# Bayesian Optimization for Probabilistic Programs*

**Tom Rainforth** TWGR@ROBOTS.OX.AC.UK
*Department of Engineering Science, University of Oxford*

**Tuan Anh Le** TUANANH@ROBOTS.OX.AC.UK
*Department of Engineering Science, University of Oxford*

**Jan-Willem van de Meent** J.VANDEMEENT@NORTHEASTERN.EDU
*College of Computer and Information Science, Northeastern University*

**Michael A Osborne** MOSB@ROBOTS.OX.AC.UK
*Department of Engineering Science, University of Oxford*

**Frank Wood** FWOOD@ROBOTS.OX.AC.UK
*Department of Engineering Science, University of Oxford*

## Abstract

We present the first general purpose framework for marginal maximum a posteriori estimation of probabilistic program variables. By using a series of code transformations, the evidence of any probabilistic program, and therefore of any graphical model, can be optimized with respect to an arbitrary subset of its sampled variables. To carry out this optimization, we develop the first Bayesian optimization package to directly exploit the source code of its target, leading to innovations in problem-independent hyperpriors, unbounded optimization, and implicit constraint satisfaction; delivering significant performance improvements over prominent existing packages. We present applications of our method to a number of tasks including engineering design and parameter optimization.

## 1. Introduction

Probabilistic programming systems (PPS) allow probabilistic models to be represented in the form of a generative model and statements for conditioning on data (Carpenter et al., 2015; Goodman et al., 2008; Goodman and Stuhlmüller, 2014; Mansinghka et al., 2014; Minka et al., 2010; Wood et al., 2014). Their core philosophy is to decouple model specification and inference, the former corresponding to the user-specified program code and the latter to an inference engine capable of operating on arbitrary programs. Removing the need for users to write inference algorithms significantly reduces the burden of developing new models and makes effective statistical methods accessible to non-experts.

Although significant progress has been made on the problem of general purpose *inference* of program variables, less attention has been given to their *optimization*. Optimization is an essential tool for effective machine learning, necessary when the user requires a single estimate. It also often forms a tractable alternative when full inference is infeasible (Murphy, 2012). Moreover, coincident optimization and inference is often required, corresponding to a marginal maximum a posteriori (MMAP) setting where one wishes to maximize some variables, while marginalizing out others. Examples of MMAP problems include hyperparameter optimization, expectation maximization, and policy search (van de Meent et al., 2016).

In this paper we develop the first system that extends probabilistic programming (PP) to this more general MMAP framework, wherein the user specifies a model in the same manner as existing systems, but then selects some subset of the sampled variables in the program to be optimized, with the rest marginalized out using existing inference algorithms. The *optimization query* we introduce can be implemented and utilized in any PPS that supports an inference method returning a marginal likelihood estimate. This framework increases the scope of models that can be expressed in PPS and gives additional flexibility in the outputs a user can request from the program.

*Please cite this version:

Tom Rainforth, Tuan-Anh Le, Jan-Willem van de Meent, Michael A Osborne, and Frank Wood. Bayesian optimization for probabilistic programs. In *Advances in Neural Information Processing Systems*, pages 280–288, 2016

MMAP estimation is difficult as it corresponds to the optimization of an intractable integral, such that the optimization target is expensive to evaluate and gives noisy results. Current PPS inference engines are typically unsuited to such settings. We therefore introduce BOPP[1] (Bayesian optimization for probabilistic programs) which couples existing inference algorithms from PPS, like *Anglican* (Wood et al., 2014), with a new Gaussian process (GP) (Rasmussen and Williams, 2006) based Bayesian optimization (BO) package (Gutmann and Corander, 2016; Jones et al., 1998; Osborne et al., 2009; Shahriari et al., 2016a).

To demonstrate the functionality provided by BOPP, we consider an example application of engineering design. Engineering design relies extensively on simulations which typically have two things in common: the desire of the user to find a single best design and an uncertainty in the environment in which the designed component will live. Even when these simulations are deterministic, this is an approximation to a truly stochastic world. By expressing the utility of a particular design-environment combination using an approximate Bayesian computation (ABC) likelihood (Csilléry et al., 2010), one can pose this as a MMAP problem, optimizing the design while marginalizing out the environmental uncertainty.

Figure 1 illustrates how BOPP can be applied to engineering design, taking the example of optimizing the distribution of power between radiators in a house so as to homogenize the temperature, while marginalizing out possible weather conditions and subject to a total energy budget. The probabilistic program shown in Figure 2 allows us to define a prior over the uncertain weather, while conditioning on the output of a deterministic simulator (here Energy2D (Xie, 2012)-a finite element package for heat transfer) using an ABC likelihood. BOPP now allows the required coincident inference and optimization to be carried out automatically, directly returning increasingly optimal configurations.

BO is an attractive choice for the required optimization in MMAP as it is typically efficient in the number of target evaluations, operates on non-differentiable targets, and incorporates noise in the target function evaluations. However, applying BO to probabilistic programs presents challenges, such as the need to give robust performance on a wide range of problems with varying scaling and potentially unbounded support. Furthermore, the target program may contain unknown constraints, implicitly defined by the generative model, and variables whose type is unknown (i.e. they may be continuous or discrete).

On the other hand, the availability of the target source code in a PPS presents opportunities to overcome these issues and go beyond what can be done with existing BO packages. BOPP exploits the source code in a number of ways, such as optimizing the acquisition function using the original generative model to ensure the solution satisfies the implicit constaints, performing adaptive domain scaling to ensure that GP kernel hyperparameters can be set according to problem-independent hyperpriors, and defining an adaptive non-stationary mean function to support unbounded BO.

Together, these innovations mean that BOPP can be run in a manner that is fully black-box from the user's perspective, requiring only the identification of the target variables relative to current syntax for operating on arbitrary programs. We further show that BOPP is competitive with existing BO engines for direct optimization on common benchmarks problems that do not require marginalization.

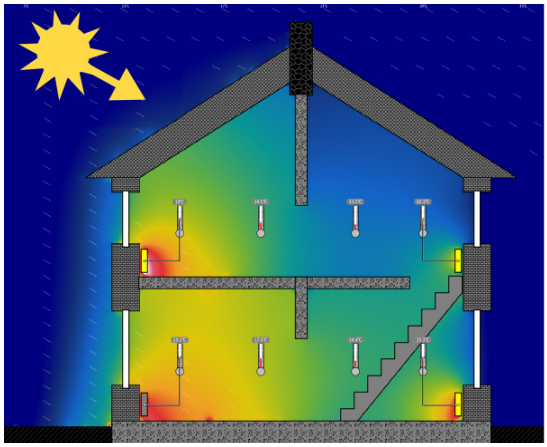
## 2. Background

### 2.1 Probabilistic Programming

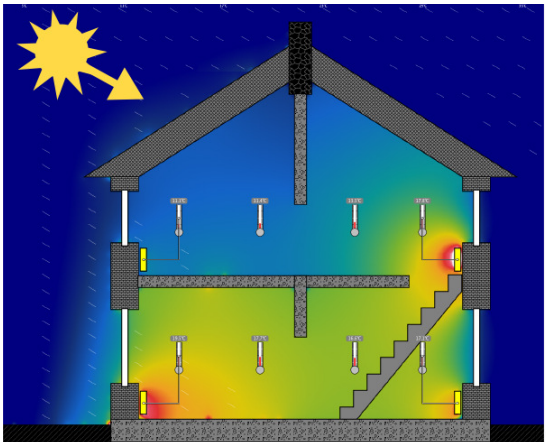
Probabilistic programming systems allow users to define probabilistic models using a domain-specific programming language. A probabilistic program implicitly defines a distribution on random variables, whilst the system back-end implements general-purpose inference methods.

PPS such as Infer.Net (Minka et al., 2010) and Stan (Carpenter et al., 2015) can be thought of as defining graphical models or factor graphs. Our focus will instead be on systems such as Church (Goodman et al., 2008), Venture (Mansinghka et al., 2014), WebPPL (Goodman and Stuhlmüller, 2014), and Anglican (Wood et al., 2014), which employ a general-purpose programming language for model specification. In these systems, the set of random variables is dynamically typed, such that it is possible to write programs in which this set

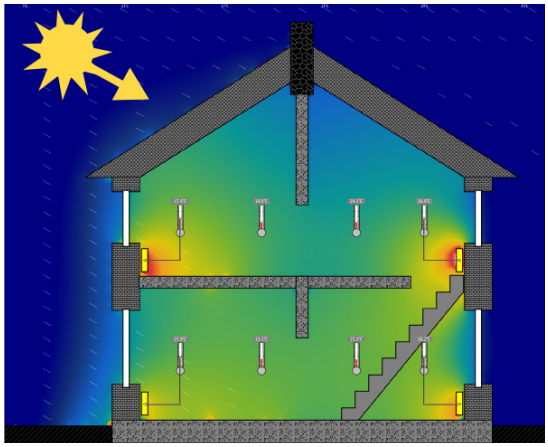
[1] Code available at `http://www.github.com/probprog/bopp/`



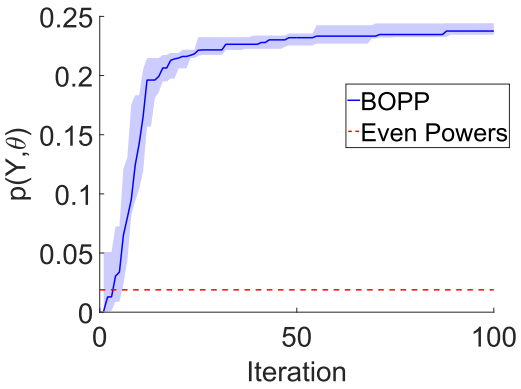
(a) Radiator powers set evenly



(b) Best setup from BOPP initialization



(c) Best setup after 100 iterations of BOPP




(d) Convergence of evidence

Figure 1: Simulation-based optimization of radiator powers subject to varying solar intensity. Shown are output heat maps from Energy2D (Xie, 2012) simulations at one intensity, corresponding to setting all the radiators to the same power (*top left*), the best result from a set of 5 randomly chosen powers used for initializing BOPP (*top right*), and the best setup found after 100 iterations of BOPP (*bottom left*). The bottom right plot shows convergence of the evidence of the respective model, giving the median and 25/75% quartiles.

```
(defopt house-heating [alphas target-temperatures] [powers]
 (let [solar-intensity (sample weather-prior)
       powers (sample (dirichlet alphas))
       temperatures (simulate solar-intensity powers)]
  (observe (abc-likelihood temperatures) target-temperatures)))
```

Figure 2: BOPP query for optimizing the power allocation to radiators in a house. Here `weather-prior` is a distribution over the solar intensity and a uniform Dirichlet prior with concentration `alpha` is placed over the powers. Calling **`simulate`** performs an Energy2D simulation of house temperatures. The utility of the resulting output is incorporated using **`abc-likelihood`**, which measures a discrepency from the `target-temperatures`. Calling **`doopt`** on this query invokes the BOPP algorithm to perform MMAP estimation, where the second input `powers` indicates the variable to be optimized.

differs from execution to execution. This allows an unspecified number of random variables and incorporation of arbitrary black box deterministic functions, such as was exploited by the `simulate` function in Figure 2. The price for this expressivity is that inference methods must be formulated in such a manner that they are applicable to models where the density function is intractable and can only be evaluated during forwards simulation of the program.

One such general purpose system, *Anglican*, will be used as a reference in this paper. In Anglican, models are defined using the inference macro `defquery`. These models, which we refer to as queries (Goodman et al., 2008), specify a joint distribution $p(Y, X)$ over data $Y$ and variables $X$. Inference on the model is performed using the macro `doquery`, which produces a sequence of approximate samples from the conditional distribution $p(X|Y)$ and, for importance sampling based inference algorithms (e.g. sequential Monte Carlo), a marginal likelihood estimate $p(Y)$.

Random variables in an Anglican program are specified using `sample` statements, which can be thought of as terms in the prior. Conditioning is specified using `observe` statements which can be thought of as likelihood terms. Outputs of the program, taking the form of posterior samples, are indicated by the return values. There is a finite set of `sample` and `observe` statements in a program source code, but the number of times each statement is called can vary between executions. We refer the reader to `http://www.robots.ox.ac.uk/~fwood/anglican/` for more details.

### 2.2 Bayesian Optimization

Consider an arbitrary black-box target function $f : \vartheta \to \mathbb{R}$ that can be evaluated for an arbitrary point $\theta \in \vartheta$ to produce, potentially noisy, outputs $\hat{w} \in \mathbb{R}$. BO (Jones et al., 1998; Osborne et al., 2009) aims to find the global maximum

$$\theta^* = \underset{\theta \in \vartheta}{\operatorname{argmax}}\, f(\theta). \tag{1}$$

The key idea of BO is to place a prior on $f$ that expresses belief about the space of functions within which $f$ might live. When the function is evaluated, the resultant information is incorporated by conditioning upon the observed data to give a posterior over functions. This allows estimation of the expected value and uncertainty in $f(\theta)$ for all $\theta \in \vartheta$. From this, an acquisition function $\zeta : \vartheta \to \mathbb{R}$ is defined, which assigns an expected utility to evaluating $f$ at particular $\theta$, based on the trade-off between exploration and exploitation in finding the maximum. When direct evaluation of $f$ is expensive, the acquisition function constitutes a cheaper to evaluate substitute, which is optimized to ascertain the next point at which the target function should be evaluated in a sequential fashion. By interleaving optimization of the acquisition function, evaluating $f$ at the suggested point, and updating the surrogate, BO forms a global optimization algorithm that is typically very efficient in the required number of function evaluations, whilst naturally dealing with noise in the outputs. Although alternatives such as random forests (Bergstra et al., 2011; Hutter et al., 2011) or neural networks (Snoek et al., 2015) exist, the most common prior used for $f$ is a GP (Rasmussen and Williams, 2006). For further information on BO we refer the reader to the recent review by Shahriari et al Shahriari et al. (2016b).

### 2.3 Gaussian Processes

Informally one can think of a Gaussian Process (GP) (Rasmussen and Williams, 2006) as being a nonparametric distribution over functions which is fully specified by a mean function $\mu : \vartheta \to \mathbb{R}$ and covariance function $k : \vartheta \times \vartheta \to \mathbb{R}$, the latter of which must be a bounded (i.e. $k(\theta, \theta') < \infty,\ \forall \theta, \theta' \in \vartheta$) and reproducing kernel. We can describe a function $f$ as being distributed according to a GP:

$$f(\theta) \sim GP(\mu(\theta), k(\theta, \theta')) \tag{2}$$

which by definition means that the functional evaluations realized at any finite number of sample points is distributed according to a multivariate Gaussian. Note that the inputs to $\mu$ and $k$ need not be numeric and as such a GP can be defined over anything for which kernel can be defined.

An important property of a GP is that it is conjugate with a Gaussian likelihood. Consider pairs of input-output data points $\{\hat{\theta}_j, \hat{w}_j\}_{j=1:m}$, $\hat{W} = \{\hat{w}_j\}_{j=1:m}$, $\hat{\Theta} = \{\hat{\theta}_j\}_{j=1:m}$ and the separable likelihood function

$$p(\hat{W}|\hat{\Theta}, f) = \prod_{j=1}^{m} p(\hat{w}_j|f(\hat{\theta}_j)) = \prod_{j=1}^{m} \frac{1}{\sigma_n\sqrt{2\pi}} \exp\left(-\frac{\left(\hat{w}_j - f(\hat{\theta}_j)\right)^2}{2\sigma_n^2}\right) \tag{3}$$

where $\sigma_n$ is an observation noise. Using a GP prior $f(\theta) \sim GP(\mu_{\text{prior}}(\theta), k_{\text{prior}}(\theta, \theta))$ leads to an analytic GP posterior

$$\mu_{post}(\theta) = \mu_{\text{prior}}(\theta) + k_{\text{prior}}\left(\theta, \hat{\Theta}\right)\left[k_{\text{prior}}\left(\hat{\Theta}, \hat{\Theta}\right) + \sigma_n^2 I\right]^{-1}\left(\hat{W} - \mu_{\text{prior}}\left(\hat{\Theta}\right)\right) \tag{4}$$

$$k_{post}(\theta, \theta') = k_{\text{prior}}(\theta, \theta') - k_{\text{prior}}\left(\theta, \hat{\Theta}\right)\left[k_{\text{prior}}\left(\hat{\Theta}, \hat{\Theta}\right) + \sigma_n^2 I\right]^{-1} k_{\text{prior}}\left(\hat{\Theta}, \theta'\right) \tag{5}$$

and Gaussian predictive distribution

$$w|\theta, \hat{W}, \hat{\Theta} \sim \mathcal{N}\left(\mu_{post}(\theta), k_{post}(\theta, \theta) + \sigma_n^2 I\right) \tag{6}$$

where we have used the shorthand $k_{\text{prior}}(\hat{\Theta}, \hat{\Theta}) = \begin{bmatrix} k_{\text{prior}}(\hat{\theta}_1, \hat{\theta}_1) & k_{\text{prior}}(\hat{\theta}_1, \hat{\theta}_2) & \dots \\ k_{\text{prior}}(\hat{\theta}_2, \hat{\theta}_1) & k_{\text{prior}}(\hat{\theta}_2, \hat{\theta}_2) & \dots \\ \dots & \dots & \dots \end{bmatrix}$ and similarly for $\mu_{\text{prior}}$, $\mu_{\text{post}}$ and $k_{\text{post}}$.

## 3. Problem Formulation

Given a program defining the joint density $p(Y, X, \theta)$ with fixed $Y$, our aim is to optimize with respect to a subset of the variables $\theta$ whilst marginalizing out latent variables $X$

$$\theta^* = \underset{\theta \in \vartheta}{\text{argmax}}\ p(\theta|Y) = \underset{\theta \in \vartheta}{\text{argmax}}\ p(Y, \theta) = \underset{\theta \in \vartheta}{\text{argmax}} \int p(Y, X, \theta) dX. \tag{7}$$

To provide syntax to differentiate between $\theta$ and $X$, we introduce a new query macro **`defopt`**. The syntax of **`defopt`** is identical to **`defquery`** except that it has an additional input identifying the variables to be optimized. To allow for the interleaving of inference and optimization required in MMAP estimation, we further introduce **`doopt`**, which, analogous to **`doquery`**, returns a lazy sequence $\{\hat{\theta}_m^*, \hat{\Omega}_m^*, \hat{u}_m^*\}_{m=1,\dots}$ where $\hat{\Omega}_m^* \subseteq X$ are the program outputs associated with $\theta = \hat{\theta}_m^*$ and each $\hat{u}_m^* \in \mathbb{R}^+$ is an estimate of the corresponding log marginal $\log p(Y, \hat{\theta}_m^*)$ (see Section 4.2). The sequence is defined such that, at any time, $\hat{\theta}_m^*$ corresponds to the point expected to be most optimal of those evaluated so far and allows both inference and optimization to be carried out online.

Although no restrictions are placed on $X$, it is necessary to place some restrictions on how programs use the optimization variables $\theta = \phi_{1:K}$ specified by the optimization argument list of **`defopt`**. First, each optimization variable $\phi_k$ must be bound to a value directly by a **`sample`** statement with fixed measure-type distribution argument. This avoids change of variable complications arising from nonlinear deterministic mappings. Second, in order for the optimization to be well defined, the program must be written such that any possible execution trace binds each optimization variable $\phi_k$ exactly once. Finally, although any $\phi_k$ may be lexically multiply bound, it must have the same base measure in all possible execution traces, because, for instance, if the base measure of a $\phi_k$ were to change from Lebesgue to counting, the notion of optimality would no longer admit a conventional interpretation. Note that although the transformation implementations shown in Figure 3 do not contain runtime exception generators that disallow continued execution of programs that violate these constraints, those actually implemented in the BOPP system do.

## 4. Bayesian Program Optimization

In addition to the syntax introduced in the previous section, there are five main components to BOPP:

- A program transformation, `q→q-marg`, allowing estimation of the evidence $p(Y, \theta)$ at a fixed $\theta$.
- A high-performance, GP based, BO implementation for actively sampling $\theta$.
- A program transformation, `q→q-prior`, used for automatic and adaptive domain scaling, such that a problem-independent hyperprior can be placed over the GP hyperparameters.
- An adaptive non-stationary mean function to support unbounded optimization.
- A program transformation, `q→q-acq`, and annealing maximum likelihood estimation method to optimize the acquisition function subject the implicit constraints imposed by the generative model.

Together these allow BOPP to perform online MMAP estimation for arbitrary programs in a manner that is black-box from the user's perspective - requiring only the definition of the target program in the same way as existing PPS and identifying which variables to optimize. The BO component of BOPP is both probabilistic programming and language independent, and is provided as a stand-alone package.[2] It requires as input only a target function, a sampler to establish rough input scaling, and a problem specific optimizer for the acquisition function that imposes the problem constraints.

Figure 3 provides a high level overview of the algorithm invoked when **`doopt`** is called on a query `q` that defines a distribution $p(Y, a, \theta, b)$. We wish to optimize $\theta$ whilst marginalizing out $a$ and $b$, as indicated by the the second input to `q`. In summary, BOPP performs iterative optimization in 5 steps

- Step 1 (blue arrows) generates unweighted samples from the transformed prior program `q-prior` (*top center*), constructed by removing all conditioning. This initializes the domain scaling for $\theta$.
- Step 2 (red arrows) evaluates the marginal $p(Y, \theta)$ at a small number of the generated $\hat{\theta}$ by performing inference on the marginal program `q-marg` (*middle centre*), which returns samples from the distribution $p(a, b|Y, \theta)$ along with an estimate of $p(Y, \theta)$. The evaluated points (*middle right*) provide an initial domain scaling of the outputs and starting points for the BO surrogate.
- Step 3 (black arrow) fits a mixture of GPs posterior Rasmussen and Williams (2006) to the scaled data (*bottom centre*) using a problem independent hyperprior. The solid blue line and shaded area show the posterior mean and $\pm 2$ standard deviations respectively. The new estimate of the optimum $\hat{\theta}^*$ is the value for which the mean estimate is largest, with $\hat{u}^*$ equal to the corresponding mean value.
- Step 4 (purple arrows) constructs an acquisition function $\zeta \colon \vartheta \to \mathbb{R}^+$ (*bottom left*) using the GP posterior. This is optimized, giving the next point to evaluate $\hat{\theta}_{\text{next}}$, by performing annealed importance sampling on a transformed program `q-acq` (*middle left*) in which all **`observe`** statements are removed and replaced with a single **`observe`** assigning probability $\zeta(\theta)$ to the execution.
- Step 5 (green arrow) evaluates $\hat{\theta}_{\text{next}}$ using `q-marg` and continues to step 3.

### 4.1 Program Transformation to Generate the Target

Consider the **`defopt`** query `q` in Figure 3, the body of which defines the joint distribution $p(Y, a, \theta, b)$. Calculating (7) (defining $X = \{a, b\}$) using a standard optimization scheme presents two issues: $\theta$ is a random variable within the program rather than something we control and its probability distribution is only defined conditioned on $a$.

We deal with both these issues simultaneously using a program transformation similar to the disintegration transformation in Hakaru (Zinkov and Shan, 2016). Our *marginal* transformation returns a new **`query`** object, `q-marg` as shown in Figure 3, that defines the same joint distribution on program variables and inputs, but

[2] Code available at `http://www.github.com/probprog/deodorant/`

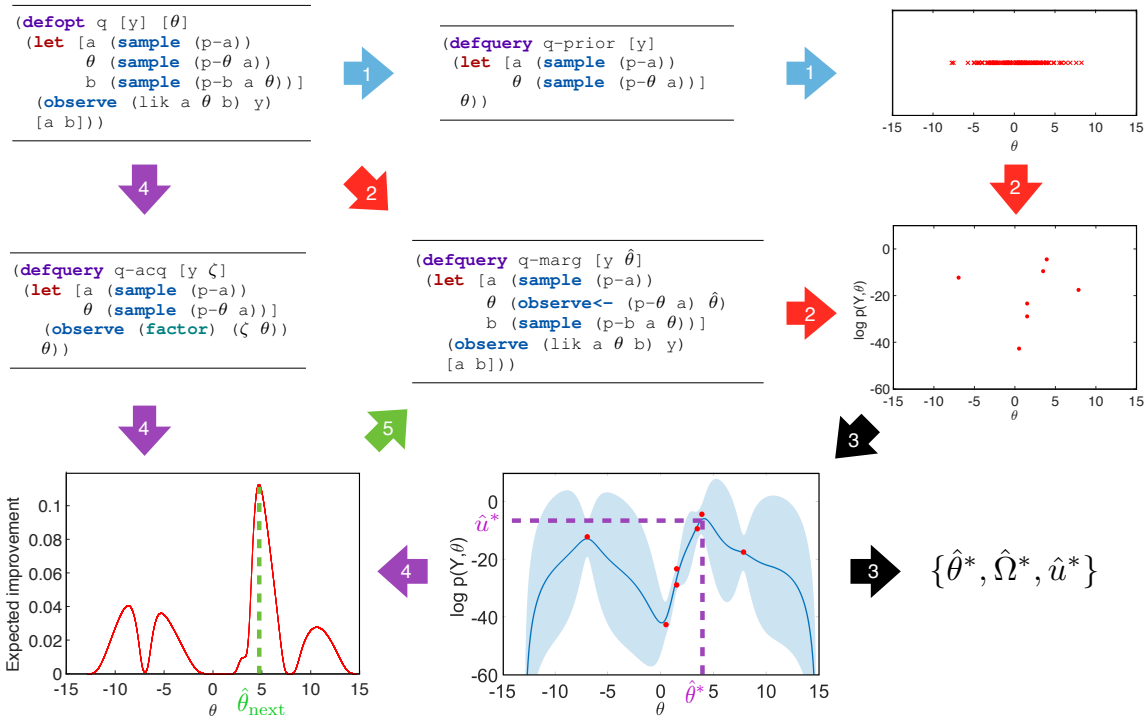



Figure 3: Overview of the BOPP algorithm, description given in main text. `p-a`, `p-θ`, `p-b` and `lik` all represent distribution object constructors. **`factor`** is a special distribution constructor that assigns probability $p(y) = y$, in this case $y = \zeta(\theta)$.

now accepts the value for $\theta$ as an input. This is done by replacing all **`sample`** statements associated with $\theta$ with equivalent **`observe<-`** statements, taking $\theta$ as the observed value, where **`observe<-`** is identical to **`observe`** except that it returns the observed value. As both **`sample`** and **`observe`** operate on the same variable type - a distribution object - this transformation can always be made, while the identical returns of **`sample`** and **`observe<-`** trivially ensures validity of the transformed program.

### 4.2 Bayesian Optimization of the Marginal

The target function for our BO scheme is $\log p(Y, \theta)$, noting $\operatorname{argmax} f(\theta) = \operatorname{argmax} \log f(\theta)$ for any $f : \vartheta \to \mathbb{R}^+$. The log is taken because GPs have unbounded support, while $p(Y, \theta)$ is always positive, and because we expect variations over many orders of magnitude. PPS with importance sampling based inference engines, e.g. sequential Monte Carlo (Wood et al., 2014) or the particle cascade (Paige et al., 2014), can return noisy estimates of this target given the transformed program `q-marg`.

Our BO scheme uses a GP prior and a Gaussian likelihood. Though the rationale for the latter is predominantly computational, giving an analytic posterior, there are also theoretical results suggesting that this choice is appropriate (Bérard et al., 2014). We use as a default covariance function a combination of a Matérn-3/2 and Matérn-5/2 kernel. Specifically, let $D = \|\theta\|_0$ be the dimensionality of $\theta$ and define

$$d_{3/2}(\theta, \theta') = \sqrt{\sum_{i=1}^{D} \frac{\theta_i - \theta'_i}{\rho_i}} \tag{8a}$$

$$d_{5/2}(\theta,\theta') = \sqrt{\sum_{i=1}^{D} \frac{\theta_i - \theta'_i}{\varrho_i}} \tag{8b}$$

where $i$ indexes a dimension of $\theta$ and $\rho_i$ and $\varrho_i$ are dimension specific length scale hyperparameters. Our prior covariance function is now given by

$$\begin{aligned} k_{\text{prior}}(\theta,\theta') =& \sigma_{3/2}^2 \left(1+\sqrt{3}d_{3/2}(\theta,\theta')\right)\exp\left(-\sqrt{3}d_{3/2}(\theta,\theta')\right) + \\ & \sigma_{5/2}^2 \left(1+\sqrt{5}d_{5/2}(\theta,\theta') + \frac{5}{3}(d_{5/2}(\theta,\theta'))^2\right)\exp\left(-\sqrt{5}d_{5/2}(\theta,\theta')\right) \end{aligned} \tag{9}$$

where $\sigma_{3/2}$ and $\sigma_{5/2}$ represent signal standard deviations for the two respective kernels. The full set of GP hyperparameters is defined by $\alpha = \{\sigma_n, \sigma_{3/2}, \sigma_{5/2}, \rho_{i=1:D}, \varrho_{i=1:D}\}$. A key feature of this kernel is that it is only once differentiable and therefore makes relatively weak assumptions about the smoothness of $f$. The ability to include branching in a probabilistic program means that, in some cases, an even less smooth kernel than (9) might be preferable. However, there is clear a trade-off between generality of the associated reproducing kernel Hilbert space and modelling power.

As noted by (Snoek et al., 2012), the performance of BO using a single GP posterior is heavily influenced by the choice of these hyperparameters. We therefore exploit the automated domain scaling introduced in Section 4.3 to define a problem independent hyperprior $p(\alpha)$ and perform inference to give a mixture of GPs posterior. Details on this hyperprior are given in Appendix B.

Inference over $\alpha$ is performed using Hamiltonian Monte Carlo (HMC) (Duane et al., 1987), giving an unweighted mixture of GPs. Each term in this mixture has an analytic distribution fully specified by its mean function $\mu_m^i \colon \vartheta \to \mathbb{R}$ and covariance function $k_m^i \colon \vartheta \times \vartheta \to \mathbb{R}$, where $m$ indexes the BO iteration and $i$ the hyperparameter sample. HMC was chosen because of the availability of analytic derivatives of the GP log marginal likelihoods. As we found that the performance of HMC was often poor unless a good initialization point was used, BOPP runs a small number of independent chains and allocates part of the computational budget to their initialization using a L-BFGS optimizer (Broyden, 1970).

The inferred posterior is first used to estimate which of the previously evaluated $\hat{\theta}_j$ is the most optimal, by taking the point with highest expected value , $\hat{u}_m^* = \max_{j \in 1\ldots m} \sum_{i=1}^{N} \mu_m^i(\hat{\theta}_j)$. This completes the definition of the output sequence returned by the **`doopt`** macro. Note that as the posterior updates globally with each new observation, the relative estimated optimality of previously evaluated points changes at each iteration. Secondly it is used to define the acquisition function $\zeta$, for which we take the expected improvement (Snoek et al., 2012), defining $\sigma_m^i(\theta) = \sqrt{k_m^i(\theta,\theta)}$ and $\gamma_m^i(\theta) = \frac{\mu_m^i(\theta) - \hat{u}_m^*}{\sigma_m^i(\theta)}$,

$$\zeta(\theta) = \sum_{i=1}^{N} \left(\mu_m^i(\theta) - \hat{u}_m^*\right)\Phi\left(\gamma_m^i(\theta)\right) + \sigma_m^i(\theta)\,\phi\left(\gamma_m^i(\theta)\right) \tag{10}$$

where $\phi$ and $\Phi$ represent the pdf and cdf of a unit normal distribution respectively. We note that more powerful, but more involved, acquisition functions, e.g. (Hernández-Lobato et al., 2014), could be used instead.

### 4.3 Automatic and Adaptive Domain Scaling

Domain scaling, by mapping to a common space, is crucial for BOPP to operate in the required black-box fashion as it allows a general purpose and problem independent hyperprior to be placed on the GP hyperparameters. BOPP therefore employs an affine scaling to a $[-1,1]$ hypercube for both the inputs and outputs of the GP. To initialize scaling for the input variables, we sample directly from the generative model defined by the program. This is achieved using a second transformed program, `q-prior`, which removes all conditioning, i.e. **`observe`** statements, and returns $\theta$. This transformation also introduces code to terminate execution of the query once all $\theta$ are sampled, in order to avoid unnecessary computation. As **`observe`** statements return `nil`, this transformation trivially preserves the generative model of the program, but the

probability of the execution changes. Simulating from the generative model does not require inference or calling potentially expensive likelihood functions and is therefore computationally inexpensive. By running inference on `q-marg` given a small number of these samples as arguments, a rough initial characterization of output scaling can also be achieved. If points are observed that fall outside the hypercube under the initial scaling, the domain scaling is appropriately updated[3] so that the target for the GP remains the $[-1, 1]$ hypercube.

### 4.4 Unbounded Bayesian Optimization via Non-Stationary Mean Function Adaptation

Unlike standard BO implementations, BOPP is not provided with external constraints and we therefore develop a scheme for operating on targets with potentially unbounded support. Our method exploits the knowledge that the target function is a probability density, implying that the area that must be searched in practice to find the optimum is finite, by defining a non-stationary prior mean function. This takes the form of a bump function that is constant within a region of interest, but decays rapidly outside. Specifically we define this bump function in the transformed space as

$$\mu_{\text{prior}}\left(r; r_e, r_\infty\right) = \begin{cases} 0 & \text{if } r \leq r_{\text{e}} \\ \log\left(\frac{r-r_{\text{e}}}{r_\infty - r_{\text{e}}}\right) + \frac{r-r_{\text{e}}}{r_\infty - r_{\text{e}}} & \text{otherwise} \end{cases} \tag{11}$$

where $r$ is the radius from the origin, $r_e$ is the maximum radius of any point generated in the initial scaling or subsequent evaluations, and $r_\infty$ is a parameter set to $1.5r_e$ by default. Consequently, the acquisition function also decays and new points are never suggested arbitrarily far away. Adaptation of the scaling will automatically update this mean function appropriately, learning a region of interest that matches that of the true problem, without complicating the optimization by over-extending this region. We note that our method shares similarity with the recent work of Shahriari et al (Shahriari et al., 2016a), but overcomes the sensitivity of their method upon a user-specified bounding box representing soft constraints, by initializing automatically and adapting as more data is observed.

### 4.5 Optimizing the Acquisition Function

Optimizing the acquisition function for BOPP presents the issue that the query contains implicit constraints that are unknown to the surrogate function. The problem of unknown constraints has been previously covered in the literature (Gardner et al., 2014; Hernández-Lobato et al., 2016) by assuming that constraints take the form of a black-box function which is modeled with a second surrogate function and must be evaluated in guess-and-check strategy to establish whether a point is valid. Along with the potentially significant expense such a method incurs, this approach is inappropriate for *equality* constraints or when the target variables are potentially discrete. For example, the Dirichlet distribution in Figure 2 introduces an equality constraint on `powers`, namely that its components must sum to 1.

We therefore take an alternative approach based on directly using the program to optimize the acquisition function. To do so we consider a transformed program `q-acq` that is identical to `q-prior` (see Section 4.3), but adds an additional **`observe`** statement that assigns a weight $\zeta(\theta)$ to the execution. By setting $\zeta(\theta)$ to the acquisition function, the maximum likelihood corresponds to the optimum of the acquisition function subject to the implicit program constraints. We obtain a maximum likelihood estimate for `q-acq` using a variant of annealed importance sampling (Neal, 2001) in which lightweight Metropolis Hastings (LMH) (Wingate et al., 2011) with local random-walk moves is used as the base transition kernel.

## 5. Experiments

We first demonstrate the ability of BOPP to carry out unbounded optimization using a 1D problem with a significant prior-posterior mismatch as shown in Figure 4. It shows BOPP adapting to the target and effectively

[3] An important exception is that the output mapping to the bottom of the hypercube remains fixed such that low likelihood new points are not incorporated. This ensures stability when considering unbounded problems.

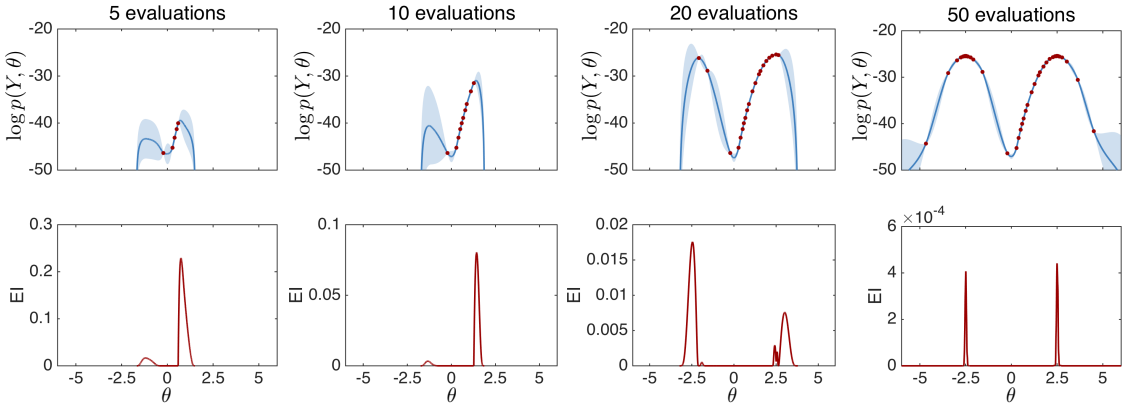



Figure 4: Convergence on an unconstrained bimodal problem with $p(\theta) = \text{Normal}(0, 0.5)$ and $p(Y|\theta) = \text{Normal}(5 - |\theta|, 0.5)$ giving significant prior misspecification. The top plots show a regressed GP, with the solid line corresponding to the mean and the shading shows $\pm$ 2 standard deviations. The bottom plots show the corresponding acquisition functions.

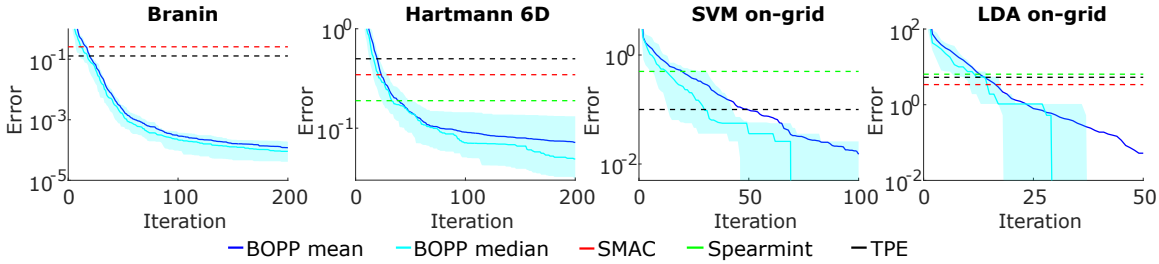



Figure 5: Comparison of BOPP used as an optimizer to prominent BO packages on common benchmark problems. The dashed lines shows the final mean error of SMAC (red), Spearmint (green) and TPE (black) as quoted by Eggensperger et al. (2013). The dark blue line shows the mean error for BOPP averaged over 100 runs, whilst the median and 25/75% percentiles are shown in cyan. Results for Spearmint on Branin and SMAC on SVM on-grid are omitted because both BOPP and the respective algorithms averaged zero error to the provided number of significant figures in Eggensperger et al. (2013).

establishing a maxima in the presence of multiple modes. After 20 evaluations the acquisitions begin to explore the right mode, after 50 both modes have been fully uncovered.

### 5.1 Classic Optimization Benchmarks

Next we compare BOPP to the prominent BO packages SMAC Hutter et al. (2011), Spearmint Snoek et al. (2012) and TPE Bergstra et al. (2011) on a number of classical benchmarks as shown in Figure 5. These results demonstrate that BOPP provides substantial advantages over these systems when used simply as an optimizer on both continuous and discrete optimization problems. In particular, it offers a large advantage over SMAC and TPE on the continuous problems (Branin and Hartmann), due to using a more powerful surrogate, and over Spearmint on the others due to not needing to make approximations to deal with discrete problems.

### 5.2 Marginal Maximum a Posteriori Estimation Problems

We now demonstrate application of BOPP on a number of MMAP problems. Comparisons here are more difficult due to the dearth of existing alternatives for PPS. In particular, simply running inference on the original

```
(defopt mvn-mixture [data mu0 kappa psi] [nu alpha]
 (let [[n d] (shape data)
       alpha (sample (uniform-continuous 0.01 100))
       nu (sample (uniform-continuous (- d 1) 100))
       obs-proc0 (mvn-niw mu0 kappa nu psi)]
       (loop [data data
              obs-procs {}
              mix-proc (dirichlet-discrete
                         (vec (repeat d alpha)))]
            (let [y (first data)]
             (if y
              (let [z (sample (produce comp-proc))
                    obs-proc (get obs-procs z obs-proc0)
                    obs-dist (produce obs-proc)]
                (observe obs-dist y)
                (recur (rest data)
                       (assoc obs-procs z (absorb obs-proc y))
                (absorb mix-proc z)))
              mix-proc)))))
```

Figure 6: Anglican query for hyperparameter optimization of a Gaussian mixture model, defined in terms of two parameters `nu` and `alpha`. A **`mvn-niw`** process is used to represent the marginal likelihood of observations under a Gaussian-inverse-Wishart prior, whereas a **`dirichlet-discrete`** process models the prior probability of cluster assignments under a Dirichlet-discrete prior. The command **`produce`** returns the predictive distribution for the next sample from a process. **`absorb`** conditions on the value of the next sample.

query does not return estimates for $p(Y, \theta)$. We consider the possible alternative of using our conditional code transformation to design a particle marginal Metropolis Hastings (PMMH, Andrieu et al. (2010)) sampler which operates in a similar fashion to BOPP except that new $\theta$ are chosen using a MH step instead of actively sampling with BO. For these MH steps we consider both LMH (Wingate et al., 2011) with proposals from the prior and the random-walk MH (RMH) variant introduced in Section 4.5.

### 5.2.1 HYPERPARAMETER OPTIMIZATION FOR GAUSSIAN MIXTURE MODEL

We start with an illustrative case study of optimizing the hyperparameters in a multivariate Gaussian mixture model. We consider a Bayesian formulation with a symmetric Dirichlet prior on the mixture weights and a Gaussian-inverse-Wishart prior on the likelihood parameters:

$$\boldsymbol{\pi} \sim \mathrm{Dir}(\alpha, \ldots, \alpha) \tag{12}$$

$$(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \sim \mathrm{NIW}(\boldsymbol{\mu}_0, \kappa, \boldsymbol{\Psi}, \nu) \qquad \text{for } k = 1, \ldots, K \tag{13}$$

$$z_n \sim \mathrm{Disc}(\boldsymbol{\pi}) \tag{14}$$

$$\boldsymbol{y}_n \sim \mathrm{Norm}(\boldsymbol{\mu}_{z_n}, \boldsymbol{\Sigma}_{z_n}) \qquad \text{for } n = 1, \ldots, N \tag{15}$$

Anglican code for this model is shown in Figure 4. Anglican provides stateful objects, which are referred to as random processes, to represent the predictive distributions for the cluster assignments $z$ and the observations $\boldsymbol{y}^k$ assigned to each cluster

$$z_{n+1} \sim p(\cdot \mid z_{1:n}, \alpha), \tag{16}$$

$$\boldsymbol{y}^k_{m+1} \sim p(\cdot \mid \boldsymbol{y}^k_{1:m}, \boldsymbol{\mu}_0, \kappa, \boldsymbol{\Psi}, \nu). \tag{17}$$

In this collapsed representation marginalization over the model parameters $\boldsymbol{\pi}$, $\boldsymbol{\mu}_{k=1:K}$, and $\boldsymbol{\Sigma}_{k=1:K}$ is performed analytically. Using the Iris dataset, a standard benchmark for mixture models that contains 150 labeled examples with 4 real-valued features, we optimize the marginal with respect to the subset of the

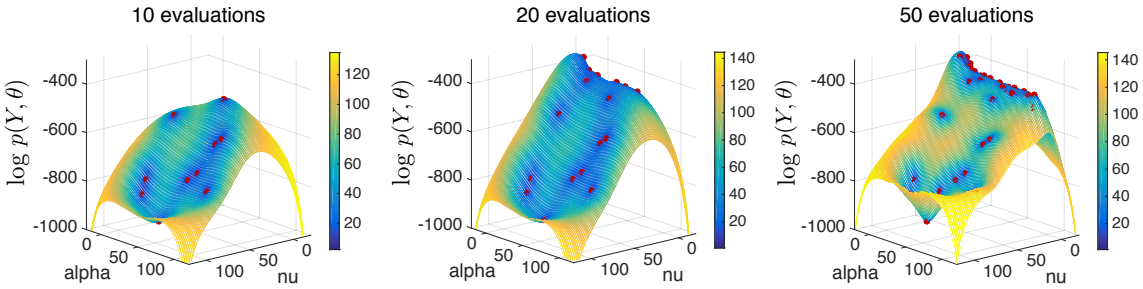



Figure 7: Bayesian optimization of hyperparameters in a Gaussian mixture model evaluated on the Iris dataset. Panels show the GP posterior as a function of number of evaluations, with the surface corresponding to the posterior mean and the color bars the posterior standard deviation. Optimization is performed over the parameter $\alpha$ of a 10-dimensional symmetric Dirichlet distribution and the degrees of freedom $\nu$ of the inverse-Wishart prior. At each evaluation we obtain an estimate of the log marginal $\log p(Y, \theta)$ obtained by performing sequential Monte Carlo inference with 1000 particles. The apparent maximum after initialization with 10 randomly sampled points lies at $\nu = 31$, $\alpha = 60$, and $\log p(Y, \theta) = -456.3$ (*left*). The surface after 10 optimization steps shows a new maximum at $\nu = 9.2$, $\alpha = 0.8$, and $\log p(Y, \theta) = -364.2$ (*middle*). After 40 steps and 50 total evaluations this optimum is refined to $\nu = 16$, $\alpha = 0.2$, and $\log p(Y, \theta) = -352.5$ (*right*).

parameters $\nu$ and $\alpha$ under uniform priors over a fixed interval. For this model, BOPP aims to maximize

$$p(\nu, \alpha | \boldsymbol{y}_{n=1:N}, \boldsymbol{\mu}_0, \kappa, \boldsymbol{\Psi}) = \iiiint p(\nu, \alpha, z_{n=1:N}, \boldsymbol{\pi}, \boldsymbol{\mu}_{k=1:K}, \boldsymbol{\Sigma}_{k=1:K} | \boldsymbol{y}_{n=1:N}, \mu_0, \kappa, \boldsymbol{\Psi}) \mathrm{d}z_{n=1:N} \mathrm{d}\boldsymbol{\pi} \mathrm{d}\boldsymbol{\mu}_{k=1:K} \mathrm{d}\boldsymbol{\Sigma}_{k=1:K}. \tag{18}$$

Figure 7 shows GP regressions on the evidence after different numbers of the SMC evaluations have been performed on the model. This demonstrates how the GP surrogate used by BO builds up a model of the target, used to both estimate the expected value of $\log p(Y, \theta)$ for a particular $\theta$ and actively sample the $\theta$ at which to undertake inference.

### 5.2.2 EXTENDED KALMAN FILTER FOR THE PICKOVER CHAOTIC ATTRACTOR

We next consider the case of learning the dynamics parameters of a chaotic attractor. Chaotic attractors present an interesting case for tracking problems as, although their underlying dynamics are strictly deterministic with bounded trajectories, neighbouring trajectories diverge exponentially[4]. Therefore regardless of the available precision, a trajectory cannot be indefinitely extrapolated to within a given accuracy and probabilistic methods such as the extended Kalman filter must be incorporated (Fujii, 2013; Ruan et al., 2003). From an empirical perspective, this forms a challenging optimization problem as the target transpires to be multi-modal, has variations at different length scales, and has local minima close to the global maximum.

Suppose we observe a noisy signal $y_t \in \mathbb{R}^K,\ t = 1, 2, \ldots, T$ in some $K$ dimensional observation space were each observation has a lower dimensional latent parameter $x_t \in \mathbb{R}^D,\ t = 1, 2, \ldots, T$ whose dynamics correspond to a chaotic attractor of known type, but with unknown parameters. Our aim will be to find the MMAP values for the dynamics parameters $\theta$, marginalizing out the latent states. The established parameters can then be used for forward simulation or tracking.

[4]It is beyond the scope of this paper to properly introduce chaotic systems. We refer the reader to Devaney et al. (1989) for an introduction.

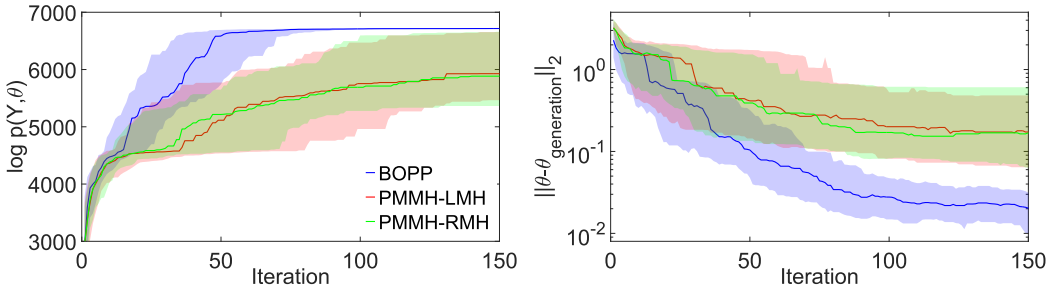
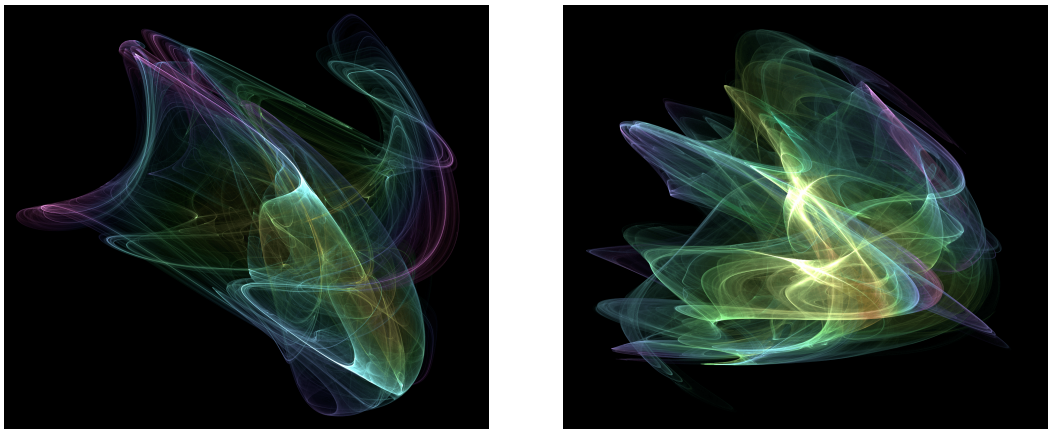



Figure 8: Convergence for transition dynamics parameters of the pickover attractor in terms of the cumulative best $\log p\left(Y, \theta\right)$ (*left*) and distance to the "true" $\theta$ used in generating the data (*right*). Solid line shows median over 100 runs, whilst the shaded region the 25/75% quantiles.



(a) 1 iteration, $\theta = [-1.478, 0.855]^T$

(b) 20 iterations, $\theta = [-2.942, 1.550]^T$

(c) 100 iterations, $\theta = [-2.306, 1.249]^T$

(d) Ground truth, $\theta = [-2.3, 1.25]^T$

Figure 9: A series of trajectories for different parameters, demonstrating convergence to the true attractor. The colormap is based on the speed and curvature of the trajectory, with rendering done using the program Chaoscope (available at http://www.chaoscope.org/).

To carry out the required MMAP estimation, we apply BOPP to the extended Kalman smoother

$$x_1 \sim \mathcal{N}(\mu_1, \sigma_1 I) \tag{19}$$
$$x_t = A(x_{t-1}, \theta) + \delta_{t-1}, \qquad \delta_{t-1} \sim \mathcal{N}(0, \sigma_q I) \tag{20}$$
$$y_t = C x_t + \varepsilon_t, \qquad \varepsilon_t \sim \mathcal{N}(0, \sigma_y I) \tag{21}$$

where $I$ is the identity matrix, $C$ is a known $K \times D$ matrix, $\mu_1$ is the expected starting position, and $\sigma_1, \sigma_q$ and $\sigma_y$ are all scalars which are assumed to be known. The transition function $A(\cdot, \cdot)$

$$x_{t,1} = \sin(\beta x_{t-1,2}) - \cos\left(\frac{5x_{t-1,1}}{2}\right) x_{t-1,3} \tag{22a}$$
$$x_{t,2} = -\sin\left(\frac{3x_{t-1,1}}{2}\right) x_{t-1,3} - \cos(\eta x_{t-1,2}) \tag{22b}$$
$$x_{t,3} = \sin(x_{t-1,1}) \tag{22c}$$

corresponds to a Pickover attractor (Pickover, 1995) with unknown parameters $\theta = \{\beta, \eta\}$ which we wish to optimize. Note that $\eta$ and $-\eta$ will give the same behaviour.

Synthetic data was generated for 500 time steps using the parameters of $\mu_1 = [-0.2149, -0.0177, 0.7630]^T$, $\sigma_1 = 0$, $\sigma_q = 0.01$, $\sigma_y = 0.2$, a fixed matrix $C$ where $K = 20$ and each column was randomly drawn from a symmetric Dirichlet distribution with parameter 0.1, and ground truth transition parameters of $\beta = -2.3$ and $\eta = 1.25$ (note that the true global optimum for finite data need not be exactly equal to this).

MMAP estimation was performed on this data using the same model and parameters, with the exceptions of $\theta$, $\mu_1$ and $\sigma_1$. The prior on $\theta$ was set to a uniform in over a bounded region such that

$$p(\beta, \eta) = \begin{cases} 1/18, & \text{if } -3 \leq \beta \leq 3 \ \cap \ 0 \leq \eta \leq 3 \\ 0, & \text{otherwise} \end{cases}. \tag{23}$$

The changes $\mu_1 = [0, 0, 0]$ and $\sigma_1 = 1$ were further made to reflect the starting point of the latent state being unknown. For this problem, BOPP aims to maximize

$$p(\beta, \eta | y_{t=1:T}) = \int p(\beta, \eta, x_{t=1:T} | y_{t=1:T}) \mathrm{d}x_{t=1:T}. \tag{24}$$

Inference on the transformed marginal query was carried out using SMC with 500 particles. Convergence results are given in Figure 8 showing that BOPP comfortably outperforms the PMMH variants, while Figure 9 shows the simulated attractors generated from the dynamics parameters output by various iterations of a particular run of BOPP.

#### 5.2.3 HIDDEN MARKOV MODEL WITH UNKNOWN NUMBER OF STATES

We finally consider a hidden Markov model (HMM) with an unknown number of states. This example demonstrates how BOPP can be applied to models which conceptually have an unknown number of variables, by generating all possible variables that might be needed, but then leaving some variables unused for some execution traces. This avoids problems of varying base measures so that the MMAP problem is well defined and provides a function with a fixed number of inputs as required by the BO scheme. From the BO perspective, the target function is simply constant for variations in an unused variable.

HMMs are Markovian state space models with discrete latent variables. Each latent state $x_t \in \{1, \ldots, K\}, t = 1, \ldots, T$ is defined conditionally on $x_{t-1}$ through a set of discrete transition probabilities, whilst each output $y_t \in \mathbb{R}$ is considered to be generated i.i.d. given $x_t$. We consider the following HMM, in which the number of states $K$, is also a random variable:

$$K \sim \text{Discrete}\{1, 2, 3, 4, 5\} \tag{25}$$
$$T_k \sim \text{Dirichlet}\{1_{1:K}\}, \quad \forall k = 1, \ldots, K \tag{26}$$

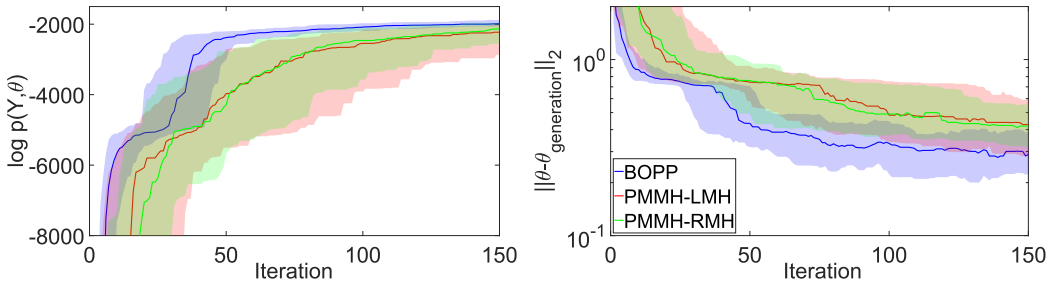



Figure 10: Convergence for HMM in terms of the cumulative best $\log p\,(Y, \theta)$ (*left*) and distance to the "true" $\theta$ used in generating the data (*right*). Solid line shows median over 100 runs, whilst the shaded region the 25/75% quantiles. Note that for the distance to true $\theta$ was calculated by selecting which three states (out of the 5 generates) that were closest to the true parameters.

$$\phi_k \sim \text{Uniform}[0,1], \quad \forall k = 1,\ldots,K \tag{27}$$
$$\mu_0 \leftarrow \min\{y_{1:T}\} \tag{28}$$
$$\mu_k \leftarrow \mu_{k-1} + \phi_k \cdot (\max\{y_{1:T}\} - \mu_{k-1}), \quad \forall k = 1,\ldots,K \tag{29}$$
$$x_1 \leftarrow 1 \tag{30}$$
$$x_t|x_{t-1} \sim \text{Discrete}\{T_{x_{t-1}}\} \tag{31}$$
$$y_t|x_t \sim \mathcal{N}(\mu(x_{t-1}), 0.2). \tag{32}$$

Our experiment is based on applying BOPP to the above model to do MMAP estimation with a single synthetic dataset, generated using $K = 3,\ \mu_1 = -1,\ \mu_2 = 0,\ \mu_3 = 4,\ T_1 = [0.9, 0.1, 0],\ T_2 = [0.2, 0.75, 0.05]$ and $T_3 = [0.1, 0.2, 0.7]$.

We use BOPP to optimize both the number of states $K$ and the stick-breaking parameters $\phi_k$, with full inference performed on the other parameters. BOPP therefore aims to maximize

$$p(K, \phi_{k=1:5}|y_{t=1:T}) = \iint p(K, \phi_{k=1:5}, x_{t=1:T}, T_{k=1:K}|y_{t=1:T})\mathrm{d}x_{t=1:T}\mathrm{d}T_{k=1:K}. \tag{33}$$

As with the chaotic Kalman filter example, we compare to two PMMH variants using the same code transformations. The results, given in Figure 10, again show that BOPP outperforms these PMMH alternatives.

## 6. Discussion and Future Work

We have introduced a new method for carrying out MMAP estimation of probabilistic program variables using Bayesian optimization, representing the first unified framework for optimization and inference of probabilistic programs. By using a series of code transformations, our method allows an arbitrary program to be optimized with respect to a defined subset of its variables, whilst marginalizing out the rest. To carry out the required optimization, we introduce a new GP-based BO package that exploits the availability of the target source code to provide a number of novel features, such as automatic domain scaling and constraint satisfaction.

The concepts we introduce lead directly to a number of extensions of interest, including but not restricted to smart initialization of inference algorithms, adaptive proposals, and nested optimization. Further work might consider maximum marginal likelihood estimation and risk minimization. Though only requiring minor algorithmic changes, these cases require distinct theoretical considerations.

## Appendix A. Program Transformations in Detail

In this section we give a more detailed and language specific description of our program transformations, code for which can be found at `http://www.github.com/probprog/bopp`.

### A.1 Anglican

Anglican is a probabilistic programming language integrated into Clojure (a dialect of Lisp) and inherits most of the corresponding syntax. Anglican extends Clojure with the special forms **`sample`** and **`observe`** (Tolpin et al., 2015). Each random draw in an Anglican program corresponds to a **`sample`** call, which can be thought of as a term in the prior. Each **`observe`** statement applies weighting to a program trace and thus constitutes a term in the likelihood. Compilation of an Anglican program, performed by the macro **`query`**, corresponds to transforming the code into a variant of continuation-passing style (CPS) code, which results in a function that can be executed using a particular inference algorithm.

Anglican program code is represented by a nested list of expressions, symbols, non-literals for contructing data structures (e.g. `[...]` for vectors), and command dependent literals (e.g. `[...]` as a second argument of a **`let`** statement which is used for binding pairs). In order to perform program transformations, we can recursively traverse this nested list which can be thought of as an abstract syntax tree of the program.

Our program transformations also make use of the Anglican forms **`store`** and **`retrieve`**. These allow storing any variable in the probabilistic program's execution trace in a state which is passed around during execution and from which we can retrieve these stored values. The core use for this is to allow the outer query to return variables which are only locally scoped.

To allow for the early termination that will be introduced in Section A.5, it was necessary to add a mechanism for non-local returns to Anglican. Clojure supports non-local returns only through Java exception handling, via the keywords **`try`** **`throw`**, **`catch`** and **`finally`**. Unfortunately, these are not currently supported by Anglican and their behaviour is far from ideal for our purposes. In particular, for programs containing nested **`try`** statements, throwing to a particular **`try`** in the stack, as opposed to the most recently invoked, is cumbersome and error prone.

We have instead, therefore, added to Anglican a non-local return mechanism based on the Common Lisp control form **`catch`**/**`throw`**. This uses a *catch tag* to link each **`throw`** to a particular **`catch`**. For example

```
(catch :tag
  (when (> a 0)
    (throw :tag a))
  0)
```

is equivalent to `(max a 0)`. More precisely, **`throw`** has syntax `(throw tag value)` and will cause the **`catch`** block with the corresponding `tag` to exit, returning `value`. If a **`throw`** goes uncaught, i.e. it is not contained within a **`catch`** block with a matching tag, a custom Clojure exception is thrown.

### A.2 Representations in the Main Paper

In the main paper we presented the code transformations as static transformations as shown in Figure 3. Although for simple programs, such as the given example, these transformations can be easily expressed as static transformations, for more complicated programs it would be difficult to actually implement these as purely static generic transformations in a higher-order language. Therefore, even though all the transformations dynamically execute as shown at runtime, in truth, the generated source code for the prior and acquisition transformations varies from what is shown and has been presented this way in the interest of exposition. Our true transformations exploit **`store`**, **`retrieve`**, **`catch`** and **`throw`** to generate programs that dynamically execute in the same way at run time as the static examples shown, but whose actual source code varies significantly.

### A.3 Prior Transformation

The prior transformation recursively traverses the program tree and applies two local transformations. Firstly it replaces all **observe** statements by `nil`. As **observe** statements return `nil`, this trivially preserves the generative model of the program, but the probability of the execution changes. Secondly, it inspects the binding variables of **let** forms in order to modify the binding expressions for the optimization variables, as specified by the second input of **defopt**, asserting that these are directly bound to a **sample** statement of the form `(sample dist)`. The transformation then replaces this expression by one that stores the result of this sample in Anglican's **store** before returning it. Specifically, if the binding variable in question is `phi-k`, then the original binding expression `(sample dist)` is transformed into

```
(let [value (sample dist)]
  ;; Store the sampled value in Anglican's store
  (store OPTIM-ARGS-KEY
         'phi-k
         value)
  value)
```

After all these local transformation have been made, we wrap the resulting query block in a **do** form and append an expression extracting the optimization variables using Anglican's **retrieve**. This makes the optimization variables the output of the query. Denoting the list of optimization variable symbols from **defopt** as `optim-args` and the query body after applying all the above location transformations as ..., the prior query becomes

```
(query query-args
  (do
    ...
    (map (fn [x] (retrieve OPTIM-ARGS-KEY x))
       optim-args)))
```

Note that the difference in syntax from Figure 3 is because **defquery** is in truth a syntactic sugar allowing users to bind **query** to a variable. As previously stated, **query** is macro that compiles an Anglican program to its CPS transformation. An important subtlety here is that the order of the returned samples is dictated by `optim-args` and is thus independent of the order in which the variables were actually sampled, ensuring consistent inputs for the BO package.

We additionally add a check (not shown) to ensure that all the optimization variables have been added to the store, and thus sampled during the execution, before returning. This ensures that our assumption that each optimization variable is assigned for each execution trace is satisfied.

### A.4 Acquisition Transformation

The acquisition transformation is the same as the prior transformation except we append the acquisition function, `ACQ-F`, to the inputs and then **observe** its application to the optimization variables before returning. The acquisition query is thus

```
(query [query-args ACQ-F]
  (do
    ...
    (let [theta (map (fn [x] (retrieve OPTIM-ARGS-KEY x))
                     optim-args)]
      (observe (factor) (ACQ-F theta))
      theta)))
```

### A.5 Early Termination

To ensure that `q-prior` and `q-acq` are cheap to evaluate and that the latter does not include unnecessary terms which complicate the optimization, we wish to avoid executing code that is not required for generating

the optimization variables. Ideally we would like to directly remove all such redundant code during the transformations. However, doing so in a generic way applicable to all possible programs in a higher order language represents a significant challenge. Therefore, we instead transform to programs with additional early termination statements, triggered when all the optimization variables have been sampled. Provided one is careful to define the optimization variables as early as possible in the program (in most applications, e.g. hyperparameter optimization, they naturally occur at the start of the program), this is typically sufficient to ensure that the minimum possible code is run in practise.

To carry out this early termination, we first wrap the query in a **catch** block with a uniquely generated tag. We then augment the transformation of an optimization variable's binding described in Section A.3 to check if all optimization variables are already stored, and invoke a **throw** statement with the corresponding tag if so. Specifically we replace relevant binding expressions `(sample dist)` with

```
(let [value (sample dist)]
  ;; Store the sampled value in Anglican's store
  (store OPTIM-ARGS-KEY
         'phi-k
         value)
  ;; Terminate early if all optimization variables are sampled
  (if (= (set (keys (retrieve OPTIM-ARGS-KEY)))
         (set optim-args))
    (throw BOPP-CATCH-TAG prologue-code)
    value))
```

where `prologue-code` refers to one of the following expressions depending on whether it is used for a prior or an acquisition transformation

```
;; Prior query prologue-code
(map (fn [x] (retrieve OPTIM-ARGS-KEY x))
             optim-args)

;; Acquisition query prologue-code
(do
  (let [theta (map (fn [x] (retrieve OPTIM-ARGS-KEY x))
                   optim-args)]
  (observe (factor) (ACQ-F theta))
  theta))
```

We note that valid programs for both `q-prior` and `q-acq` should always terminate via one of these early stopping criteria and therefore never actually reach the appending statements in the **query** blocks shown in Sections A.3 and A.4. As such, these are, in practise, only for exposition and error catching.

### A.6 Marginal/MMAP Transformation

The marginal transformation inspects all **let** binding pairs and if a binding variable `phi-k` is one of the optimization variables, the binding expression `(sample dist)` is transformed to the following

```
(do (observe dist phi-k-hat)
    phi-k-hat)
```

corresponding to the **observe<-** form used in the main paper.

### A.7 Error Handling

During program transformation stage, we provide three error-handling mechanisms to enforce the restrictions on the probabilistic programs described in Section 3.

1. We inspect **let** binding pairs and throw an error if an optimization variable is bound to anything other than a **sample** statement.

2. We add code that throws a runtime error if any optimization variable is assigned more than once or not at all.

3. We recursively traverse the code and throw a compilation error if `sample` statements of different base measures are assigned to any optimization variable. At present, we also throw an error if the base measure assigned to an optimization variable is unknown, e.g. because the distribution object is from a user defined `defdist` where the user does not provide the required measure type meta-information.

## Appendix B. Problem Independent Gaussian Process Hyperprior

Remembering that the domain scaling introduced in Section 4.3 means that both the input and outputs of the GP are taken to vary between $\pm 1$, we define the problem independent GP hyperprior as $p(\alpha) = p(\sigma_n)p(\sigma_{3/2})p(\sigma_{5/2})\prod_{i=1}^{D} p(\rho_i)p(\varrho_i)$ where

$$\log(\sigma_n) \sim \mathcal{N}(-5, 2) \tag{34a}$$
$$\log\left(\sigma_{3/2}\right) \sim \mathcal{N}(-7, 0.5) \tag{34b}$$
$$\log\left(\sigma_{5/2}\right) \sim \mathcal{N}(-0.5, 0.15) \tag{34c}$$
$$\log(\rho_i) \sim \mathcal{N}(-1.5, 0.5) \quad \forall i \in \{1, \ldots, D\} \tag{34d}$$
$$\log(\varrho_i) \sim \mathcal{N}(-1, 0.5) \quad \forall i \in \{1, \ldots, D\}. \tag{34e}$$

The rationale of this hyperprior is that the smoother Matérn 5/2 kernel should be the dominant effect and model the higher length scale variations. The Matérn 3/2 kernel is included in case the evidence suggests that the target is less smooth than can be modelled with the Matérn 5/2 kernel and to provide modelling of smaller scale variations around the optimum.

## Appendix C. Full Details for House Heating Experiment

In this case study, illustrated in Figure 1, we optimize the parameters of a stochastic engineering simulation. We use the Energy2D system from Xie (2012) to perform finite-difference numerical simulation of the heat equation and Navier-Stokes equations in a user-defined geometry.

In our setup, we designed a 2-dimensional representation of a house with 4 interconnected rooms using the GUI provided by Energy2D. The left side of the house receives morning sun, modelled at a constant incident angle of $30°$. We assume a randomly distributed solar intensity and simulate the heating of a cold house in the morning by 4 radiators, one in each of the rooms. The radiators are given a fixed budget of total power density $P_{\text{budget}}$. The optimization problem is to distribute this power budget across radiators in a manner that minimizes the variance in temperatures across 8 locations in the house.

Energy2D is written in Java, which allows the simulation to be integrated directly into an Anglican program that defines a prior on model parameters and an ABC likelihood for evaluating the utility of the simulation outputs. Figure 2 shows the corresponding program query. In this, we define a Clojure function `simulate` that accepts a solar power intensity $I_{\text{sun}}$ and power densities for the radiators $P_{\text{r}}$, returning the thermometer temperature readings $\{T_{i,t}\}$. We place a symmetric Dirichlet prior on $\frac{P_r}{P_{\text{budget}}}$ and a gamma prior on $\frac{I_{\text{sun}}}{I_{base}}$, where $P_{\text{budget}}$ and $I_{base}$ are constants. This gives the generative model:

$$p_r \sim \text{Dirichlet}([1, 1, 1, 1]) \tag{35}$$
$$P_r \leftarrow P_{\text{budget}} \cdot p_r \tag{36}$$
$$\upsilon \sim \text{Gamma}(5, 1) \tag{37}$$
$$I_{\text{sun}} \leftarrow I_{\text{base}} \cdot \upsilon. \tag{38}$$

After using these to call `simulate`, the standard deviations of the returned temperatures is calculated for each time point,

$$\omega_t = \sqrt{\sum_{i=1}^{8} T_{i,t}^2 - \left(\sum_{i=1}^{8} T_{i,t}\right)^2} \tag{39}$$

and used in the ABC likelihood `abc-likelihood` to weight the execution trace using a multivariate Gaussian:

$$p\left(\{T_{i,t}\}_{i=1:8,t=1:\tau}\right) = \text{Normal}\left(\omega_{t=1:\tau}; \mathbf{0}, \sigma_T^2 \mathbf{I}\right)$$

where $\mathbf{I}$ is the identity matrix and $\sigma_T = 0.8°\text{C}$ is the observation standard deviation.

Figure 1 demonstrates the improvement in homogeneity of temperatures as a function of total number of simulation evaluations. Visual inspection of the heat distributions also shown in Figure 1 confirms this result, which serves as an exemplar of how BOPP can be used to estimate marginally optimal simulation parameters.

## Acknowledgements

Tom Rainforth is supported by a BP industrial grant. Tuan Anh Le is supported by a Google studentship, project code DF6700. Frank Wood is supported under DARPA PPAML through the U.S. AFRL under Cooperative Agreement FA8750-14-2-0006, Sub Award number 61160290-111668.

## References

Christophe Andrieu, Arnaud Doucet, and Roman Holenstein. Particle Markov chain Monte Carlo methods. *J Royal Stat. Soc.: Series B (Stat. Methodol.)*, 72(3):269–342, 2010.

Jean Bérard, Pierre Del Moral, Arnaud Doucet, et al. A lognormal central limit theorem for particle approximations of normalizing constants. *Electronic Journal of Probability*, 19(94):1–28, 2014.

James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *NIPS*, pages 2546–2554, 2011.

Charles George Broyden. The convergence of a class of double-rank minimization algorithms 1. general considerations. *IMA Journal of Applied Mathematics*, 6(1):76–90, 1970.

B Carpenter, A Gelman, M Hoffman, D Lee, B Goodrich, M Betancourt, M A Brubaker, J Guo, P Li, and A Riddell. Stan: a probabilistic programming language. *Journal of Statistical Software*, 2015.

Katalin Csilléry, Michael GB Blum, Oscar E Gaggiotti, and Olivier François. Approximate Bayesian Computation (ABC) in practice. *Trends in Ecology & Evolution*, 25(7):410–418, 2010.

Robert L Devaney, Luke Devaney, and Luke Devaney. *An introduction to chaotic dynamical systems*, volume 13046. Addison-Wesley Reading, 1989.

Simon Duane, Anthony D Kennedy, Brian J Pendleton, and Duncan Roweth. Hybrid Monte Carlo. *Physics letters B*, 1987.

Katharina Eggensperger, Matthias Feurer, Frank Hutter, James Bergstra, Jasper Snoek, Holger Hoos, and Kevin Leyton-Brown. Towards an empirical foundation for assessing Bayesian optimization of hyperparameters. In *NIPS workshop on Bayesian Optimization in Theory and Practice*, pages 1–5, 2013.

Keisuke Fujii. Extended Kalman filter. *Refernce Manual*, 2013.

Jacob R Gardner, Matt J Kusner, Zhixiang Eddie Xu, Kilian Q Weinberger, and John Cunningham. Bayesian optimization with inequality constraints. In *ICML*, pages 937–945, 2014.

N Goodman, V Mansinghka, D M Roy, K Bonawitz, and J B Tenenbaum. Church: a language for generative models. In *UAI*, pages 220–229, 2008.

Noah D Goodman and Andreas Stuhlmüller. *The Design and Implementation of Probabilistic Programming Languages*. 2014.

Michael U Gutmann and Jukka Corander. Bayesian optimization for likelihood-free inference of simulator-based statistical models. *JMLR*, 17:1–47, 2016.

José Miguel Hernández-Lobato, Matthew W Hoffman, and Zoubin Ghahramani. Predictive entropy search for efficient global optimization of black-box functions. In *NIPS*, pages 918–926, 2014.

José Miguel Hernández-Lobato, Michael A. Gelbart, Ryan P. Adams, Matthew W. Hoffman, and Zoubin Ghahramani. A general framework for constrained Bayesian optimization using information-based search. *JMLR*, 17:1–53, 2016. URL `http://jmlr.org/papers/v17/15-616.html`.

Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Learn. Intell. Optim.*, pages 507–523. Springer, 2011.

Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *J Global Optim*, 13(4):455–492, 1998.

Vikash Mansinghka, Daniel Selsam, and Yura Perov. Venture: a higher-order probabilistic programming platform with programmable inference. *arXiv preprint arXiv:1404.0099*, 2014.

T Minka, J Winn, J Guiver, and D Knowles. Infer .NET 2.4, Microsoft Research Cambridge, 2010.

Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

Radford M Neal. Annealed importance sampling. *Statistics and Computing*, 11(2):125–139, 2001.

Michael A Osborne, Roman Garnett, and Stephen J Roberts. Gaussian processes for global optimization. In *3rd international conference on learning and intelligent optimization (LION3)*, pages 1–15, 2009.

Brooks Paige, Frank Wood, Arnaud Doucet, and Yee Whye Teh. Asynchronous anytime sequential monte carlo. In *NIPS*, pages 3410–3418, 2014.

Clifford A Pickover. *The pattern book: Fractals, art, and nature*. World Scientific, 1995.

Tom Rainforth, Tuan-Anh Le, Jan-Willem van de Meent, Michael A Osborne, and Frank Wood. Bayesian optimization for probabilistic programs. In *Advances in Neural Information Processing Systems*, pages 280–288, 2016.

Carl Rasmussen and Chris Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

Huawei Ruan, Tongyan Zhai, and Edwin Engin Yaz. A chaotic secure communication scheme with extended Kalman filter based parameter estimation. In *Control Applications, 2003. CCA 2003. Proceedings of 2003 IEEE Conference on*, volume 1, pages 404–408. IEEE, 2003.

Bobak Shahriari, Alexandre Bouchard-Côté, and Nando de Freitas. Unbounded Bayesian optimization via regularization. *AISTATS*, 2016a.

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

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical Bayesian optimization of machine learning algorithms. In *NIPS*, pages 2951–2959, 2012.

Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mostofa Ali, Ryan P Adams, et al. Scalable Bayesian optimization using deep neural networks. In *ICML*, 2015.

David Tolpin, Jan-Willem van de Meent, and Frank Wood. Probabilistic programming in Anglican. Springer International Publishing, 2015. URL `http://dx.doi.org/10.1007/978-3-319-23461-8_36`.

Jan-Willem van de Meent, Brooks Paige, David Tolpin, and Frank Wood. Black-box policy search with probabilistic programs. In *AISTATS*, pages 1195–1204, 2016.

David Wingate, Andreas Stuhlmueller, and Noah D Goodman. Lightweight implementations of probabilistic programming languages via transformational compilation. In *AISTATS*, pages 770–778, 2011.

Frank Wood, Jan Willem van de Meent, and Vikash Mansinghka. A new approach to probabilistic programming inference. In *AISTATS*, pages 2–46, 2014.

Charles Xie. Interactive heat transfer simulations for everyone. *The Physics Teacher*, 50(4):237–240, 2012.

Robert Zinkov and Chung-Chieh Shan. Composing inference algorithms as program transformations. *arXiv preprint arXiv:1603.01882*, 2016.