Bayesian Optimization for Probabilistic Programs

Tom Rainforth **Jan-Willem van de Meent** **Michael A. Osborne** **Frank Wood**
Dept of Engineering Science
University of Oxford
{twgr, jwvdm, mosb, fwood}@robots.ox.ac.uk

Abstract

We outline a general purpose framework for black-box marginal maximum a posteriori estimation of probabilistic program variables using Bayesian optimization with Gaussian processes. We introduce the concept of an optimization query, whereby a probabilistic program returns an infinite lazy sequence of increasingly optimal estimates, and explain how a general purpose program transformation would allow the evidence of any probabilistic program, and therefore any graphical model, to be optimized with respect to an arbitrary subset of its variables.

1 Introduction and Background

Probabilistic programming systems (PPS) [1–11] allow probabilistic models to be represented in the form of a generative model and statements for conditioning on data. Informally, one can think of the generative model as the definition of a prior, the conditioning statements as the definition of a likelihood and the output of the program as samples representing expectation values of conditional distribution. The core philosophy of PPS 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, often taking the form of samplers based on Markov chain Monte Carlo (MCMC) [12, 13] or sequential Monte Carlo (SMC) [11, 14].

In this paper we introduce the idea of carrying out marginal maximum a posteriori (MMAP) estimation for probabilistic programs (PP). We aim to optimize with respect to some variables in a program, whilst marginalizing out others. There are number of ways this could prove useful such as hyperparameter optimization, mode finding and when the final required output is a single sample, for example engineering design. Note that our objective function will be based on the evidence of a program, rather than a return value.

In general PPS inference engines are inappropriate for optimization, particularly if the objective function is expensive to evaluate, as is typically the case when it takes the form of an intractable integral. Bayesian optimization (BO) [15–17] is an attractive technique for optimizing expensive functions, as the resulting algorithms are typically very efficient in the number of function evaluations, making it a suitable candidate for the maximization component of MMAP in PP.

Let $f\colon \Theta \to \mathbb{R}$ denote an arbitrary black-box function that can be evaluated for an arbitrary point $\theta \in \Theta$ to produce, potentially noisy, outputs $\hat{Z} \in \mathbb{R}$. BO aims to find the global maximum over a sub-space of permissible solutions $\mathcal{S} \subseteq \Theta$ defined as

$$\theta^* = \underset{\theta \in \mathcal{S}}{\operatorname{argmax}} \, f\left(\theta\right). \tag{1}$$

It is assumed that any noise is unbiased such that $\mathbb{E}[\hat{Z}|\theta] = Z_\theta$ where Z_θ represents the noiseless evaluation of $f\left(\theta\right)$. One can place a prior on f, such as a Gaussian process (GP), and condition upon observed data $D_m = \{\theta_j, \hat{Z}_j\}_{j=1:m}$, to give a posterior over functions $p\left(f|D_m\right)$. This allows estimation of the expected value and uncertainty in Z_θ for all $\theta \in \Theta$. BO calculates such a posterior and uses it to define an acquisition function $a : \mathcal{S} \rightarrow \mathbb{R}$ which assigns a utility to evaluating f at particular θ, based on the trade off between exploration and exploitation in finding the maximum. This acquisition function therefore forms a lower overhead surrogate function which can be optimized to ascertain the next point at which the target function should be evaluated in a sequential fashion.

We will first define a framework for an optimization query. We outline an algorithm for the black-box optimization of any marginal likelihood estimator, with respect to its *input* parameters, before moving on to how MMAP estimation of parameters defined *within* any arbitrary query can be achieved.

2 The Optimization Query

A PP query [18] is a function which takes a program and its inputs, and returns a characterisation of the conditional distribution, for example an infinite lazy sequence of samples. In this section we define a set of requirements for an "optimization query", which instead returns an infinite lazy sequence of increasingly optimal estimates. We will first consider the case of optimizing the evidence of a query q with respect to its input variables and discuss extension to arbitary variables with the program in section 4. We refer to our setup as BOPP (Bayesian optimization for probabilistic programs). We assume q takes as inputs data upon which the query is conditioned Y and the parameters which are being optimized with respect to θ. As we require q to provide an unbiased estimate of its marginal likelihood $p\left(Y|\theta\right)$ we are restricted in the inference engines that can be used. Examples of permissible inference algorithms include importance sampling [19], sequential Monte Carlo [11] and the particle cascade [20], all of which are supported by Anglican [11].

Let q contain a set of latent variables $X = \left\{x_i\right\}_{i=1,\ldots,N}$ (note x_i may have different dimensionality for different i) with prior $p\left(X|\theta\right) = p\left(x_1|\theta\right) \prod_{i=2}^{N} p\left(x_i|x_1,\ldots,x_{i-1},\theta\right)$ parametrized by a set of program inputs $\theta \in \Theta$, and a set of conditioning statements on observations $Y = \left\{y_i\right\}_{i=1,\ldots,N}$, $y_i \in \mathbb{R}$ such that the query defines the joint factorization[1]

$$p\left(X,Y|\theta\right) = p\left(x_1|\theta\right) p\left(y_1|x_1,\theta\right) \prod_{i=2}^{N} p\left(y_i|x_1,\ldots,x_i,\theta\right) p\left(x_i|x_1,\ldots,x_{i-1},\theta\right). \tag{2}$$

We assume that the observations Y are fixed and finite dimensional. BOPP will attempt to find

$$\theta^* = \underset{\theta \in \mathcal{S}}{\operatorname{argmax}}\, f\left(\theta\right) = \underset{\theta \in \mathcal{S}}{\operatorname{argmax}}\, p\left(Y|\theta\right) p\left(\theta\right) = \underset{\theta \in \mathcal{S}}{\operatorname{argmax}}\, p\left(\theta\right) \int_X p\left(X,Y|\theta\right) dX, \tag{3}$$

and will require as inputs Y and $p\left(\theta\right)$. It will be necessary for q to either be passed to the BOPP query, or be defined within its program block. We define the BOPP query to output an infinite lazy sequence $\hat{\theta}_1, \hat{\theta}_2, \ldots$ such that

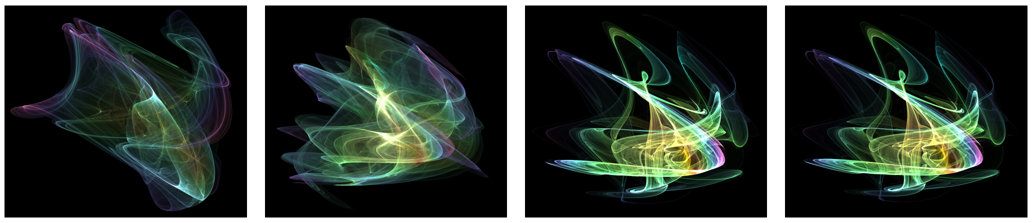
$$\mathrm{E}\left[f\left(\hat{\theta}_m\right)|D_m\right] \geq \mathrm{E}\left[f\left(\hat{\theta}_j\right)|D_m\right], \quad \forall j = 1,\ldots,m-1, \quad m = 1,\ldots. \tag{4}$$

In other words $\hat{\theta}_m$ corresponds to the point that is expected to be the most optimal of those evaluated, which need not always be the point with the maximum function evaluation $\hat{Z}$.

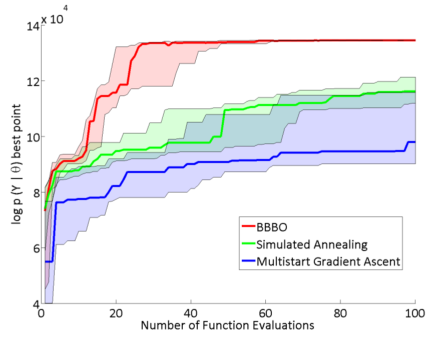
3 Black-Box Bayesian Optimization

Although BO is a technique for optimizing black-box functions, there is substantial flexibility in the BO algorithm itself. For example there are many possible choices for the class of surrogate function including Gaussian processes [16, 24, 25], random forests [26] and neural nets [27]. Further decisions need to be made on the surrogate model hyperparameters, the means of optimizing the acquisition function and, in some cases, the method used to train the surrogate. For PPS the BO algorithm must itself run in a black-box fashion; we therefore bring together a number of established techniques along with our own design decisions to form an BO algorithm that can be run without the need for problem specific user input. We refer to this as black-box Bayesian optimization (BBBO).

[1]Note, there is notational deficiency as in a higher-order PPS variable types, the order of the conditioning for the latent variables and even the number of latent variables can change depending on the program trace.



(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 $\theta = [-2.3, 1.25]^T$



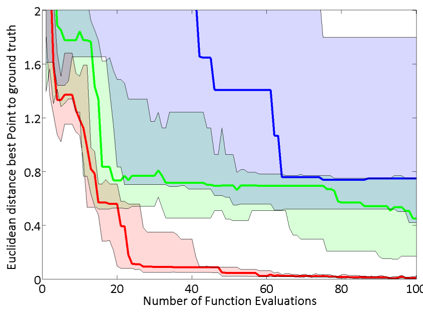


Figure 1: An example application of our BBBO to the problem of optimizing the transition function parameters of an extended Kalman filter for tracking the chaotic latent states of a dynamical system, the model for which is given in Appendix B. We marginalize over a set of latent states X, conditioned upon a series of observations Y, using a particle filtering algorithm (see Cappè SMC overview [21]), and optimize the marginal likelihood $p(Y|\theta)$ with respect to the transition function parameters θ. The synthetic data was generated using 10^4 time steps with ground truth parameters $\theta_1 = -2.3$ and $\theta_2 = 1.25$. Experiments were run using a prototype algorithm written in MATLAB. The top plots show a series of trajectories for different parameters, demonstrating converged to the true attractor. The colormap is based on the speed and curvature of the trajectory, with rendering done using the program Chaoscope [22]. The bottom left plot gives convergence in terms of log marginal likelihood (left) and the bottom right in terms of distance to ground truth. Red corresponds to BBBO, green to simulated annealing [23] and blue to a gradient descent algorithm with random restarts. The solid lines shows the median performance and the bounds of the shaded areas are the upper and lower quartiles over 20 tests, each of which was restricted to 100 function evaluations for all algorithms. In all 20 tests, the BBBO algorithm converged within a small tolerance to the global maximum.

It should be noted that there are existing packages, such as Spearmint [25] and SMAC [26], that can also be run without problem specific input. Due to space restrictions, only a brief high level overview of our algorithm is presented here, with a full scheme given in Appendix 1.

As $p(Y, \theta)$ tends to be tightly peaked around its modes and is a strictly non-negative function, we use the log joint $f(\theta) = \log p(Y|\theta) + \log p(\theta)$ as our objective function [28]. A small number of well spaced points are sampled as an initialization. We use a GP prior on f with a Matérn-$\frac{3}{2}$ [29] covariance function, chosen because it is only once differentiable and therefore only makes weak assumptions about the smoothness of f. The GP prior mean function is taken to be $\mu_{\text{prior}}(\theta) = \log p(\theta) - c$ where c is a constant based on previous evaluations of f. We define a weakly informative, separable hyperprior over the GP hyperparameters and marginalize over them [16, 25] using a Hamiltonian Monte Carlo sampling scheme [30]. This leads to an integrated acquisition function corresponding to a Monte Carlo integration over the individual acquisition functions of each GP, for which we use the expected improvement (EI) [31]. The estimate of the optimal point at any step of the algorithm is given by the point of those queried with the maximum mean value in the GP posterior (with the hyperparameters marginalized out).

4 Marginalization of Arbitrary Variables within a Program

In section 2 we defined a framework for black-box MMAP estimation for the input parameters of a query with respect to a given prior. Although this is a common scenario for optimization in machine learning (for example hyperparameter optimization), we propose extending these ideas to instead optimize with respect to an arbitrary subset of sampled variables in a program. This is equivalent to optimizing with respect to an arbitrary subset of nodes in a graphical model, whilst marginalizing over the others, representing a new method beyond the scope of current BO algorithms.

Consider the Anglican query q in figure 2 as a demonstrative example of the problem. The marginal distribution of q, $p(Y, \theta) = \int_U \int_V p(U)\, p(\theta|U)\, p(V|\theta, U)\, p(Y|V, \theta, U)\, dU dV$, still defines the same objective function as in (3) if we define $X = \{U, V\}$, but θ is no longer at the root of the dependency structure as it was in (2). This causes two problems for optimizing with respect to θ: it is sampled within the program and the corresponding probability distribution is only defined conditioned on one of the parameters we wish to marginalize over U.

We propose dealing with both these issues simultaneously using a program transformation by which we change any **sample** statements for elements of θ into **observe** statements, as detailed in figure 2. In other words we will use the defined probability distribution for sampling θ to condition the program to a particular value of θ. Critically, the distribution defined by the program has not changed, but the query is now a function of θ which can be optimized. This simple but elegant solution means that we can transform any probabilistic program, and therefore any graphical model, to an optimization problem with respect to any of its composite variables.

Some complications remain from the fact that the definition of $p(\theta)$ is now not provided externally, as was the case in the BOPP query defined in section 2, but is instead implicitly defined within the program. This will require careful engineering of our optimization query. If there are implicit hard constraints placed on θ because $p(\theta|U)$ has finite support, we will we need to, at least partially, query the program to test if a θ is valid. Work by Gelbart et al [32] and Hernández et al[33] amongst others has looked into the case of BO under unknown constraints which could prove helpful in solving this problem. One could also look to use further problem transformation to allow partial program evaluation in order to estimate $p(\theta)$ without requiring the full program to be evaluated.

We have assumed that θ is continuous with constant dimension. This need not always be case for probabilistic programs, but arbitrary θ can be considered with suitable adaptation of the GP covariance function. For example, arc kernels [34, 35] would allow the dimension of θ to vary.

Despite these outstanding issues, we believe the application of Bayesian optimization to probabilistic programs is a promising direction for future research, both from a perspective of improving the performance and applicability of PPS, and as flexible method for increasing the scope of BO to the marginalization of arbitrary variables in a graphical model.

```
(defopt q [Y]
 (let [U (sample p-U)
       theta (sample :theta (p-theta U))
       V (sample (p-V U theta))]
  (observe (lik-func U theta V) Y)
  (predict :U-V [U V])
  (maximize-wrt :theta)))

(defquery qT [Y theta]
 (let [U (sample p-U)
       _ (observe (p-theta U) theta)
       V (sample (p-V U theta))]
  (observe (lik-func U theta V) Y)
  (predict :U-V [U V])))
```

```
(defn doopt [qT Y n method]
 (letfn [(point-seq [points theta]
        (lazy-seq
         (let [samples (->> (doquery
                                method qT [Y theta])
                            (take n))
              log-Z (log-marginal samples)
              predicts (map get-predicts samples)
              points (conj points
                           [theta log-Z predicts])
              [theta-next i-best] (bo-acquire points)]
              (cons (nth points i-best)
                    (point-seq points theta-next)))))]
  (point-seq (sample-initial-points qT Y n method))))
```

Figure 2: Possible **defopt** design written in Anglican. Here **sample** samples from a distribution, **observe** conditions on data and **predict** defines program output. Top left is a simple example **defopt** query where we want to optimize θ. Note `p-U`, `p-theta`, `p-V` and `lik-func` all represent the distribution definitions. The macro **defopt** transforms the query q to a query of the form shown bottom left as `qT`. A query of this form may be passed to the function **doopt**, which returns sequence of increasingly optimal triples `[theta log-Z predicts]`. After first using **sample-initial-points** to generate and evaluate a set of initialization points, the algorithm alternates between performing inference at a given `theta`, using **doquery** to estimate the log marginal likelihood for that `theta`, and calling **bo-acquire** to select the next `theta` to evaluate. **bo-acquire** also calculates which of the points so far is expected to be the optimum using the integrated mean function from our GP posterior. This point is added to the lazy infinite sequence of returned points at each iteration.

References

[1] Brian Milch, Bhaskara Marthi, Stuart Russell, David Sontag, Daniel L. Ong, and Andrey Kolobov. BLOG : Probabilistic Models with Unknown Objects. In *IJCAI*, 2005.

[2] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. ProbLog: A probabilistic prolog and its application in link discovery. *IJCAI International Joint Conference on Artificial Intelligence*, pages 2468–2473, 2007.

[3] Noah Goodman, Vikash Mansinghka, Daniel M Roy, Keith Bonawitz, and Joshua B Tenenbaum. Church: a language for generative models. In *Proc. 24th Conf. Uncertainty in Artificial Intelligence (UAI)*, pages 220–229, 2008.

[4] Avi Pfeffer. Figaro: An object-oriented probabilistic programming language. Technical report, 2009.

[5] A McCallum, K Schultz, and S Singh. Factorie: Probabilistic programming via imperatively defined factor graphs. In *Advances in Neural Information Processing Systems*, volume 22, pages 1249–1257, 2009.

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

[7] Brooks Paige and Frank Wood. A compilation target for probabilistic programming languages. *arXiv preprint arXiv:1403.0504*, 2014.

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

[9] Stan: A c++ library for probability and sampling, version 2.7.0, 2015. URL `http://mc-stan.org/`.

[10] Tom Minka, John Winn, John Guiver, and David Knowles. Infer .net 2.4, 2010. microsoft research cambridge.

[11] Frank Wood, Jan Willem van de Meent, and Vikash Mansinghka. A new approach to probabilistic programming inference. In *Proceedings of the 17th International conference on Artificial Intelligence and Statistics*, pages 2–46, 2014.

[12] W Keith Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

[13] David Wingate, Andreas Stuhlmueller, and Noah D Goodman. Lightweight implementations of probabilistic programming languages via transformational compilation. In *International Conference on Artificial Intelligence and Statistics*, pages 770–778, 2011.

[14] Adrian Smith, Arnaud Doucet, Nando de Freitas, and Neil Gordon. *Sequential Monte Carlo methods in practice*. Springer Science & Business Media, 2013.

[15] Jonas Mockus. *Bayesian approach to global optimization: theory and applications*, volume 37. Springer Science & Business Media, 2012.

[16] 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.

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

[18] Noah D Goodman, Vikash K Mansinghka, Daniel Roy, Keith Bonawitz, and Joshua B Tenenbaum. Church: a language for generative models. 2008.

[19] Peter W Glynn and Donald L Iglehart. Importance sampling for stochastic simulations. *Management Science*, 35(11):1367–1392, 1989.

[20] Brooks Paige, Frank Wood, Arnaud Doucet, and Yee Whye Teh. Asynchronous anytime sequential monte carlo. In *Advances in Neural Information Processing Systems*, pages 3410–3418, 2014.

[21] Olivier Cappé, Simon J Godsill, and Eric Moulines. An overview of existing methods and recent advances in sequential monte carlo. *Proceedings of the IEEE*, 95(5):899–924, 2007.

[22] http://www.chaoscope.org/.

[23] Emile Aarts and Jan Korst. Simulated annealing and boltzmann machines. 1988.

[24] Michael Osborne. *Bayesian Gaussian Processes for Sequential Prediction, Optimisation and Quadrature*. PhD thesis, PhD thesis, University of Oxford, 2010.

[25] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[26] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Learning and Intelligent Optimization*, pages 507–523. Springer, 2011.

[27] Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Md Patwary, Mostofa Ali, Ryan P Adams, et al. Scalable bayesian optimization using deep neural networks. *arXiv preprint arXiv:1502.05700*, 2015.

[28] Michael Osborne, Roman Garnett, Zoubin Ghahramani, David K Duvenaud, Stephen J Roberts, and Carl E Rasmussen. Active learning of model evidence using bayesian quadrature. In *Advances in Neural Information Processing Systems*, pages 46–54, 2012.

[29] Michael L Stein. *Interpolation of spatial data: some theory for kriging*. Springer Science & Business Media, 2012.

[30] Radford M Neal. Mcmc using hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, 2, 2011.

[31] Eric Brochu, Vlad M Cora, and Nando De Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010.

[32] Michael A Gelbart, Jasper Snoek, and Ryan P Adams. Bayesian optimization with unknown constraints. *arXiv preprint arXiv:1403.5607*, 2014.

[33] José Miguel Hernández-Lobato, Matthew W Hoffman, and Zoubin Ghahramani. Predictive entropy search for efficient global optimization of black-box functions. In *Advances in Neural Information Processing Systems*, pages 918–926, 2014.

[34] Frank Hutter and Michael A Osborne. A kernel for hierarchical parameter spaces. *arXiv preprint arXiv:1310.5738*, 2013.

[35] Kevin Swersky, David Duvenaud, Jasper Snoek, Frank Hutter, and Michael A Osborne. Raiders of the lost architecture: Kernels for bayesian optimization in conditional parameter spaces. *arXiv preprint arXiv:1409.4011*, 2014.

[36] Ronald L Iman. Latin hypercube sampling. *Encyclopedia of quantitative risk analysis and assessment*, 2008.

[37] Anthony Lee and Nick Whiteley. Variance estimation and allocation in the particle filter. *arXiv preprint arXiv:1509.00394*, 2015.

[38] Daniel James Lizotte. *Practical bayesian optimization*. University of Alberta, 2008.

[39] Matthew D Homan and Andrew Gelman. The no-u-turn sampler: Adaptively setting path lengths in hamiltonian monte carlo. *The Journal of Machine Learning Research*, 15(1):1593–1623, 2014.

[40] Iain Murray and Ryan P Adams. Slice sampling covariance hyperparameters of latent gaussian models. In *Advances in Neural Information Processing Systems*, pages 1732–1740, 2010.

[41] Donald R Jones, Cary D Perttunen, and Bruce E Stuckman. Lipschitzian optimization without the lipschitz constant. *Journal of Optimization Theory and Applications*, 79(1):157–181, 1993.

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

[43] Keisuke Fujii. Extended kalman filter. *Refernce Manual*, 2013.

[44] 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.

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

A Black-Box Bayesian Optimization Algorithm in Detail

In this section we give a more extensive outline our BBBO algorithm for optimizing a target function of the form $f(\theta) = p(\theta)\,p(Y|\theta)$. Although we have focussed on the case where $p(Y|\theta)$ corresponds to a marginal likelihood, the introduced methods applies to any $p(\theta)$ and $p(Y|\theta)$, including the case where these are not even probability densities. To aid with the application of our algorithm to arbitrary programs we make two transformations to our target function that leave the solution to (3) unchanged. Firstly we take the logarithm of f to account for the anticipation that $p(Y,\theta)$ will be tightly peaked around its modes and so that the support of our target is the full real line, noting that $p(Y,\theta)$ is strictly positive and therefore cannot be modelled by a GP without a transformation. Secondly we use affine transformation after each data point is observed so that the evaluated values of $\log p(Y|\theta)$ have zero mean and unit variance in the transformed space[2]. This is done to adjust for the varying scaling of different problems and means that we can more sensibly specify automatic hyperpriors on the signal and noise variances.

Algorithm 1 outlines the general procedure for BBBO which starts by sampling m_0 initialization points. This is done to prevent because typically with a few points there is insufficient information to make well informed decisions about the optimal point to next sample and we wish to avoid the BO algorithm attempting to do anything other than exploring the space at this time. We take as a default $m_0 = \min\left(1 + 4\left\|\theta\right\|_0, 20\right)$ where $\left\|\theta\right\|_0$ is the dimensionality of θ, but note that the user may wish to change this depending on the intended total number of function evaluations. In general the choice of m_0 will not be critical to the performance of the algorithm. When the prior defines bounds corresponding to a hypercube, we use a Latin hypercube method [36] to achieve well separated points, otherwise we resort to (potentially approximately) sampling the start points from $p(\theta)$.

Algorithm 1 BLACK-BOX BAYESIAN OPTIMIZATION

```
 1: Inputs: $p_{Y|\theta}, p_\theta, m_0$
 2: $\{\theta_j\}_{j=1:m_0} \leftarrow$ GENERATEINITIALSAMPLES($p_\theta$)
 3: $\hat{Z}_j \leftarrow \log p_{Y|\theta}(\theta_j) + \log(p_\theta(\theta_j)), \quad \forall j = 1,\dots,m_0$
 4: $m \leftarrow m_0$
 5: while true do
 6:     $\alpha_\ell \sim p\left(\alpha|\theta_{1:m}, \hat{Z}_{1:m}\right) \quad \forall \ell = 1:L$    ▷ Generate samples from posterior on GP hyperparameters
 7:     $\left\{\mu_m^\ell(\cdot), k_m^\ell(\cdot,\cdot)\right\} \leftarrow$ GPTRAIN($\theta_{1:m}, \hat{Z}_{1:m}, \alpha_\ell, p_\theta$) $\quad \forall \ell = 1:L$
 8:     $\hat{\theta}_m \leftarrow \left\{\text{argmax}_{j=1:m} \frac{1}{L}\mu_m^\ell(\theta_j)\right\}$    ▷ Evaluated point with best expected $Z$
 9:     $a_m(\cdot) \leftarrow \frac{1}{L}\sum_{\ell=1:L} \zeta\left(\cdot\,; \alpha_\ell, \mu_m^\ell(\cdot), k_m^\ell(\cdot,\cdot)\right)$    ▷ Integrated acquisition function
10:     $\theta_{m+1} \leftarrow \text{argmax}_\theta\, a_m(\theta)$    ▷ Next point to try
11:     $m \leftarrow m + 1$
12:     $\hat{Z}_m \leftarrow \log p_{Y|\theta}(\theta_m) + \log(p_\theta(\theta_m))$
13: end while
14: return $\hat{\theta}_m$
```

A GP prior is used to model f and we assume a Gaussian likelihood for $p(\hat{Z}|f(\theta)) = \frac{1}{\sigma_n\sqrt{2\pi}}\exp\left(-\frac{\left(\hat{Z}-f(\theta)\right)^2}{\sigma_n^2}\right)$ where σ_n is stationary anticipated standard deviation in our noisy function evaluations that we consider to be a hyperparameter of the GP. We note that using a Gaussian likelihood will be an approximation to the truth as even though the $\hat{Z}$ are unbiased, they need not be Gaussian distributed. It may be possible to improve upon this assumption in future work, for example by using variance estimation of particle filters [37]. This assumption is, however, highly convenient as it ensures that our posterior after observing m datapoints $D_m = \{\theta_j, \hat{Z}_j\}_{j=1:m}$ is also a GP, fully defined by a posterior mean function $\mu_m(\theta;\alpha)$ and covariance function $k_m(\theta,\theta';\alpha)$.

As we have only considered bounded optimizations with uniform prior $p(\theta)$ in our experiments, we use a zero prior mean function for the GP. For unbounded optimizations it will be necessary to have a prior mean function which diminishes away from a region of interest. If $p(\theta)$ is cheap to evaluate, we suggest setting the GP prior mean to $\mu_{\text{prior}}(\theta) = \log p(\theta) - c$ where c is set to the Monte Carlo estimate for the mean of $\log p(\theta)$ over the initialization points. If $p(\theta)$ cannot be

[2]In future work we intend to refine this so that this scaling is not continually updated but based on a number of samples at the start of the process.

evaluated cheaply we suggest using the same expression but to replace $p(\theta)$ with a moment matched normal distribution based on the previous evaluations.

We use a Matérn-$\frac{3}{2}$ kernel [29] for the covariance function

$$k(\theta,\theta')_0 = \sigma_f^2 \left(1 + \frac{\sqrt{3}\,\|\theta-\theta'\|_2}{\rho}\right) \exp\left(-\frac{\sqrt{3}\,\|\theta-\theta'\|_2}{\rho}\right) \tag{5}$$

where σ_f and ρ are hyperparameters corresponding to a signal standard deviation and length scale respectively. A key feature of the Matérn-$\frac{3}{2}$ kernel is that it is only once differentiable and therefore makes relatively weak assumptions about the smoothness of f. We refer to $\alpha = \{\rho, \sigma_f, \sigma_n\}$ as the GP hyperparameters. We intend to investigate more complicated covariance functions, such as using multiple kernels and non isotropic length scales, in future work.

As the performance of our GP surrogate will depend strongly on α, we introduce a weakly informative, separable hyperprior $p(\alpha) = p(\rho)\,p(\sigma_f)\,p(\sigma_n)$:

$$\log_{10}(\rho) \sim \mathcal{N}\left(\sum_{d=1}^{\|\theta_1\|_0} \log_{10}\frac{\Delta_d}{10},\quad 1+\log_{10}(\max\Delta_d - \min\Delta_d)\right) \tag{6a}$$

$$\log_{10}(\sigma_f) \sim \mathcal{N}(1, 0.5) \tag{6b}$$

$$\log_{10}(\sigma_n) \sim \mathcal{N}(-2, 2)\,. \tag{6c}$$

For bounded optimizations, Δ_d is the maximum allowable variation of θ in dimension d as defined by the bounds. For unbounded optimizations we instead take $\Delta_d = \frac{3}{2}\left(\max_{j=1:m}\theta_j^d - \min_{j=1:m}\theta_j^d\right)$ where θ_j^d denotes dimension d of each sampled point θ_j. The hyperprior for ρ effectively reflects the range of length scales we might reasonably expect to infer given the limited evaluations that our algorithm expects to take. The hyperpriors for σ_f and σ_n are based on the fact that the problem has been scaled to have unit variance which bounds the range a sensible values for a finite number of samples.

We use the expected improvement (EI) [31] above some threshold $\xi > 0$ as the acquisition function $\zeta_m(\theta;\alpha)$ for a single GP with a particular given α

$$\begin{aligned}\zeta_m(\theta;\alpha) &= \int_{\mu^+ + \xi}^{\infty} p\left(z \mid \mu_m(\theta;\alpha), \sigma_m(\theta;\alpha)\right)\left(z - \mu^+ - \xi\right) dz \\ &= \begin{cases} \left(\mu_m(\theta;\alpha) - \mu^+ - \xi\right)\Phi\left(\gamma_m(\theta)\right) + \sigma_m(\theta;\alpha)\,\lambda\left(\gamma_m(\theta)\right), & \sigma_m(\theta;\alpha) > 0 \\ 0, & \sigma_m(\theta;\alpha) = 0 \end{cases}\end{aligned} \tag{7}$$

where $\sigma_m(\theta;\alpha) = \sqrt{k_\tau(\theta,\theta)}$, Φ is the cumulative distribution of a unit normal, , $\gamma_m(\theta) = \frac{\mu_m(\theta;\alpha) - \mu_m^+ - \xi}{\sigma_m(\theta;\alpha)}$, λ is the probability density of a unit normal and $\mu_m^+ = \max_{j\in\{1,\dots,\tau\}} \mu_m(\theta_j)$. We use Lizotte's [38] suggestion of $\xi = 0.01\sigma_f$. Marginalizing over α [16, 25] with respect to its posterior probability gives the final integrated acquisition function:

$$a_m(\theta) = \int_\alpha \zeta_m(\theta;\alpha)\, p(\alpha \mid D_m)\, d\alpha. \tag{8}$$

A Hamiltonian Monte Carlo (HMC) method [30] is used to sample from the GP-hyperparameter posterior $p(\alpha|D_m)$. HMC was chosen because the availability of analytic derivatives of the GP log marginal likelihood. For each iteration of the BO algorithm we use a default of 100 HMC iterations (i.e. 100 accept / reject steps) but note that user may wish to vary this depending on how expensive the target function is to evaluate. For each HMC iteration we use 5 leapfrog updates and for hyperparameter κ, the step size is set to $\min\left(0.01\sigma_\kappa, 0.1/\left|\frac{\partial \log p(\alpha|D_m)}{\partial\kappa}\right|\right)$ where σ_κ is the standard deviation of κ under the hyperprior and $\left|\frac{\partial \log p(\alpha|D_m)}{\partial\kappa}\right|$ is the absolute value of the derivative of the log GP-hyperparameter posterior with respect to κ at the current point. Although we found this worked well in practise, a more advanced HMC sampler that avoids the need to set the number of leapfrog updates or the step size might be preferable, for example NUTS [39]. Alternatively one could consider using slice sampling for the GP hyperparameters [40] as is employed in a number of other BO implementations. Note that the hyperparameter samples are discarded from one iteration of the BO algorithm to the next.

We use a simple simulated annealing [23] algorithm for optimizing the acquisition function. A number of annealing trajectories are simulated in parallel and heuristics used for setting the cooling schedule, step sizes and starting point of the trajectories. These heuristics are based on preliminary evaluations of the acquisition function at randomly sampled points. We acknowledge that the choice of simulated annealing is unlikely to prove the most efficient and intend to investigate alternatives in future work. Possible alternatives include DIRECT [41], which has the advantage of being parameter free but the disadvantage that it requires the optimization to be bounded, and gradient based methods, noting that the derivatives are analytically available.

B Extended Kalman Filter for the Pickover Chaotic Attractor

As an example application we consider the case of optimizing the transition function parameters of an extended Kalman filter for the tracking 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[3]. 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 [43] must be incorporated [44]. 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 $y_t \in \mathbb{R}^K, \; t = 1, 2, \dots$ in some K dimensional observation space which we believe has a lower dimensional latent space $x_t \in \mathbb{R}^D$ corresponding to a chaotic attractor of known type but with unknown parameters. Given observations up to some time T, we wish to performance inference over the latent space using an extended Kalman filter as defined by

$$x_0 \sim \mathcal{N}\left(\mu_0, \sigma_0 I\right) \tag{9}$$

$$x_t = A\left(x_{t-1}, \theta\right) + \delta_{t-1}, \quad \delta_{t-1} \sim \mathcal{N}\left(0, \sigma_q I\right) \tag{10}$$

$$y_t = C x_t + \varepsilon_t, \quad \varepsilon_t \sim \mathcal{N}\left(0, \sigma_y I\right) \tag{11}$$

where I is the identity matrix, C is a known $K \times D$ matrix, μ_0 is the expected starting position, and σ_0, σ_q and σ_y are all scalars which are assumed to be known. The transition function $A\left(\cdot,\cdot\right)$ is

$$x_{t,1} = \sin\left(\beta x_{t-1,2}\right) - \cos\left(\frac{5x_{t-1,1}}{2}\right) x_{t-1,3} \tag{12a}$$

$$x_{t,2} = -\sin\left(\frac{3x_{t-1,1}}{2}\right) x_{t-1,3} - \cos\left(\eta x_{t-1,2}\right) \tag{12b}$$

$$x_{t,3} = \sin\left(x_{t-1,1}\right) \tag{12c}$$

corresponding to a type of Pickover attractor [45] with unknown parameters $\theta = \{\beta, \eta\}$ which we wish to optimize. Note that η and $-\eta$ will give the same behaviour.

Data was generated for 10^4 time steps using the parameters of $\mu_0 = [-0.2149, -0.0177, 0.7630]^T$, $\sigma_0 = 0.01$, $\sigma_q = 0.01$, $\sigma_y = 0.2$, 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). Our prior $p\left(\theta\right)$, corresponds to a uniform in over a bounded region such that

$$p\left(\theta\right) = \begin{cases} 1/18, & \text{if} -3 \le \beta \le 3 \;\cap\; 0 \le \eta \le 3 \\ 0, & \text{otherwise} \end{cases}. \tag{13}$$

[3]It is beyond the scope of this paper to properly introduce chaotic systems. We refer the reader to Devaney [42] for an introduction