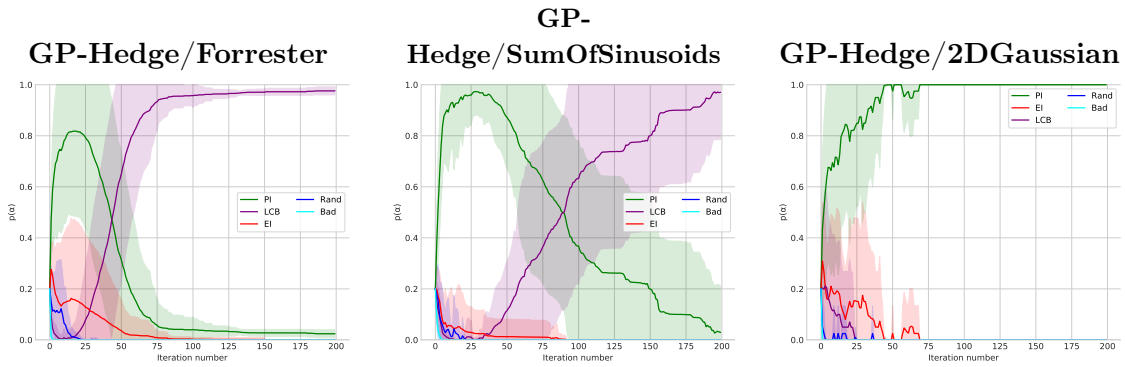


Figure 39: Probabilities of GP-Hedge choosing acquisitions' nominees over time, on the Forrester, Sum-Of-Sinusoids, and 2D Gaussian objectives (4.1).

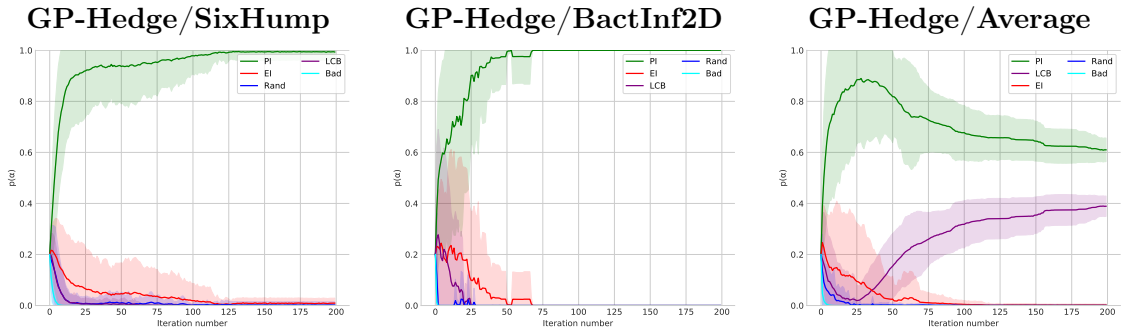


Figure 40: Probabilities of GP-Hedge choosing acquisitions' nominees over time, on the Six-Hump-Camel and BactInf2D objectives (4.1), and also averaged over all 5 objectives.

We believe this reward system is biased towards *exploitation*-oriented acquisitions, which explains GP-Hedge's affinity for α_{PI}. We have found α_{PI} to be the most exploitative of any acquisition we experimented with; it almost always nominates points near or around the minimizer of μ_t. Other acquisitions, namely α_{LCB} and $\alpha_{PostVar}$, often prioritize exploring areas at which $\mu_t(\boldsymbol{\theta})$ is high, but the uncertainty $v_t(\boldsymbol{\theta})$ is also high. Because *every* acquisition is given a reward at each iteration (even the ones whose nominees were not chosen), the exploratory acquisitions are repeatedly punished for picking high-uncertainty, high-expected-discrepancy points, while acquisitions like α_{PI} are repeatedly rewarded for 'playing it safe'.

In practice, this causes GP-Hedge to act too greedily to reap the long-term benefits that come with full exploration of the parameter space. GP-Hedge's extreme affinity for α_{PI} does make it especially efficient at learning to pick α_{Bad} and α_{Rand} with low probability, but this hardly seems relevant when it always blindly picks α_{PI} anyways.

5.3.2 GP-Exp3

At each iteration t, GP-Exp3 only gives a reward to the acquisition whose nominee was selected:

$$r_t^k = \begin{cases} -\mu_t(\boldsymbol{\theta}_t^k) & if\ \boldsymbol{\theta}_t^k\ chosen, \\ 0 & otherwise \end{cases}$$

Because the rewards are negative, this means that selected acquisitions actually see their future selection probability temporarily decrease. In practice, this causes the probabilities to oscillate quite significantly in the short term:

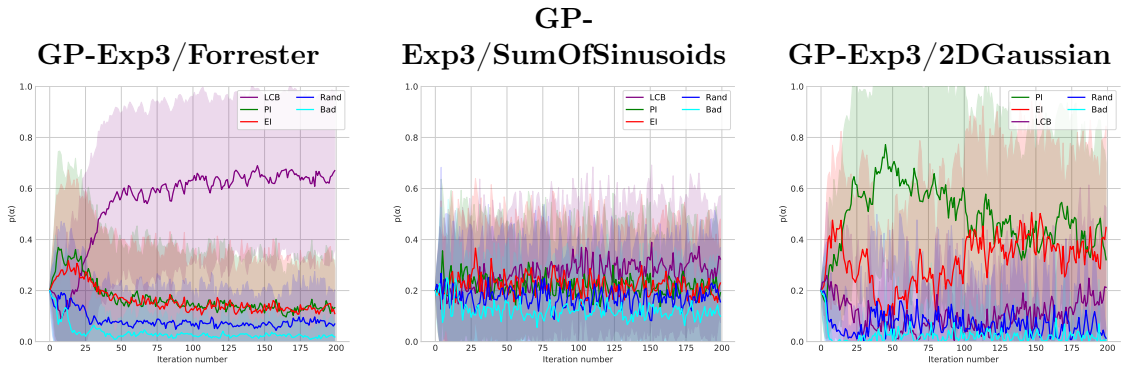


Figure 41: Probabilities of GP-Exp3 choosing acquisitions' nominees over time, on the Forrester, Sum-Of-Sinusoids, and 2D Gaussian objectives (4.1).

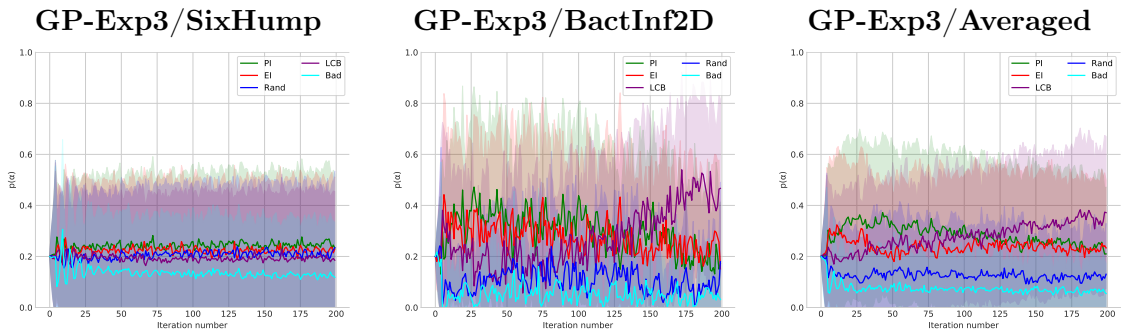


Figure 42: Probabilities of GP-Exp3 choosing acquisitions' nominees over time, on the Six-Hump-Camel and BactInf2D objectives (4.1), and also averaged over all 5 objectives.

We do not believe these short-term oscillations cause any real issues. On the other hand, it appears that this strategy of temporarily decreasing the probability of selected acquisitions makes it difficult for GP-Exp3 to learn to strongly favor any acquisition. This is not necessarily a strict downside - utilizing all of the standard acquisitions often makes GP-Exp3 perform more consistently than the other portfolios (see 5.2).

Compared to GP-Hedge, however, GP-Exp3 is much less effective at learning to ignore α_{Bad}. On the SixHump and Sum-Of-Sinusoids objectives in particular, it fails to lower $p(\alpha_{Bad})$ much below its initial starting value of 0.2. That said, it does pick α_{Bad} with the lowest probability on *every* objective, α_{Rand} the second lowest on all but Six-Hump-Camel, and also it learns to pick α_{LCB} with the highest probability fairly often - on 3 out of 5 objectives.

Overall, we believe GP-Exp3 is a successful improvement over GP-Hedge, as it does not fall into the α_{PI} trap that plagues GP-Hedge.

5.3.3 GP-Explorer

At each iteration t, GP-Explorer rewards only the selected acquisition. It attempts to reward acquisitions based on how much information their nominee points contribute to the final likelihood approximation. See 2.5.2.3 for details. Unlike the others, GP-Explorer gives positive rewards, and its selection probabilities tend to converge to a steady state after 50 or so iterations:

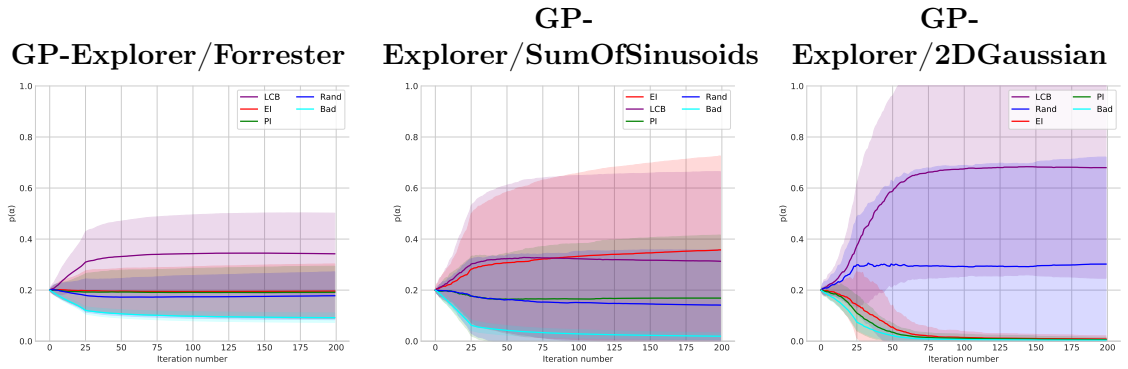


Figure 43: Probabilities of GP-Explorer choosing acquisitions' nominees over time, on the Forrester, Sum-Of-Sinusoids, and 2D Gaussian objectives (4.1).

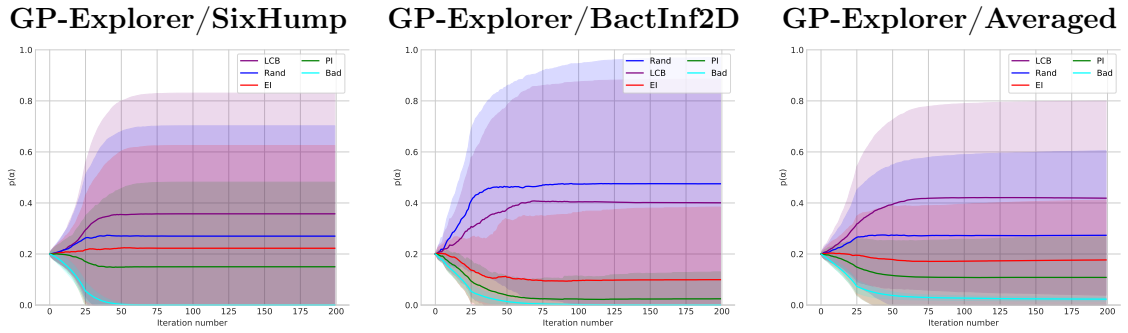


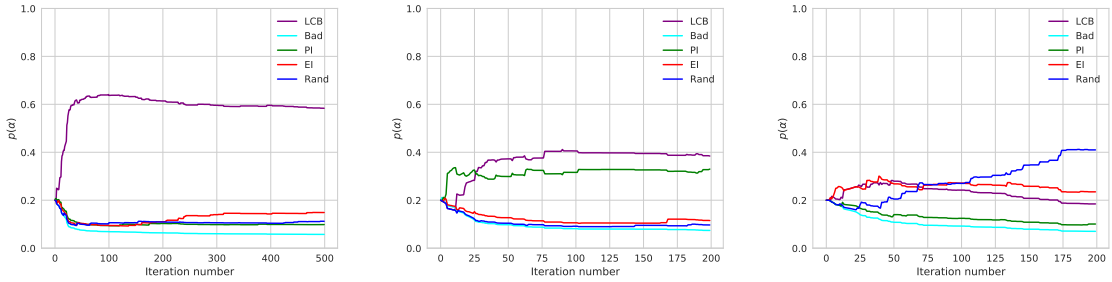
Figure 44: Probabilities of GP-Explorer choosing acquisitions' nominees over time, on the Six-Hump-Camel and BactInf2D objectives (4.1), and also averaged over all 5 objectives.

GP-Explorer shows some of the most desirable acquisition preferences: namely, it usually places high probability on α_{LCB}. It fares reasonably well at learning to ignore α_{Bad}, learning to pick it with probability nearly zero on all objectives except Forrester. However, it does often place high probability on α_{Rand}, a generally under-performing acquisition.

GP-Explorer suffers from two main downsides: early convergence of selection probabilities, and the large variability in probabilities it converges to.

Early Convergence The first is this issue of convergence - in almost all cases, its selection probabilities do not significantly change after iteration 50 or so. In previous sections (5.1), we have shown that exploration-oriented acquisitions are best during the early iterations, but later on exploitation-oriented ones are just as good or even better. By converging early, GP-Explorer is unable to capitalize on this shift in performance.

Variability There is a very high amount of variance in the final probability distribution GP-Explorer converges to - often towards high probability for α_{LCB}, other times for α_{Rand}, and less often for α_{EI}/α_{PI}. Figure 45 demonstrates some of these situations:



Relatively stable convergence over a long span (500 iterations).

One of the uncommon runs where PI sees high probability during convergence.

A run in which probabilities do not seem to have converged.

Figure 45: Probabilities GP-Explorer picks acquisitions' nominees over time on single independent runs - on the Forrester objective.

This high variability could be an indication that the probabilities GP-Explorer 'learns' are just as much a product of chance as they are of acquisition performance.

6 Conclusions

Standard Acquisition Functions As expected, we found that no standard acquisition function performed the best in all situations. In almost all cases, we found that exploration was unambiguously the best strategy during the early stages of optimization, as potential areas of high likelihood must be identified before their exploitation can begin. As optimization progresses and most or all of the high-likelihood areas have been identified, we found that exploitation was generally best.

Of all the standard acquisition functions, we found that the lower confidence bound (LCB) criterion was the best- or second best-performing on every objective. While some acquisitions tended to exploit too early (α_{PI}, α_{EI}) and others were unable to shift towards exploitation later on (namely, $\alpha_{PostVar}$), LCB was able to perform well over the whole optimization process.

Portfolios of Acquisition Functions As a whole, we were unable to get any of the portfolio algorithms to perform better than α_{LCB}. They did however, perform better than α_{EI} and α_{PI} during the early iterations, and better than $\alpha_{PostVar}$ during the later iterations. We believe this adaptive ability is not because they are truly learning which acquisitions are best over time, but because they use a mixture of them, making them a middle ground between the heavily exploitation- and exploration-oriented acquisitions.

Each portfolio has its own weakness: GP-Hedge's extreme affinity for α_{PI}, GP-Exp3's volatility and frequent inability to learn selection probabilities that are different from their starting values, and GP-Explorer's issues of selection probability convergence and variability. We believe that, with some modifications to their structure, we could improve their performance to rival that of α_{LCB}. We list some ideas for improvement in Section 6.2.

6.1 Open Questions

- We assume the hyperparameters of the Gaussian process model are fixed and known. In reality, there is always some level of uncertainty to their optimal values.
  - Hyperparameter settings can significantly affect data acquisition behaviour (Section 2.4.2). Can our results generalize to situations where the hyperparameters are constantly changing (e.g. updating them online as data is gathered)?
- All the simulators we tested on have constant observation noise variance σ_n^2. Can our results generalize to those with highly heteroschedastic observation noise?
- All of these experiments were based off of a single (nonparametric) method for calculating the approximate likelihood - kernel density estimation with a uniform kernel. Do our results hold when other methods are used (e.g. parametric likelihood approximations, or KDE with a Gaussian kernel)?
  - GP-Explorer is tailor-made to this form of likelihood approximation. Will its performance be as good with other forms of likelihood approximation?
- There are an innumerably large amount of possible simulators, and it is difficult to guess how these acquisitions/portfolios will perform on a simulator unlike any we have tested with.

6.2 Future Work

There are a huge number of possible avenues to expand upon the work done in this project. We considered trying many of them, but in the end found there was unfortunately no time to try all of them.

More Standard Acquisitions One avenue that comes to mind immediately is to try out more standard acquisition functions, for instance Thompson sampling (Thompson, 1933) or the 'expected integrated variance' (expintvar) acquisition (Järvenpää et al., 2017b). These would be interesting to experiment with both as standalone acquisitions and as 'base' acquisitions in a portfolio. In all of our experiments, the portfolios were set up with only five base acquisitions, so it would be interesting to see how they behave when a more diverse selection of acquisitions is available to them.

Modifying the Portfolios Other future work could also try modifying the portfolio algorithms to alleviate their problems. A modification to GP-Hedge's reward system could help its issue of 'greediness' (see 5.3) - perhaps, for example, by incorporating the posterior variance υ_t into the reward.

GP-Explorer could also be modified to help alleviate its issues of probability convergence and variability (see 5.3). For example, we toyed with the idea of incorporating a 'forgetting factor' $\beta \in [0, 1]$ into its reward system:

$$\boldsymbol{g}_t = \beta \boldsymbol{g}_{t-1} + \boldsymbol{r}_t,$$

where $\boldsymbol{r}_t$ is a vector of individual rewards r_t^k. This addition would cause GP-Explorer to slowly 'forget' acquisition performance from long ago, allowing it to adapt dynamically to the changing needs of each stage of optimization.

LCBEI Near the end of the project timeline, we also created a portfolio that attempts to directly leverage the knowledge that exploration is best in the beginning of optimization, and exploitation generally the best later on. Unfortunately, we did not have time to run full tests on it, so we include a description here as a starting point for potential future work.

This portfolio, which we simply called 'LCBEI', has only two base acquisitions: α_{LCB} and α_{EI} (although any pair with one exploration-oriented and one exploitation-oriented acquisition could work). At each iteration, it assesses approximately how much of the parameter space has been explored, then uses that knowledge to form probabilities of choosing α_{LCB} and α_{EI}:

Algorithm 4 LCBEI Portfolio

Select parameter $N \in \mathbb{Z}^+$.
for $t = 1, 2, ...$ **do**
  Sample N parameter settings uniformly at random from within the parameter boundaries: $\boldsymbol{\theta}^{(1)}, ..., \boldsymbol{\theta}^{(N)}$.
  $p_t(\alpha_{LCB}) = \frac{1}{N\sigma_f^2} \sum_{i=1}^{N} \upsilon_t(\boldsymbol{\theta}^{(i)})$.
  $\omega \sim U[0, 1]$.
  if $\omega < p_t(\alpha_{LCB})$ **then**
    $\boldsymbol{\theta}_t = \text{argmin}_{\boldsymbol{\theta}}\, \alpha_{LCB}(\boldsymbol{\theta})$.
  else
    $\boldsymbol{\theta}_t = \text{argmin}_{\boldsymbol{\theta}}\, \alpha_{EI}(\boldsymbol{\theta})$.
  end if
  Sample the objective function $\Delta_{\boldsymbol{\theta},t} = f(\boldsymbol{\theta}_t)$.
  Augment the data $\mathcal{E}_t = \{\mathcal{E}_{t-1}, (\boldsymbol{\theta}_t, \Delta_{\boldsymbol{\theta},t})\}$ and update the model accordingly.
end for

In Section 2.3.4, we briefly mentioned that the signal variance σ_f^2 can be interpreted as the maximum level of uncertainty the GP model can have about any point $\boldsymbol{\theta}$. With

LCBEI, we use this knowledge to estimate the proportion of the parameter space that has been explored. If much of it is still unexplored, then LCBEI places high probability on α_{LCB}, the generally best-performing acquisition in this situation. If much of it is already explored, then LCBEI places high probability on α_{EI}, the acquisition we found to be most effective when most or all high-likelihood areas of the parameter space have already been broadly located.

With this portfolio, we were only able to undertake 10 independent runs of Bayesian optimization on the Six-Hump-Camel objective, the results of which are shown below in Figure 46. Judging from these results alone, it appears that LCBEI is a slight improvement over GP-Explorer, but still far behind α_{LCB}.

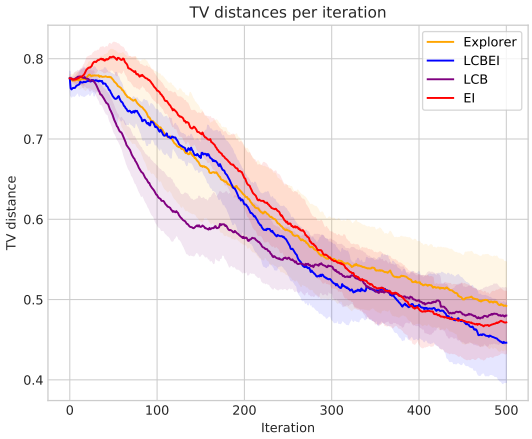


Figure 46: TV distances over time on the Six-Hump-Camel objective.

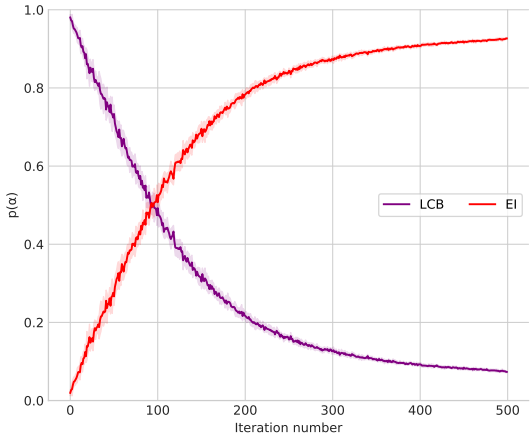


Figure 47: LCBEI acquisition selection probabilities over time.

References

Auer, Peter, Cesa-Bianchi, Nicolò, Freund, Yoav, and Schapire, Robert E. Gambling in a rigged casino: The adversarial multi-armed bandit problem. Electronic Colloquium on Computational Complexity (ECCC), 7(68), 2000.

Brochu, Eric, Cora, Vlad M., and de Freitas, Nando. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning, 2010.

Chaudhuri, Kamalika, Freund, Yoav, and Hsu, Daniel J. A parameter-free hedging algorithm. CoRR, 2009.

Diggle, Peter J. and Gratton, Richard J. Monte Carlo methods of inference for implicit statistical models. Journal of the Royal Statistical Society. Series B (Methodological), 46(2):193–227, 1984.

Fearnhead, Paul and Prangle, Dennis. Constructing summary statistics for approximate Bayesian computation: semi-automatic approximate Bayesian computation. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 74, 2012.

Freund, Yoav and Schapire, Robert E. A decision-theoretic generalization of on-line learning and an application to boosting. Journal of Computer and System Sciences, 55: 119–139, 1997.

Green, Peter J., Łatuszyński, Krzysztof, Pereyra, Marcelo, and Robert, Christian P. Bayesian computation: a summary of the current state, and samples backwards and forwards. Statistics and Computing, 25(4):835–862, Jul 2015.

Gutmann, Michael U. and Corander, Jukka. Bayesian optimization for likelihood-free inference of simulator-based statistical models. Journal of Machine Learning Research, 17, Jan 2016.

Harvard Intelligent Probabilistic Systems Group (HIPS). Spearmint. `https://github.com/HIPS/Spearmint`, 2014–.

Hoffman, Matthew W, Brochu, Eric, and de Freitas, Nando. Portfolio allocation for Bayesian optimization. In Uncertainty in Artificial Intelligence, pp. 327–336, 2011.

Jones, Donald R. A taxonomy of global optimization methods based on response surfaces. Journal of Global Optimization, 21(4):345–383, 2001.

Jones, Eric, Oliphant, Travis, Peterson, Pearu, et al. SciPy: Open source scientific tools for Python, 2001–. URL `http://www.scipy.org/`.

Järvenpää, Marko, Gutmann, Michael, Vehtari, Aki, and Marttinen, Pekka. Gaussian process modeling in approximate Bayesian computation to estimate horizontal gene transfer in bacteria. Annals of Applied Statistics, 2017a.

Järvenpää, Marko, Gutmann, Michael U., Pleska, Arijus, Vehtari, Aki, and Marttinen, Pekka. Efficient acquisition rules for model-based approximate Bayesian computation, 2017b.

Marcin, Molga and Smutnicki, Czesław. Test functions for optimization needs, 2005.

Meeds, Edward and Welling, Max. Gps-abc: Gaussian process surrogate approximate Bayesian computation, 2014.

Numminen, E., Cheng, L., Gyllenberg, M., and Corander, J. Estimating the transmission dynamics of streptoccocus pneuomoniae from strain prevalence data. Biometrics, 69: 748–757, 2013.

Nunes, M.A. and Balding, D.J. On optimal selection of summary statistics for approximate Bayesian computation. Statistical Applications in Genetics and Molecular Biology, 9, 2010.

Parzen, Emanuel. On estimation of a probability density function and mode. Ann. Math. Statist., 33(3):1065–1076, 09 1962.

Rasmussen, Carl Edward and Williams, Christopher K. I. Gaussian Processes for Machine Learning. The MIT Press, 2006.

Robbins, Herbert. Some aspects of the sequential design of experiments. 58(5):527–535, 1952.

Rosenblatt, Murray. Remarks on some nonparametric estimates of a density function. Ann. Math. Statist., 27(3):832–837, 09 1956.

Shahriari, Bobak, Swersky, Kevin, Wang, Ziyu, Adams, Ryan P., and de Freitas, Nando. Taking the human out of the loop: A review of Bayesian optimization. Proceedings of the IEEE, 104(1):148–175, 2016.

Snoek, Jasper, Larochelle, Hugo, and Adams, Ryan P. Practical Bayesian optimization of machine learning algorithms, 2012.

The GPyOpt authors. GPyOpt: A Bayesian optimization framework in python. `http://github.com/SheffieldML/GPyOpt`, 2016–.

Thompson, William R. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. Biometrika, 25(3/4):285–294, 1933.

Turner, Brandon M. and Zandt, Trisha Van. A tutorial on approximate Bayesian computation. Journal of Mathematical Psychology, 2012.

Wilkinson, Richard D. Accelerating abc methods using gaussian processes. In Proceedings of the 17th International Conference and Artificial Intelligence and Statistics (AISTATS), 2014.

A All Trade-off Curves in One Plot

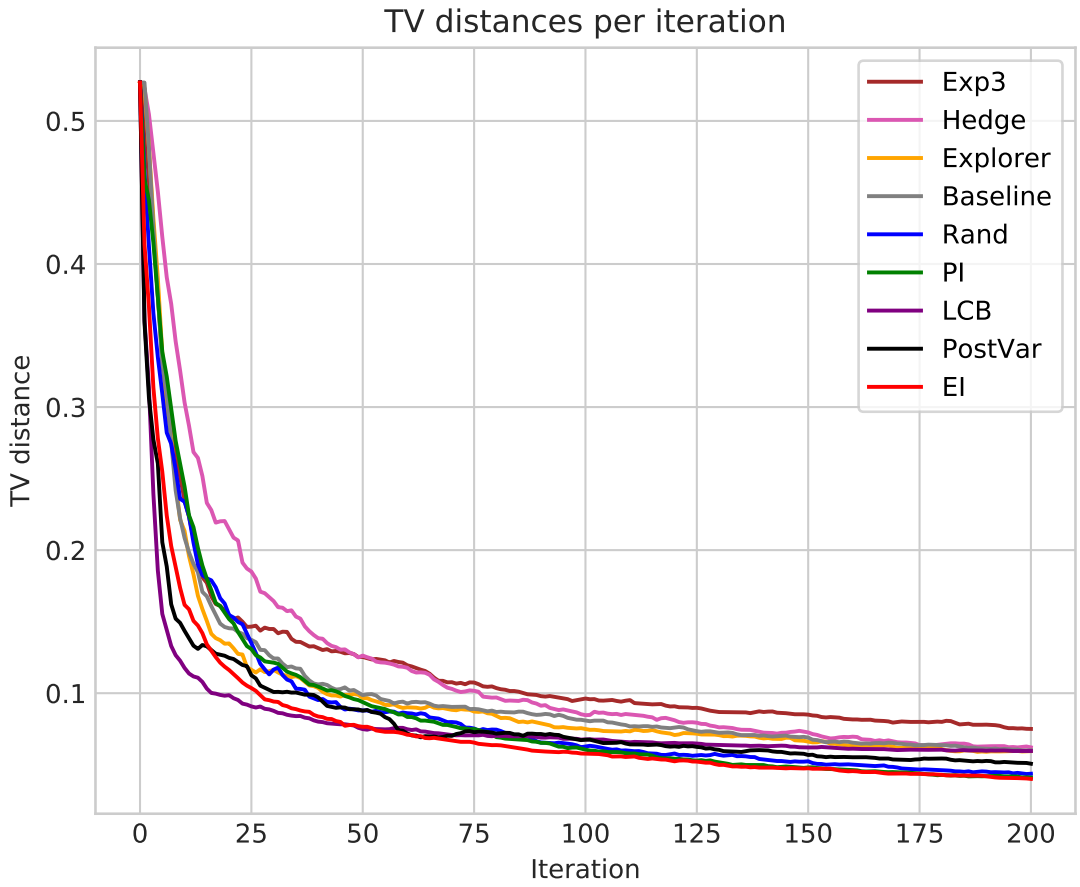


Figure 48: TV distances over time on the Forrester objective.

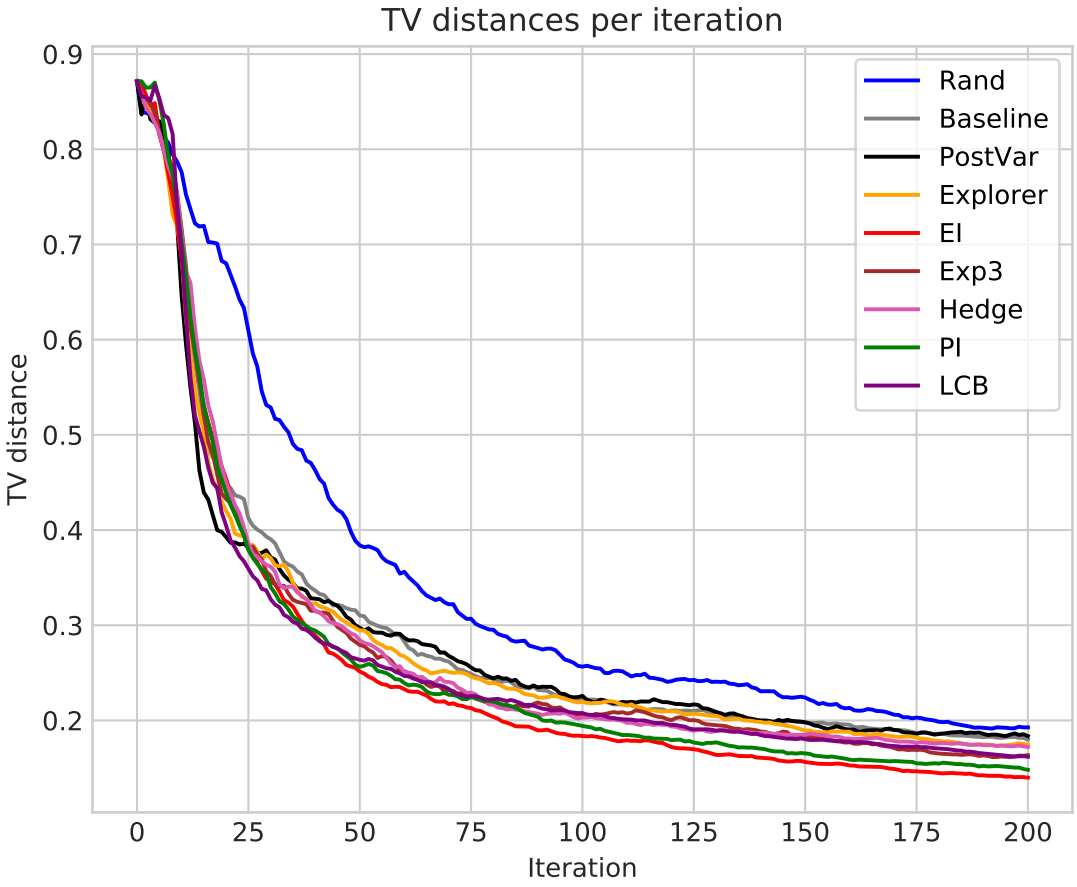


Figure 49: TV distances over time on the Sum-Of-Sinusoids objective.

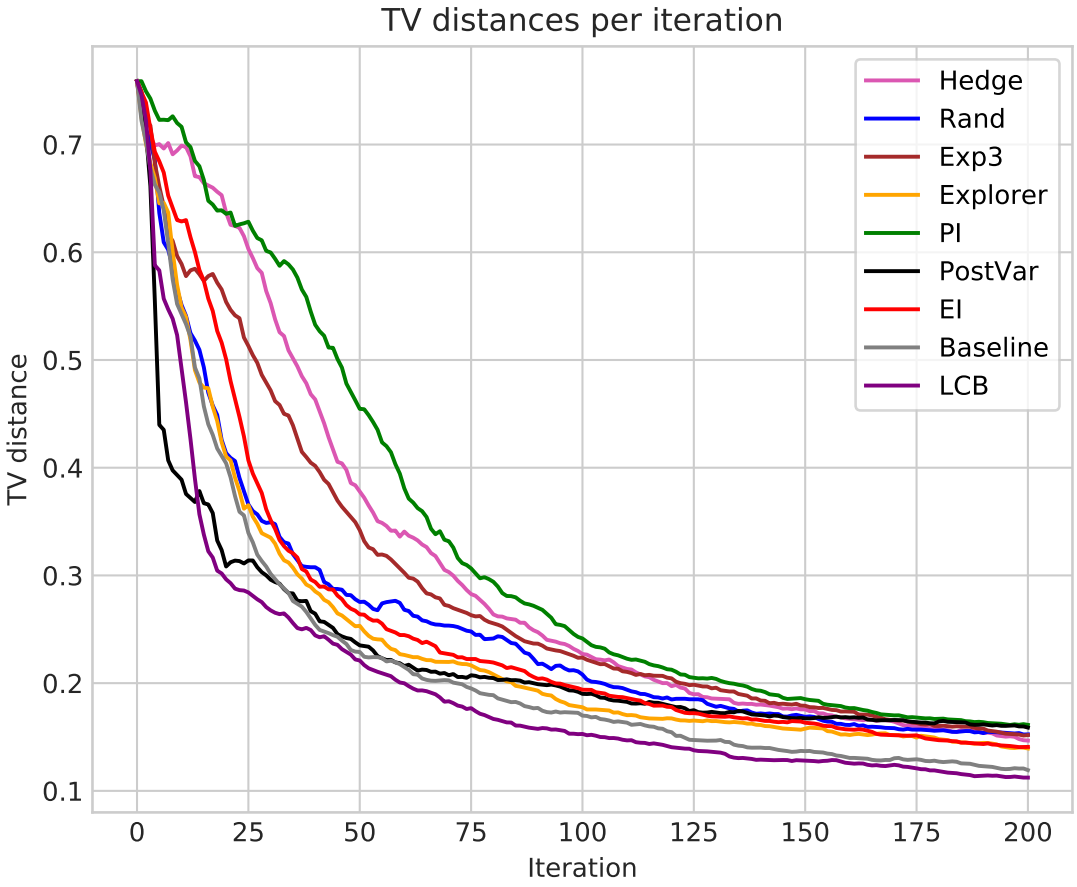


Figure 50: TV distances over time on the 2D Gaussian objective.

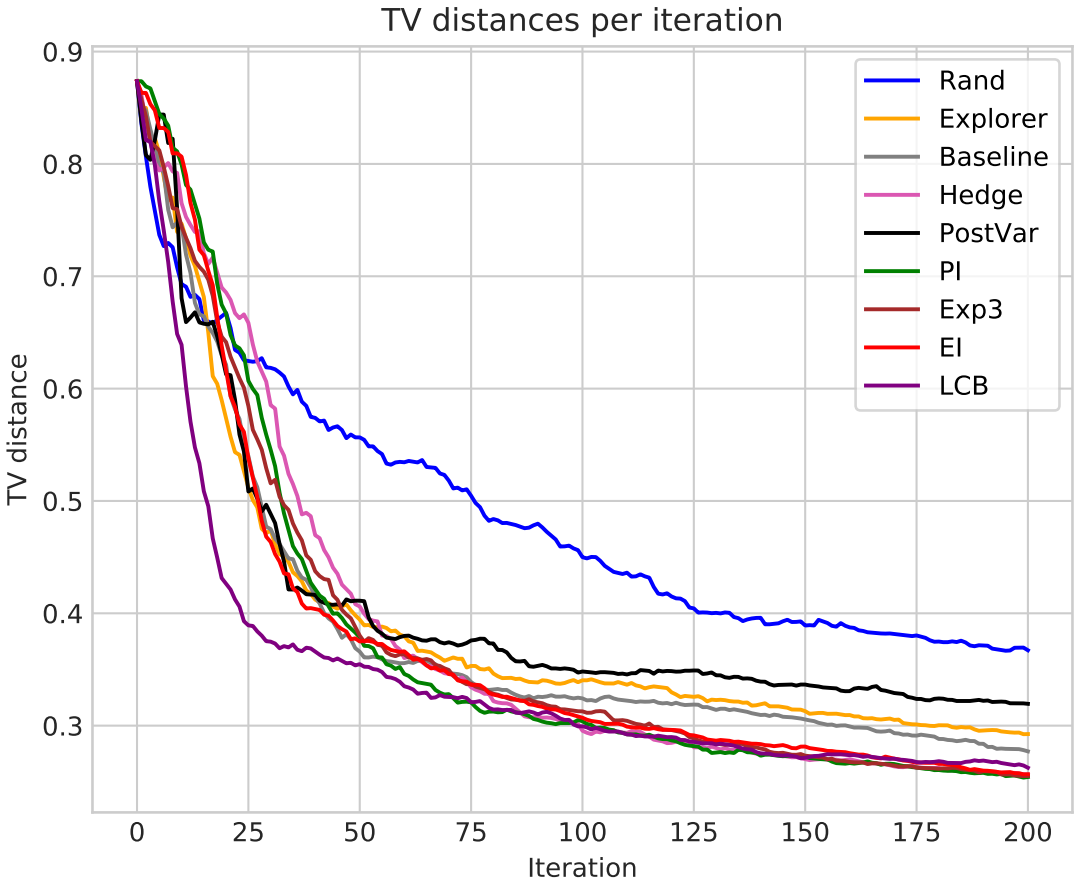


Figure 51: TV distances over time on the BactInf2D objective.