Constrained Bayesian Optimization and Applications

Constrained Bayesian Optimization and Applications

A dissertation presented

by

Michael Adam Gelbart

to

the Committee on Higher Degrees in Biophysics

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

in the subject of

Biophysics

Harvard University

Cambridge, Massachusetts

May 2015

Dissertation Advisor: Ryan Prescott Adams Michael Adam Gelbart

Constrained Bayesian Optimization and Applications

Abstract

Bayesian optimization is an approach for globally optimizing black-box functions that are expensive to evaluate, non-convex, and possibly noisy. Recently, Bayesian optimization has been used with great effectiveness for applications like tuning the hyperparameters of machine learning algorithms and automatic A/B testing for websites. This thesis considers Bayesian optimization in the presence of black-box constraints. Prior work on constrained Bayesian optimization consists of a variety of methods that can be used with some efficacy in specific contexts. Here, by forming a connection with multi-task Bayesian optimization, we formulate a more general class of constrained Bayesian optimization problems that we call *Bayesian optimization with decoupled constraints*. In this general framework, the objective and constraint functions are divided into *tasks* that can be evaluated independently of each other, and *resources* with which these tasks can be performed. We then present two methods for solving problems in this general class. The first method, an extension to a constrained variant of *expected improvement*, is fast and straightforward to implement but performs poorly in some circumstances and is not sufficiently flexible to address all varieties of decoupled problems. The second method, *Predictive Entropy Search with Constraints* (PESC), is highly effective and sufficiently flexible to address all problems in the general class of decoupled problems without any *ad hoc* modifications. The two weaknesses of PESC are its implementation difficulty and slow execution time. We address these issues by, respectively, providing a publicly available implementation within the popular Bayesian optimization software *Spearmint*, and developing an extension to PESC that achieves greater speed without significant performance losses. We demonstrate the effectiveness of these methods on real-world machine learning meta-optimization problems.

Contents

List of Figures

List of Algorithms

List of Tables

Acknowledgements

I am very greatful to Michele Jakoulov and Jim Hogle from the Biophysics Program for being encouraging, understanding, approachable, and an incredible team. It has been a pleasure exchanging ideas with everyone in the Harvard Intelligent Probabilistic Systems group, especially my collaborators José Miguel Hernández-Lobato, Jasper Snoek, and Oren Rippel, as well as Brandon Reagen and Sophia Shao from the Harvard Architecture, Circuits and Compilers group. This dissertation would not have been possible without the guidance, great ideas, and, most of all, contagious enthusiasm of my advisor Ryan Adams. I also thank my committee members David Parkes, Tirthankar Dasgupta, Yaron Singer, and Finale Doshi-Velez. In addition to my research activities, I have had the privilege of teaching alongside Harry Lewis and John Girash, both of whom have been thoughtful mentors to me. Finally, I thank my friends across the globe, my wife Adriana, my parents Dan and Daphne Gelbart, and the rest of my family in British Columbia. You have all been amazing supporters.

Notation and Terminology

The term *Bayesian optimization* is slightly unfortunate. Another name, which I prefer, is *model-based optimization*. This name captures the distinguishing feature of these methods, namely that they rely on a model of the unknown objective function to reason about future data collection. Bayesian optimization is the specific case of model-based optimization in which the model is Bayesian. This means that one begins with a prior distribution over functions and a likelihood function, and combine them using Bayes' theorem to form a posterior distribution over unknown functions given the data. Much of the work presented in this thesis applies beyond the realm of Bayesian optimization to any model-based optimization method. However, at the cost of being less precise, I will use the term Bayesian optimization to be consistent with conventional naming practices in this field. Because the model of the objective function can be referred to as a *response surface*, *surrogate model*, or *meta-model*, Bayesian optimization may also be called response-surface optimization, surrogate-based optimization, or meta-modeling optimization. A specific modeling approach, called the *Gaussian process regression* in some fields and *kriging* in other fields, leads to the names *Gaussian process optimization* and *sequential kriging optimization*. Another term used to refer to Bayesian optimization is *efficient global optimization*; this refers a particular algorithm from Jones et al. (1998) that uses Expected Improvement.

In this thesis the term *recommendation* refers to the best solution found by a Bayesian optimization method. In other words, this is the solution the algorithm is recommending that you use, once optimization is completed. I will use the term *suggestion* to mean the suggested location of the next function evaluation during Bayesian optimization. Thus, a Bayesian optimization method makes a suggestion at every iteration, and may also make a recommendation at each iteration, but certainly makes a recommendation when it terminates.

I will use $\hat{\mathbf{x}}$ to denote a suggestion and $\mathbf{x}_\star$ to denote the random variable associated with the (unknown) true solution to the optimization problem. I occasionally also use $\mathbf{x}_\star$ to refer to a recommendation or the true solution itself. I use $\alpha(\mathbf{x})$ to denote an acquisition function, $\mathrm{H}[X]$ to denote the entropy of the random variable X, and $\mathcal{N}$ to denote the Gaussian (normal) distribution. I also use the terms *function evaluation*, *query*, *observation*, *experiment* and *job* more or less interchangeably. The term *task* has a similar but more precise meaning; see Sections 2.8

and 5.3. I use the terms *input*, *location*, and *parameters*, and the symbol $\mathbf{x}$, to refer to a setting of the variables over which we are optimizing. I refer to the space over which we are optimizing as the *domain*, *search space*, *input space*, or *design space*, and use the symbol $\mathcal{X}$.

Another case of contentious terminology lies in the problem of *exploration* vs. *exploitation* (sometimes also called *learning* vs. *earning* in financial settings). I define exploration as performing an action expected to produce lower immediate reward in hopes of learning about the system and thus attaining greater future reward. Exploitation, on the other hand, is greedy behavior aimed to produce the highest possible immediate reward. Confusingly, the terms exploration and exploitation are often used to describe methods like Expected Improvement (EI, Section 2.4.1). In fact, by the definition above, EI is a method that always exploits, since it always maximizes expected immediate reward; even when EI selects a location with high variance rather than low mean, it does so because of the possibility of immediately unveiling a good solution, rather than as a strategy to learn about the function for future gains. In other words, myopic methods like EI seek actions that are sensible if they were the last action to be taken, whereas exploration is defined precisely as non-myopic behavior. To make this distinction clear, I will use the terms *local search* and *global search* where the terms exploitation and exploration are often used. Local search refers to searching in regions with high expected reward (i.e., low expected function value), and *global search* refers to searching in regions with high variance.

Finally, the optimization literature is split between works that consider maximization and those that consider minimization. In this thesis, I use the minimization convention. For the sake of consistency, I describe other work in terms of minimization even when the original paper describes a method for maximization, making the appropriate changes in the descriptions as needed. Similarly, inequality constraints can be written either as $c(\mathbf{x}) \leq 0$ or $c(\mathbf{x}) \geq 0$. In this thesis I take all inequality constraints to be of the latter form, namely $c(\mathbf{x}) \geq 0$. Note that any inequality constraint $c(\mathbf{x}) \leq c_0$ or $c(\mathbf{x}) \geq c_1$ can be represented this way by transforming to a new variable $\hat{c}(\mathbf{x}) \equiv c_0 - c(\mathbf{x}) \geq 0$ or $\hat{c}(\mathbf{x}) \equiv c(\mathbf{x}) - c_1 \geq 0$, respectively, so I only consider inequality constraints of the form $c(\mathbf{x}) \geq 0$ without loss of generality.

Chapter 1

Introduction

Consider a food company that is trying to design a low-calorie variant of its most popular cookie. The design space of possible recipes might include parameters such as quantities of various ingredients, cookie size, baking times, etc. For any proposed recipe, the company can create a batch of cookies and measure the calorie content. The minimization of the calorie count as a function of the recipe parameters is an *optimization problem*:

$$\min_{\mathbf{x}} f(\mathbf{x}),$$

where $\mathbf{x}$ is a vector representing a cookie recipe and $f(\mathbf{x})$ denotes the number of calories per cookie created with recipe $\mathbf{x}$. The field of optimization spans problems of many varieties. This thesis is about *Bayesian optimization* (Mockus et al., 1978), which is a method for solving optimization problems like the one described above, and, more concretely, optimization problems with the following challenging properties:

(A) Black-box: the function f can only be evaluated point-wise. For example, we do not have access to its derivatives, although we may assume that it is differentiable.

(B) Expensive evaluations: querying f is expensive, by which we typically mean that computing $f(\mathbf{x})$ for any $\mathbf{x}$ takes a long time.

(C) Noisy evaluations: when we evaluate the function f, we may only be able to observe its

true value corrupted with some zero-mean noise.

(D) Global and non-convex: we are searching for a global minimum of a non-convex function within a bounded domain $\mathcal{X}$.

High-dimensional problems with these properties are extremely hard to solve. Therefore, Bayesian optimization methods are typically only applied to low-dimensional settings, i.e., when the number of parameters is well below 100 (but see Section 2.10 for recent advances in this area).

Property (B) precludes the use of simplistic methods like grid search or random search. Even in $D = 10$ dimensions we suffer the curse of dimensionality and grid search could take an extremely (i.e, exponential in D) long time to find a good solution. Unlike grid search or random search, Bayesian optimization learns from past experience to make better decisions about future function evaluations. Because it seeks a global optimum, Bayesian optimization methods must automatically manage the trade-off between global search (exploring the entire domain) and local search (refining potential solutions).

We now address another aspect of complexity in our example: in addition to being low-calorie, the food company must also ensure that the new cookie is tasty. Therefore, for each proposed recipe, they perform a taste test with a group of customers and require that at least 95% of test subjects like the new cookie. We now have a *constrained optimization problem*:

$$\min_{\mathbf{x}} f(\mathbf{x}) \text{ s.t. } c(\mathbf{x}) \geq c_0 \,, \tag{1.1}$$

where $c(\mathbf{x})$ is the true fraction of the population that likes recipe $\mathbf{x}$ and c_0 in this case is 0.95. This problem has the properties enumerated above. The functions f and c are black-boxes (property (A)) in the sense that we do not know their mathematical form. They are expensive (property (B)) because significant resources are required to evaluate them (especially c) for a particular $\mathbf{x}$. The function c is noisy (property (C)) because the fraction of human test subjects in a particular group that like a cookie is only a noisy estimate of the true fraction of the population that likes the cookie $\mathbf{x}$. Finally, we are interested in a global solution (property (D)) within a bounded domain.

The goal of this thesis is to study and propose methods for solving this type of constrained optimization problem using Bayesian optimization. More precisely, we seek to solve problems with K inequality constraints of the form

$$\min_{\mathbf{x}} f(\mathbf{x}) \text{ s.t. } \forall k \in \{1, \ldots, K\},\ c_k(\mathbf{x}) \geq 0\,, \tag{1.2}$$

possessing some or all of properties (A) to (D) discussed above. One can imagine many variants on the cookie problem that take the form of Eq. (1.2). For example, the company may introduce additional constraints like the cost of ingredients. Or, perhaps the calorie content can be computed cheaply using a simple calculation rather than requiring the cookie to be made. Or, perhaps the company wishes to flip the objective and the constraint, maximizing the tastiness while keeping the calorie count below some threshold. The goal of this thesis is to present methods that solve a wide variety of problems of these types.

1.1 Motivation

A vast diversity of constrained optimization problems take the form described above. Examples include minimizing the cost of materials for a new bridge, subject to the constraint that all safety margins are met; or, maximizing profit at a store by varying prices and adding promotions; or, maximizing the expected return of a financial portfolio while keeping the risk of default below some threshold. Of particular interest are software optimization problems, also called *computer experiments*. Many software systems, including machine learning algorithms, contain parameters whose values are both crucial to the effectiveness of the system and yet also difficult to set. For example, we may wish to tune the parameters of a phone or tablet speech recognition system to maximize performance such that the user's speech is transcribed within some acceptable time limit. Both the performance and speed of the speech recognition system are black-box functions, whose dependence on the system's parameters may be poorly understood. Many machine learning systems can be viewed as black-box functions whose parameters can be tuned to increase performance. Bayesian optimization has recently been used successfully in this area; see Snoek et al. (2012); Bergstra et al. (2011).

Another use of constraints arises when the search space is known *a priori* but occupies a complicated volume that cannot be expressed as simple coordinate-wise bounds on the search variables. For example, in a chemical synthesis experiment, it may be known that certain combinations of reagents cause an explosion to occur. This constraint is not unknown in the sense of being a discovered property of the environment as in the examples above—we do not want to discover the constraint boundary by trial and error explosions of our laboratory. Rather, we would like to specify this constraint using a Boolean noise-free oracle function that declares input vectors as valid or invalid. Constrained Bayesian optimization naturally encapsulates such constraints.

1.2 Outline

This thesis is structured as follows. Chapter 2 introduces the background material and prior work upon which this work builds. Chapter 3 builds upon an existing constrained Bayesian optimization method, Expected Improvement with Constraints (EIC), by resolving some inconsistencies and then extending EIC to *decoupled* problems, in which the objective and constraint(s) can be evaluated independently of each other. Chapter 4 presents *predictive entropy search with constraints* (PESC), an improved method with an appealing theoretical motivation that outperforms EIC and other methods in experiments. We also explain why PESC is a much more natural approach than EIC when solving decoupled problems, and present a modification to PESC called PESC-F that significantly speeds up the method. Chapter 5 presents a general framework for Bayesian optimization with decoupled constraints, along with a taxonomy of different types of decoupling. Chapter 6 provides empirical analyses that (1) demonstrate the need for constrained Bayesian optimization methods in the first place, (2) examine the performance of EIC and PESC relative to each other and to other methods, (3) demonstrate the need for methods that harness the special structure of decoupled problems, (4) provide intuition for how these methods operate and why they are effective, and (5) solve challenging machine learning problems. Finally, Chapter 7 concludes this thesis with a discussion of future work.

1.3 Summary of contributions

The main contribution of this thesis is to characterize a general class of constrained optimization problems to which Bayesian optimization might be applied, and then to introduce methods for solving such problems. In particular, the general class of problems is described by considering decoupled constraints, with the coupled case considered by previous work being a special case of this more general framework. The main method is PESC, an acquisition function that provides the separate contributions to the expected information gain from each of the possible black-box functions that we might evaluate, thus enabling its application to all varieties of decoupled problems. Other contributions include formulating constrained Bayesian optimization problems with probabilistic constraints, defining the behavior of EIC when no feasible region has been found, extending EIC to the decoupled scenario, extending PESC to the bounded rationality setting, and providing empirical evaluations of these methods.

1.4 Relationship to published work

Chapter 3 and Section 6.1 of this thesis approximately correspond to the following publication:

Michael A. Gelbart, Jasper Snoek, Ryan P. Adams. Bayesian Optimization with Unknown Constraints. In *Uncertainty in Artificial Intelligence*, pages 250-259, 2014.

Sections 4.1 and 6.2 of this thesis approximately correspond to the following publication:

José Miguel Hernndez-Lobato[1], Michael A. Gelbart[1], Matthew W. Hoffman, Ryan P. Adams, Zoubin Ghahramani. Predictive Entropy Search for Bayesian Optimization with Unknown Constraints. In *International Conference on Machine Learning*, 2015.

[1] Authors contributed equally.

Chapter 2

Background and Related Work

This chapter describes the background and related work on which this thesis builds. Section 2.1 gives an overview of Bayesian optimization. Section 2.2 introduces the primary modeling tool used in this thesis, the Gaussian process. Section 2.3 discusses modeling issues that arise in the presence of black-box constraints. Section 2.4 introduces some popular acquisition functions for unconstrained Bayesian optimization. Section 2.5 discusses some meta-approaches from non-Bayesian optimization for applying unconstrained methods to constrained problems. Existing acquisition functions for constrained Bayesian optimization are reviewed in Section 2.6. Sections 2.7 and 2.8 discuss two extensions to Bayesian optimization, namely Bayesian optimization with parallel function evaluations and Bayesian optimization with multiple objectives or tasks, respectively. Section 2.9 introduces related research areas in other fields, many of which contain methods similar in spirit to Bayesian optimization. Section 2.10 discusses some limitations of Bayesian optimization. Finally, Section 2.11 provides a list of some open-source Bayesian optimization software.

2.1 Overview of Bayesian optimization

Bayesian optimization is a method for performing global optimization of black-box objective functions that are expensive to evaluate and possibly noisy. Bayesian optimization proceeds by iteratively developing a global statistical model of the objective function. We begin with a prior

Algorithm 1 Bayesian optimization for unconstrained problems.

1: **Inputs:** objective f, acquisition function α, search space $\mathcal{X}$, model $\mathcal{M}$, initial design $\mathcal{D}$
2: **repeat**
3: Fit the model $\mathcal{M}$ to the data $\mathcal{D}$
4: Maximize the acquisition function: $\hat{\mathbf{x}} = \arg\max_{\mathbf{x}\in\mathcal{X}} \alpha(\mathbf{x}, \mathcal{M})$
5: Evaluate the function: $\hat{y} = f(\hat{\mathbf{x}})$
6: Add the new data to the data set: $\mathcal{D} = \mathcal{D} \cup \{(\hat{\mathbf{x}}, \hat{y})\}$
7: **until** termination condition is met
8: **Output:** the recommendation $\mathbf{x}_\star = \arg\min_{\mathbf{x}\in\mathcal{X}} \mathbb{E}_{\mathcal{M}}\left[f(\mathbf{x})\right]$

distribution over objective functions and a likelihood function which encodes our assumptions about the noise present in our observations of the objective function. Then, at each iteration, a posterior distribution is computed by conditioning on the previous evaluations of the objective function, treating them as observations in a Bayesian nonlinear regression. This is often achieved with Gaussian processes, which are described in Section 2.2. An *acquisition function* is then used to map beliefs about the objective function to a measure of how promising each location in the input space is, if it were to be evaluated next. The goal is then to find the input that maximizes the acquisition function, and select it for function evaluation. Once the function evaluation is complete, the new observation (a single value in the unconstrained case) is added to the data set and we begin the next iteration. The termination condition is often a maximum elapsed time for the entire optimization procedure, or a maximum number of function evaluations. This procedure is illustrated in Algorithm 1. Additional reference material and perspectives on Bayesian optimization can be found in, for example, Brochu et al. (2010b) and Lizotte (2008).

One important class of acquisition functions is *myopic* acquisition functions. A myopic acquisition function considers the current action to be the final action; it makes no provisions for the future. One could also call this a *greedy* acquisition function. An example of a myopic acquisition function is Expected Improvement (Section 2.4.1). Some work has been done on non-myopic acquisition functions (e.g., Ginsbourger and Riche, 2010a), but in general this problem is extremely difficult. Chapter 5 will introduce another category of acquisition functions, *separable* acquisition functions. In some literature, acquisition functions are called *infill sampling criteria*.

Maximizing the acquisition function is itself an optimization problem, but typically it is one that does not possess the challenging properties (A) to (C) discussed in Chapter 1. The main

challenge of maximizing the acquisition function is that it is still a global optimization problem (property (D)). Loosely speaking, Bayesian optimization methods are well-suited to a problem only if optimizing the acquisition function is much easier than the original optimization problem of interest, since we must optimize the acquisition function many times as we solve the original problem.

2.2 Modeling: Gaussian processes

Gaussian processes (GPs) are often used to model the unknown objective function (and, when present, constraint functions) in Bayesian optimization. A GP is a generalization of the multivariate normal distribution to infinite length vectors or functions, and is specified by its positive definite covariance kernel function $\kappa(\mathbf{x}, \mathbf{x}')$ and, optionally, a mean function $\mu_0(\mathbf{x})$. By combining the GP, a prior over unknown objective functions, with a likelihood function, we can compute a posterior distribution over unknown objective functions. Computing a normalized posterior requires integrating over all possible functions; amazingly, this can be done tractably with a GP for Gaussian likelihood functions. Thus, GPs allow us to condition on observed data and tractably compute the posterior distribution of the model for any finite number of query points. In particular, if X is the design matrix containing the locations of N collected data, $\mathbf{y}$ is a vector of the corresponding N observations, X' is a matrix containing test points, $K(X, X')$ is a matrix containing the kernel function κ applied to the pairs of inputs in X and X', and we assume $\mu_0(\mathbf{x}) = 0$ for simplicity, then the predictive distribution at any finite set of test points X' follows a multivariate normal distribution with mean μ and covariance matrix Σ given by

$$\begin{aligned}\mu(X') &= K(X', X)\left[K(X, X) + \nu^2\mathbb{I}\right]^{-1}\mathbf{y} \\ \Sigma(X') &= K(X', X') - K(X', X)\left[K(X, X) + \nu^2\mathbb{I}\right]^{-1}K(X, X')\end{aligned} \tag{2.1}$$

where ν^2 is the variance of the i.i.d. Gaussian noise.

Eq. (2.1) has a number of interesting properties. One is that the posterior predictive covariance matrix Σ does not depend on the observation values $\mathbf{y}$. Intuitively, this means that one's

posterior uncertainty depends only on where one has collected data, not on the actual observation values. Another interesting property is that the computation in Eq. (2.1) involves inverting the $N \times N$ matrix $K(X, X) + \nu^2 \mathbb{I}$. Because matrix inversion (or computing the Cholesky decomposition) requires $\mathcal{O}(N^3)$ operations, this step is the computational bottleneck of GP regression as N grows large. This issue is discussed further in Section 7.2.1. Thirdly, a consequence of the multivariate normal posterior predictive distribution is that the marginal distribution at any single point is a univariate Gaussian whose mean and variance can be computed using Eq. (2.1). This property often leads to efficient acquisition function computations in Bayesian optimization, such as a closed form expression for Expected Improvement (Section 2.4.1). Gaussian process regression is also called *kriging*. For an in-depth treatment of GPs for machine learning, see Rasmussen and Williams (2006).

2.2.1 GP kernels

The assumptions encoded in the GP prior depend on the kernel function κ. The kernel function is a positive definite function of two inputs and, intuitively, tells us how the correlation of the function value at two inputs $\mathbf{x}$ and $\mathbf{x}'$ depend on $\mathbf{x}$ and $\mathbf{x}'$. Often, we restrict ourselves to the class of *stationary* kernels. A stationary kernel is one whose value depends on $\mathbf{x}$ and $\mathbf{x}'$ only through their difference $\mathbf{x} - \mathbf{x}'$. Intuitively, this means that the structure of the function is translation invariant. For recent work on modeling non-stationary functions with GPs for Bayesian optimization, see Snoek et al. (2014).

Two popular kernels that we will focus on here are the squared exponential kernel and the Matérn 5/2 kernel. The squared exponential kernel corresponds to the assumption that the function being modeled is infinitely differentiable, and takes the form

$$\kappa_{\mathrm{SE}}(\mathbf{x}, \mathbf{x}') = \sigma^2 \exp\left(-\frac{1}{2} r^2(\mathbf{x}, \mathbf{x}')\right) \tag{2.2}$$

where $r^2(\mathbf{x}, \mathbf{x}') = \|M(\mathbf{x} - \mathbf{x}')\|_2^2$, $M = \mathrm{diag}(1/\ell_1, \ldots, 1/\ell_D)$ for some length scale parameters $\{\ell_d\}$, and σ is an amplitude parameter. The Matérn 5/2 kernel corresponds to the assumption that

the function being modeled is twice differentiable, and takes the form

$$\kappa_{\mathrm{M52}}(\mathbf{x}, \mathbf{x}') = \sigma^2 \left(1 + \sqrt{5r^2(\mathbf{x}, \mathbf{x}')} + \frac{5}{3}r^2(\mathbf{x}, \mathbf{x}')\right) \exp\left(-\sqrt{5r^2(\mathbf{x}, \mathbf{x}')}\right) \tag{2.3}$$

where r and σ are defined as above.

One may also wish to define kernels over categorical or integer inputs for Bayesian optimization. This is a relatively unexplored area for GP-based Bayesian optimization. However, other models such as random forests for Bayesian optimization (Hutter et al., 2011) can naturally handle such parameters.

2.2.2 Learning the kernel hyperparameters

This kernels above have $D + 1$ hyperparameters in D dimensions: one characteristic length scale per dimension, and an overall amplitude σ. One can take a variety of approaches to learning these hyperparameters. The simplest method is to fix the hyperparamters *a priori*. The problem with this approach is that a mismatch of hyperparameters and the data can lead to very poor performance. For example, if the length scales are set much too large, then the GP prior will overlook the higher frequency variations in the true function; on the other hand, if the length scales are too small, the GP will fail to generalize across meaningful distances.

A more sophisticated approach is to learn the kernel hyperparamters from the data. This can be achieved by maximizing the marginal likelihood of the GP given the kernel hyperparameters. The marginal likelihood of a GP is given by

$$\log p(\mathbf{y}|X, \theta, \nu) = -\frac{1}{2}\mathbf{y}^\top \left[\kappa_\theta(X, X) + \nu^2\mathbb{I}\right]^{-1}\mathbf{y} - \frac{1}{2}\log\det\left[\kappa_\theta(X, X) + \nu^2\mathbb{I}\right] - \frac{N}{2}\log 2\pi \tag{2.4}$$

where θ is a vector containing the kernel hyperparameters and we show the explicit dependence of κ on θ by writing it as κ_θ. One can then perform maximum likelihood estimation by maximizing this quantity with respect to the hyperparameters θ. This is typically an optimization problem with linear constraints since, for example, the length scales and amplitude parameter must be non-negative. It is possible to analytically compute the gradient of Eq. (2.4) and use a gradient-based optimization scheme, although this is not always necessary for good performance

(Martinez-Cantin, 2014).

An even more sophisticated approach is a fully Bayesian treatment of the kernel hyperparamters. This is achieved by placing a prior on these hyperparameters and marginalizing them out. This marginalization can be performed using Markov chain Monte Carlo (MCMC) methods such as slice sampling (Neal, 2000); this is the approach taken in Snoek et al. (2012).

2.3 Modeling: Constraints

This thesis focuses exclusively on optimization problems with inequality constraints (Eq. (1.2)). However, constraint observations can arise in several different forms. In some problems, constraint function evaluations return a continuous value that is well modeled by a Gaussian likelihood. In these cases, we can model the constraint with an independent GP in the same manner as the objective. Below we describe two more complicated cases of constraint observations through two examples.

2.3.1 Output transformations for constraint function observations

Consider optimizing some property of a computer program such that its running time $\tau(\mathbf{x})$ must not exceed some maximum value $\tau_{\max}$. Because $\tau(\mathbf{x})$ is a measure of time, it is nonnegative for all $\mathbf{x}$ and thus not well-modeled by a GP prior. A reasonable solution is to model time in logarithmic units. In particular, we can define $c(\mathbf{x}) = \log \tau_{\max} - \log \tau(\mathbf{x})$, so that the condition $c(\mathbf{x}) \geq 0$ corresponds to our constraint condition $\tau(\mathbf{x}) \leq \tau_{\max}$. We can then place a GP prior on $c(\mathbf{x})$, and use a Gaussian likelihood for observations of $c(\mathbf{x})$. In this way, we retain the closed-form inference procedures described in Section 2.2. For every problem, a transformation of the output space implies particular assumptions about the original variables. For example, the choice to model observation noise in $c(\mathbf{x})$ with a Gaussian distribution implies that we are assuming a log-normal noise distribution for the original variable τ.

2.3.2 Boolean observations and Gaussian process classification

Another common case is that in which the black-box constraint function only returns a Boolean value indicating whether or not the constraint was satisfied. This type of observation is less informative than the continuous type discussed above, because the observations do not provide a clue about how close to the constraint boundary one is. For this reason, it is important not to artificially threshold a continuous constraint observation to a Boolean one; in fact, this makes both modeling and optimization more difficult.

As an example, consider the problem discussed in Snoek (2013) of optimizing the hyperparameters of a neural network, subject to the difficulty that for certain hyperparameters the training will diverge and no objective value can be reported. The notion of whether or not training diverged for a particular input $\mathbf{x}$ can be captured with a Boolean value. Clearly, Boolean values are not well described by a Gaussian likelihood, and therefore the standard GP machinery of Section 2.2 cannot be used. In order to model this type of constraint with a GP, we introduce a real-valued latent function $c(\mathbf{x})$ as in Section 2.3.1, and place a GP prior on $c(\mathbf{x})$. However, unlike in Section 2.3.1, in the Boolean case we cannot meaningfully map the domain $\{0, 1\}$ to $\mathbb{R}$ and use a Gaussian likelihood in the transformed space. Instead, we can work with, for example, the binomial likelihood function to compute the posterior distribution over $c(\mathbf{x})$. Upon doing so, the closed form posterior predictive equations (Section 2.2) can not longer be applied. Approximate inference methods such as variational methods, Expectation Propagation, or sampling can instead be used to approximate the posterior; this is called GP classification. GP classification has been used for constraint modeling in Bayesian optimization in e.g., Gramacy and Lee (2011); Tesch et al. (2013); Snoek (2013). See Rasmussen and Williams (2006, §3) for an in-depth treatment of GP classification.

2.3.3 Hidden constraints

In the above neural network example, the constraint is not an explicit part of the problem formulation but arises when the objective cannot be evaluated. Such constraints are called *hidden constraints* (Conn et al., 2009). Some black-box constrained optimization problems possess hidden constraints; in other cases, the objective function values are available regardless

of whether or not the constraints are satisfied. The methods presented in this thesis can be applied in both of these scenarios.

2.4 Unconstrained acquisition functions

This section describes acquisition functions for unconstrained Bayesian optimization that are relevant to this thesis.

2.4.1 Expected improvement

The Expected Improvement (EI) (Mockus et al., 1978; Jones et al., 1998) acquisition function measures the expected amount by which observing $f(\mathbf{x})$ leads to *improvement* over some target η:

$$\alpha_{\mathrm{EI}}(\mathbf{x}) = \mathbb{E}_y[\max(0, \eta - y)] = \int_{-\infty}^{\infty} \max(0, \eta - y) p(y \,|\, \mathbf{x}) \, \mathrm{d}y \tag{2.5}$$

where $p(y \,|\, \mathbf{x})$ is the posterior predictive marginal distribution of the objective function at $\mathbf{x}$. EI manages the local vs. global search trade-off because it can be made large both by high variance (global search) or low mean (local search). Typically, the target η is set to be the current recommendation or *incumbent*; hence, the name Expected Improvement refers literally to the amount by which one expects to improve over the function value at the current best solution. In unconstrained Bayesian optimization, η may be defined as the minimum over previous observations, the minimum of the expected value of the objective under the model, or some biased version thereof; see Lizotte (2008, §3.2) for further discussion.

When the predictive distribution under the model is Gaussian, EI has a closed-form expression (Jones, 2001):

$$\alpha_{\mathrm{EI}}(\mathbf{x}) = \sigma(\mathbf{x}) \left(z(\mathbf{x}) \Phi\left(z(\mathbf{x})\right) + \phi\left(z(\mathbf{x})\right) \right) \tag{2.6}$$

where $z(\mathbf{x}) \equiv \frac{\eta - \mu(\mathbf{x})}{\sigma(\mathbf{x})}$, $\mu(\mathbf{x})$ is the posterior predictive mean at $\mathbf{x}$, $\sigma^2(\mathbf{x})$ is the posterior predictive marginal variance at $\mathbf{x}$, $\Phi(\cdot)$ is the standard normal cumulative distribution function (CDF), and $\phi(\cdot)$ is the standard normal probability density function (PDF). Eq. (2.6) is differentiable and

fast to compute, and can therefore be maximized with standard gradient-based optimizers. EI is a popular method backed both by theoretical guarantees (Bull, 2011) and by proven effectiveness in practice (e.g., Snoek et al., 2012).

Bull (2011) studies the theoretical properties of EI. Bull shows that when the GP kernel hyperparameters are estimated from data via maximum likelihood, EI may not converge. He then proposes a method of estimating the kernel hyperparameters that is guaranteed to converge. Non-convergence can also be circumvented by using a so-called ϵ-greedy meta-strategy (Sutton and Barto, 1998). In an ϵ-greedy approach, one takes an action based on a strategy of choice (such as EI) with probability $1-\epsilon$ and selects an action at random with probability ϵ. Thus, for any $\epsilon > 0$, the algorithm is guaranteed to eventually converge (Bull, 2011), but if ϵ is sufficiently small then performance is not significantly hindered by the random actions.

EI has also been extended to the *generalized EI* (Schonlau et al., 1998), as follows:

$$\alpha_{\text{EI-g}}(\mathbf{x}) = \mathbb{E}\left[\max(0, \eta - y)^g\right] = \int_{-\infty}^{\infty} \max(0, \eta - y)^g p(y \mid \mathbf{x}) \, \mathrm{d}y \tag{2.7}$$

The new parameter $g \in \mathbb{Z}^+$ controls the tradeoff between global search and local search: the larger g is, the more the algorithm is biased towards global search. Interestingly, setting $g = 0$ returns the Probability of Improvement (PI) acquisition function, which is simply

$$\alpha_{\text{PI}}(\mathbf{x}) = \Pr(y < \eta | \mathbf{x}) = \Phi(z(\mathbf{x})) \tag{2.8}$$

for the definition of $z(\mathbf{x})$ given above. It is well-known that the PI algorithm tends too much towards local search (Brochu et al., 2010a; Lizotte, 2008). This is consistent with the fact that PI is a special case of generalized EI with $g = 0$. Schonlau et al. (1998) also derive a closed-form expression for the generalized EI.

2.4.2 GP-UCB

The Gaussian Process Upper Confidence Bound (GP-UCB) acquisition function (Srinivas et al., 2010) is defined as

$$\alpha_{\mathrm{UCB}}(\mathbf{x}) = -\mu(\mathbf{x}) + \beta_n^{1/2}\sigma(\mathbf{x})\,, \tag{2.9}$$

where μ and σ^2 are the posterior predictive marginal GP mean and variance, respectively, and β_n is a constant. GP-UCB explicitly balances local and global search by favoring regions with low posterior mean and high posterior variance. Srinivas et al. (2010) show that, assuming the true function is a sample from a GP with known kernel hyperparameters, or the function has low complexity as measured under a reproducing kernel Hilbert space norm, the GP-UCB algorithm achieves sublinear regret for an appropriate choice of β_n. de Freitas et al. (2012) build on this work and show that in the deterministic (noise-free) case, even faster convergence rates can be achieved.

2.4.3 Thompson Sampling

Thompson sampling (TS) (Thompson, 1933), also known as *probability matching*, is a randomized acquisition strategy in which the probability that $\hat{\mathbf{x}}$ is chosen for the next evaluation is proportional to the probability (according to the model) that $\hat{\mathbf{x}}$ is the utility-maximizing action. However, instead of marginalizing out the uncertainty, TS works by randomly sampling the unknown quantities and then maximizing the utility given this sample. In the bandit setting (Section 2.9.3), this corresponds to sampling the unknown rewards from their probability distributions and then selecting the arm with the largest sampled reward. In the context of GP-based Bayesian optimization, TS corresponds to sampling the objective function from the GP posterior and then minimizing this sample (one could also think of this procedure as randomly sampling the acquisition function from the GP posterior predictive distribution, multiplying it by -1, and then maximizing it). TS has been shown to work well in practice (Chapelle and Li, 2011) and is simple to implement. TS is related to PES (Section 2.4.7) and PESC (Chapter 4); this relationship is explored in Section 4.3.4.

2.4.4 The information-based acquisition function

The goal of the information-based acquisition function is to, at each iteration, maximize the expected *information gain* about the solution to the optimization problem. This goal is achieved by considering the distribution over the location of the solution given the data, namely $p(\mathbf{x}_\star \mid \mathcal{D})$. This is a D-dimensional probability distribution when the search space has D dimensions. As a proxy for solving the optimization problem $\min_{\mathbf{x}} f(\mathbf{x})$, one can consider minimizing the differential entropy of this distribution, $\mathrm{H}[\mathbf{x}_\star \mid \mathcal{D}]$. As the entropy decreases, our uncertainty about the location of the minimizer $\mathbf{x}_\star$ decreases.

Here we restrict ourselves to greedy information-based acquisition functions, also called *stepwise uncertainty reduction* (SUR) (Geman and Jedynak, 1996) methods, which aim to, at each iteration, reduce the entropy $\mathrm{H}[\mathbf{x}_\star \mid \mathcal{D}]$ as much as possible in expectation. In symbols,

$$\alpha_{\mathrm{SUR}}(\mathbf{x}) = \mathrm{H}[\mathbf{x}_\star \mid \mathcal{D}] - \mathbb{E}_{y \mid \mathcal{D}, \mathbf{x}}\left[\mathrm{H}[\mathbf{x}_\star \mid \mathcal{D} \cup \{(\mathbf{x}, y)\}]\right] . \tag{2.10}$$

In practice, the first term in Eq. (2.10) can be ignored because it does not depend on $\mathbf{x}$ and therefore will not affect the suggestion $\hat{\mathbf{x}} = \arg\max_{\mathbf{x}} \alpha_{\mathrm{SUR}}(\mathbf{x})$.

There are two major challenges of computing α_{SUR}. The first is computing the entropy $\mathrm{H}[\mathbf{x}_\star \mid \mathcal{D} \cup \{(\mathbf{x}, y)\}]$, which requires computing $p(\mathbf{x}_\star \mid \mathcal{D} \cup \{(\mathbf{x}, y)\})$ as an intermediate step. The second is computing the expectation of this entropy over $p(y \mid \mathbf{x})$. The following sections introduce three different methods for approximating Eq. (2.10).

2.4.5 Information-based acquisition functions: IAGO

Villemonteix et al. (2009) address the challenges described above using Monte Carlo sampling. In particular, the expectation in Eq. (2.10) is computed by Monte Carlo sampling of y from $p(y \mid \mathbf{x})$, which is a univariate Gaussian distribution with known mean and variance when the underlying model is a GP. Next, given $\mathbf{x}$ and a Monte Carlo sample of y, $p(\mathbf{x}_\star \mid \mathcal{D} \cup \{(\mathbf{x}, y)\})$ is estimated by Monte Carlo on a grid. In particular, we first draw function samples $f \sim p(f \mid \mathcal{D} \cup \{(\mathbf{x}, y)\})$ on the grid and, for each sample, we note its minimum. A histogram of these minima then forms the estimate of $p(\mathbf{x}_\star \mid \mathcal{D} \cup \{(\mathbf{x}, y)\})$. Finally, the (discrete) entropy of this empirical distribution

Algorithm 2 The IAGO algorithm (Villemonteix et al., 2009), conceptual implementation

1: **Inputs:** data set $\mathcal{D}$, candidate $\mathbf{x}$, # of y samples N_1, # of f samples N_2, grid size N_3
2: **for** $i = 1$ **to** N_1 **do**
3: sample $y_i \sim p(y|\mathbf{x}, \mathcal{D})$
4: initialize $p_{\text{empirical}} = \mathbf{0}$ on grid of size N_3
5: **for** $j = 1$ **to** N_2 **do**
6: sample $f_j \sim p(f \mid \mathcal{D} \cup \{(\mathbf{x}, y_i)\})$ on grid (multivariate normal)
7: compute the grid index of the minimizer: $k = \arg\min f_j$
8: record the result: $p_{\text{empirical}}[k] = p_{\text{empirical}}[k] + 1$
9: **end for**
10: normalize the empirical distribution: $p_{\text{empirical}}[k] = p_{\text{empirical}}[k]/N_2$ for $k = 1, \ldots, N_3$
11: compute the entropy $\mathrm{H}_i = -\sum_{k=1}^{N_3} p_{\text{empirical}}[k] \log p_{\text{empirical}}[k]$
12: **end for**
13: compute the average entropy $\mathrm{H} = \frac{1}{N_1} \sum_{i=1}^{N_1} \mathrm{H}_i$
14: **Output:** $\alpha_{\text{SUR}}(\mathbf{x}) = -\mathrm{H}$

is computed; this is an approximation of the differential entropy that appears in Eq. (2.10). This procedure is illustrated in Algorithm 2. Villemonteix et al. (2009) dub this method the Informational Approach to Global Optimization (IAGO) algorithm. Note that computing the suggestion $\hat{\mathbf{x}}$ requires maximizing the acquisition function over $\mathbf{x}$ and thus performing the entire computation in Algorithm 2 for many values of $\mathbf{x}$.

Algorithm 2 is a conceptual illustration of IAGO. The computational cost of this method is $\mathcal{O}(N_3^3 + N_1 N_2 N_3^2)$, where N_1 is the number of samples of y, N_2 is the number of samples of f in estimating $p(\mathbf{x}_\star \mid \mathcal{D} \cup \{(\mathbf{x}, y)\})$, and N_3 is the size of the grid on which samples of f are drawn. This running time is dominated by the $\mathcal{O}(N_3^2)$ matrix-vector product at line 6, which is repeated $N_1 N_2$ times, plus the $\mathcal{O}(N_3^3)$ required to compute the Cholesky decomposition of an $N_3 \times N_3$ kernel matrix. However, this method can be sped up to $\mathcal{O}(N_3^3 + N_2 N_3^2 + N_1 N_2 N_3)$ using a series of tricks described in Appendix B.11. In practice, the fast implementation can be an order of magnitude faster or more.

2.4.6 Information-based acquisition functions: entropy search

Entropy Search (ES) (Hennig and Schuler, 2012) is a method for approximating the SUR acquisition function that uses Expectation Propagation (EP) (Minka, 2001a) (Appendix A.1) instead of Monte Carlo to approximate the entropy in Eq. (2.10). Like IAGO, this approach also relies on

a grid but, unlike IAGO, the EP approximation in ES results in a differentiable acquisition function; this makes maximizing the acquisition function easier. Rather than using a fixed uniform or space-filling grid, Hennig and Schuler (2012) use shrinking rank slice sampling (Thompson and Neal, 2010) to sample the grid from the EI function, treating it as an unnormalized probability distribution. This heuristic tends to lead to a higher density of grid points in areas where $p(\mathbf{x}_\star)$ is higher, which is a desirable property when estimating the entropy of $p(\mathbf{x}_\star)$ on a grid. Hennig and Schuler (2012) show performance gains compared to EI and other methods on two-dimensional problems, but conjecture that IAGO may perform better than ES in higher-dimensional settings.

2.4.7 Information-based acquisition functions: predictive entropy search

Predictive Entropy Search (PES) (Hernández-Lobato et al., 2014) is another method for approximating the uncertainty reduction acquisition function in Eq. (2.10). A key insight is that we can rewrite Eq. (2.10) by noting that it is a mutual information. The mutual information between the random variables A and B, denoted $\mathrm{I}(A; B)$, is symmetric in its arguments and is given by

$$\mathrm{I}(A; B) = \mathrm{I}(B; A) = \mathrm{H}(A) - \mathbb{E}_B \left[\mathrm{H}(A \,|\, B)\right] \tag{2.11}$$

where $\mathrm{H}(A \,|\, B)$ is entropy of the conditional probability distribution $p(A = a \,|\, B = b)$. Using this definition, we can rewrite Eq. (2.10) as

$$\alpha_{\mathrm{PES}}(\mathbf{x}) = \alpha_{\mathrm{SUR}}(\mathbf{x}) = \mathrm{I}\left(\mathbf{x}_\star; y \,|\, \mathcal{D}, \mathbf{x}\right) . \tag{2.12}$$

Because the mutual information is symmetric in its arguments, we can now apply Eq. (2.11) again, but with the arguments flipped:

$$\alpha_{\mathrm{PES}}(\mathbf{x}) = \mathrm{H}\left[y \,|\, \mathcal{D}, \mathbf{x}\right] - \mathbb{E}_{\mathbf{x}_\star \,|\, \mathbf{x}, \mathcal{D}} \left[\mathrm{H}\left[y \,|\, \mathcal{D}, \mathbf{x}, \mathbf{x}_\star\right]\right] . \tag{2.13}$$

Hernández-Lobato et al. (2014) argue that Eq. (2.13) is easier to approximate than Eq. (2.10) because it only involves entropies of the one-dimensional random variable y rather than the D-dimensional random variable $\mathbf{x}_\star$. Indeed, because the posterior predictive marginal distribution of a GP is univariate Gaussian, the left-hand term is simply the differential entropy of a Gaussian distribution with variance s^2, namely

$$\mathrm{H}\left[\mathcal{N}(\mu, s^2)\right] = \frac{1}{2}\log(2\pi e s^2)\,. \tag{2.14}$$

The core of the PES method is an approximation to the right-hand term in Eq. (2.13). The expectation over $\mathbf{x}_\star$ is handled with Monte Carlo sampling of the global optimizer $\mathbf{x}_\star$. As in IAGO, this is achieved by sampling the unknown function f and then numerically finding the optimum. However, unlike in IAGO, Hernández-Lobato et al. (2014) do not sample $\mathbf{x}_\star$ on a grid. Rather, the GP kernel is approximated with a finite basis function approximation (Rahimi and Recht, 2007), yielding a linear approximation to the GP sample that can be computed with running time that is independent of the number of collected data. This significant speedup enables practical numerical optimization of the sample to find its minimizer with high accuracy, not constrained to a grid.

Given a sample of $\mathbf{x}_\star$ obtained in this manner, it remains to approximate the conditional entropy $\mathrm{H}\left[y \,|\, \mathcal{D}, \mathbf{x}, \mathbf{x}_\star\right]$. This is achieved with EP. Here, we are interested in the distribution $p(y \,|\, \mathcal{D}, \mathbf{x}, \mathbf{x}_\star)$; in words, we want to reason about the distribution over y given that we know the global minimizer of f is $\mathbf{x}_\star$. The conditioning on $\mathbf{x}_\star$ is handled approximately by enforcing the following weaker conditions: that $\mathbf{x}_\star$ is a a *local* minimum (i.e., $\nabla f(\mathbf{x}_\star) = \mathbf{0}$ and $\nabla^2 f(\mathbf{x}_\star)$ is negative definite), and that $f(\mathbf{x}_\star)$ is smaller than $f(\mathbf{x})$ and all past observations. These conditions are encoded as factors and incorporated into EP, yielding a Gaussian approximation to $p\left(y \,|\, \mathcal{D}, \mathbf{x}, \mathbf{x}_\star\right)$. One can then apply Eq. (2.14) to compute the approximate entropy.

Hernández-Lobato et al. (2014) show that PES is a better approximation to Eq. (2.10) than ES, and also that PES outperforms both ES and EI in several experiments. One advantage of PES over ES is that one can jointly average over the GP hyperparameter samples and the $\mathbf{x}_\star$ samples. Thus, one can marginalize out the GP hyperparameters without significant additional

cost. By comparison, the entire ES approximation must be performed again for each sample of the GP hyperparameters, causing GP hyperparameter marginalization with ES to be extremely expensive (Hennig and Schuler, 2012).

2.4.8 Discussion on information-based acquisition functions

Because information-based methods are based on the minimizer location $\mathbf{x}_\star$ (Eq. (2.10)), they are in a sense blind to the output units of the optimization problem. This does not mean that they are entirely unaffected by any monotonic transformation of the output space, but nonetheless this notion may be worrying from a philosophical perspective since the units of the output space are meant to represent the user's utility (see Section 7.2.5). When making this criticism of information-based acquisition functions, it is important to note that while EI is a principled acquisition function at the last iteration, it is myopic is not necessarily well-motivated across many iterations. Thus, short of solving an infinite POMDP and forming a policy covering the entire optimization, all methods discussed here are heuristics.

2.5 Unconstrained methods for constrained optimization

Before discussing acquisition functions for constrained Bayesian optimization, here we introduce three meta-methods from classical (non-Bayesian) optimization for transforming unconstrained methods to constrained methods.

2.5.1 Barrier methods

In *barrier methods*, the iterates are prevented from leaving the feasible region by the addition of a *barrier function* to the objective. The barrier function causes the objective to grow to infinity at the boundary of the feasible region. For example, adding $-\lambda \log c(\mathbf{x})$ to the objective function for some constant $\lambda \geq 0$ causes the objective to tend towards positive infinity when $c(\mathbf{x})$ approaches zero from the right, thus enforcing the constraint $c(\mathbf{x}) \geq 0$. As λ approaches zero, the solution to the barrier problem should approach the solution to the constrained problem, because the barrier will only have an effect exactly at the constraint boundary.

2.5.2 Penalty methods

Penalty methods also involve adding an extra term to the objective, but, in contrast to barrier methods, penalty methods allow the iterates to leave the feasible region. However, as optimization proceeds, the penalty is increased to infinity so that the iterates become feasible. A popular penalty function is the quadratic penalty $\frac{1}{2\rho}\min(0, c(\mathbf{x}))^2$ for some penalty parameter $\rho > 0$. When $c(\mathbf{x}) \geq 0$ (i.e., when the constraint is satisfied), the penalty function is zero and does not affect the objective. When $c(\mathbf{x}) < 0$ (i.e., when the constraint is violated), a positive penalty is added to the objective. As ρ approaches zero, the solution to the penalty problem should approach the solution to the constrained problem. Typically, the overall problem is solved by iteratively solving the unconstrained problem and then decreasing ρ towards zero by some constant factor such as $\frac{1}{2}$. Because the purpose of the penalty is to ensure feasibility, one may wish to only update the penalty parameter if the current iterate $\mathbf{x}^{(n)}$ is infeasible. Concretely, the new penalty parameter $\rho^{(n+1)}$ is given by:

$$\rho^{(n+1)} = \begin{cases} \rho^{(n)}, & \text{if } \mathbf{x}^{(n)} \text{ is feasible} \\ \frac{1}{2}\rho^{(n)}, & \text{otherwise} \end{cases}. \tag{2.15}$$

Under certain assumptions and penalty functions, the penalty parameter need not be increased to infinity, but only to some threshold, in order to guarantee convergence (Friedlander and Tseng, 2007). The penalty method can be applied to Bayesian optimization in several ways. Sasena et al. (2002) use the penalty method to directly penalize the acquisition function where the space is thought to be infeasible. Parr et al. (2010) add a penalty to the model, which indirectly influences the acquisition function.

2.5.3 Augmented Lagrangian methods

The *augmented Lagrangian* (AL) (Rockafellar, 1974) method is similar to the penalty method, but with the addition of one extra term into the objective. In AL, we convert a constrained

problem with objective $f(\mathbf{x})$ and constraints $\{c_k(\mathbf{x})\}$ to the following unconstrained problem:

$$\min_{\mathbf{x}} \; f(x) + \sum_{k=1}^{K} \left[\frac{1}{2\rho} \min(0, c_k(\mathbf{x}))^2 - \lambda_k c_k(\mathbf{x}) \right] \tag{2.16}$$

where $\rho > 0$ is a penalty parameter and the $\lambda_k \geq 0$ are approximate Lagrange multipliers. The first and third term represent the Lagrange multiplier formulation, and they are augmented by the second term, which is the quadratic penalty. Let $\mathbf{x}^{(n)}$ be the solution at iteration n with parameters $\rho^{(n)}$ and $\lambda_k^{(n)}$. Then, AL works by iteratively solving the unconstrained problem and updating the approximate Lagrange multipliers by

$$\lambda_k^{(n+1)} = \max\left(0, \lambda_k^{(n)} - \frac{c_k(\mathbf{x}^{(n)})}{\rho^{(n)}}\right) \text{ for } k = 1, \ldots, K \tag{2.17}$$

and updating the penalty parameter ρ as in Eq. (2.15). See Bertsekas (1996); Nocedal and Wright (2006) for an in-depth discussion of these methods.

2.6 Constrained acquisition functions

Below we discuss previous approaches to Bayesian optimization with black-box constraints, many of which are variants of EI. Note that Azimi et al. (2010b); Azimi (2012) study Bayesian optimization with "constrained experiments" but use this phrase to mean something different from the meaning here: in their set-up, the term "constrained" refers to the fact that one cannot specify an exact $\mathbf{x}$ at which the evaluate the function, but rather a region in which the evaluation will take place.

2.6.1 Expected improvement with constraints

An intuitive extension to EI in the presence of constraints is to define improvement as only occurring when the constraints are satisfied. Because we are uncertain about the values of the constraints, we must extend our expectation to include this uncertainty. This results in what

we call Expected Improvement with Constraints (EIC):

$$\alpha_{\text{EIC}} = \alpha_{\text{EI}}(\mathbf{x}) \prod_{k=1}^{K} \Pr(c_k(\mathbf{x}) \geq 0)\,, \tag{2.18}$$

where we can separate the constraint satisfaction probabilities due to assumed independence. Because each probability $\Pr(c_k(\mathbf{x}) \geq 0)$ is simply a Gaussian CDF, EIC has a closed form expression that is very similar to the closed form for EI given in Eq. (2.6):

$$\alpha_{\text{EI}}(\mathbf{x}) = \sigma_f(\mathbf{x}) \left(z_f(\mathbf{x})\Phi\left(z_f(\mathbf{x})\right) + \phi\left(z_f(\mathbf{x})\right)\right) \prod_{k=1}^{K} \Phi\left(\frac{\mu_k(\mathbf{x})}{\sigma_k(\mathbf{x})}\right)\,, \tag{2.19}$$

where μ_k and σ_k^2 are the posterior predictive mean and variance for constraint k, μ_f and σ_f^2 are the same for the objective, and $z_f(\mathbf{x})$ is defined as $z(\mathbf{x})$ in Section 2.4.1, namely $z_f(\mathbf{x}) \equiv \frac{\eta - \mu_f(\mathbf{x})}{\sigma_f(\mathbf{x})}$. In the constrained case the incumbent η depends on the constraints. For example, it can be defined as the best observation in which all constraints are observed to be satisfied, or the minimum of the posterior mean such that all constraints are satisfied. The issue of setting η in the presence of constraints is discussed in more detail in Chapter 3.

EIC was initially introduced by Schonlau et al. (1998, §4) and has been revisited by Snoek (2013) and Gardner et al. (2014). Chapter 3 of this thesis extends this work to more general and complete EIC methods. Note that Snoek (2013) refers to EIC as a *constraint weighted acquisition function*. This should not be confused with the *weighted expected improvement* acquisition function introduced in Sobester et al. (2005), which is not related to constrained optimization. EIC is also called *constrained expected improvement* by Forrester et al. (2008).

2.6.2 Integrated expected conditional improvement

Gramacy and Lee (2011) propose an acquisition function called the integrated expected conditional improvement (IECI), defined as

$$\alpha_{\text{IECI}}(\mathbf{x}) = \int_{\mathcal{X}} \left[\alpha_{\text{EI}}(\mathbf{x}') - \alpha_{\text{EI}}(\mathbf{x}'|\mathbf{x})\right] h(\mathbf{x}')\mathrm{d}\mathbf{x}'\,. \tag{2.20}$$

Here, $\alpha_{\text{EI}}(\mathbf{x}')$ is the expected improvement at $\mathbf{x}'$, $\alpha_{\text{EI}}(\mathbf{x}'|\mathbf{x})$ is the expected improvement at $\mathbf{x}'$

given that the objective has been observed at $\mathbf{x}$ (but without making any assumptions about the observed value), and $h(\mathbf{x}')$ is an arbitrary density over $\mathbf{x}'$. In words, the IECI at $\mathbf{x}$ is the expected reduction in EI at $\mathbf{x}'$, under the density $h(\mathbf{x}')$, caused by observing the objective at $\mathbf{x}$. Gramacy and Lee use IECI for constrained Bayesian optimization by setting $h(\mathbf{x}')$ to the probability of satisfying the constraint. This formulation encourages evaluations that inform the model in places that are likely to satisfy the constraint. The motivation for IECI is that evaluating at an infeasible location may also provide useful information, and therefore one should consider improvement over the whole optimization domain.

2.6.3 Expected volume minimization

Picheny (2014) proposes to sequentially explore the location that most decreases the expected volume of the feasible region below the best feasible objective value η found so far. This quantity is given by integrating the product of the probability of improvement and the probability of feasibility. That is,

$$\alpha_{\mathrm{EEV}}(\mathbf{x}) = \int p[f(\mathbf{x}') \leq \eta] h(\mathbf{x}') d\mathbf{x}' - \int p[f(\mathbf{x}') \leq \min(\eta, f(\mathbf{x}))] h(\mathbf{x}') d\mathbf{x}' \,, \tag{2.21}$$

where, as in IECI, $h(\mathbf{x}')$ is the probability that the constraints are satisfied at $\mathbf{x}'$, and, as in IAGO (Section 2.4.5), we can ignore the first term on the right-hand side because it does not depend on $\mathbf{x}$. This method is limited by having to compute the integral in Eq. (2.21) over the entire domain, which is done numerically over a grid on $\mathbf{x}'$ (Picheny, 2014). The resulting acquisition function must then be globally optimized, which also requires a grid on $\mathbf{x}$. This is also the case for IECI.

2.6.4 Modeling an augmented Lagrangian

Gramacy et al. (2014) propose a combination of EI (Section 2.4.1) and the augmented Lagrangian method (Section 2.5.3) for constrained black-box optimization. In their method, Bayesian optimization is used to solve the unconstrained *inner* loop of the AL optimization. The authors point out that directly modeling the AL (Eq. (2.16)) leads to modeling difficulties because the

penalty may be very large and the max function can introduce kinks into the AL. Instead, they model the objective and constraints with independent GPs as in Snoek (2013); Gramacy and Lee (2011), and many other works including this thesis. They then consider two acquisition functions: the (negative) predictive mean, and EI. They derive closed-form expressions for the former. For the latter, they propose Monte Carlo sampling and also derive an analytic approximation to the true EI of the AL in the case of a single constraint by dropping the max in Eq. (2.16). Dropping the max is tantamount to penalizing both $c(\mathbf{x}) > 0$ and $c(\mathbf{x}) < 0$, forcing the iterates towards the constraint boundary. If the solution is known to be on the constraint boundary, this may be a reasonable approach. This method assumes that the evaluations of the constraints $c_1, \ldots, c_k$ are noise-free and focuses on the case in which the objective f is known; suggestions for extending to unknown f are provided.

2.7 Parallel Bayesian optimization

When doing Bayesian optimization one often wishes to run multiple function evaluations in parallel. This may be done either by submitting queries in batches, or by submitting them asynchronously. In the batch case, the goal is to select q queries for simultaneous evaluation; this requires optimizing a *multipoints* (Ginsbourger et al., 2010) acquisition function in $(q \times D)$ dimensions. In the asynchronous case, at each iteration we maximize the acquisition function to yield one suggestion $\hat{\mathbf{x}}$ given a set of *pending* queries $\{\mathbf{x}_j^{\text{pend}}\}$. Schonlau (1997, §5.3); Schonlau et al. (1998, §5) propose applying asynchronous methods to the batch case to avoid the $(q \times D)$-dimensional optimization, by iteratively selecting points and then assuming they have been submitted even if, in reality, they will all be submitted as a batch once all points are chosen.

One may question why any specialized method for parallel evaluations is needed at all. The reason is that, if no modifications are made to the acquisition function after a query is submitted, then maximizing the acquisition function will repeatedly return the same suggestion, and the same query will be redundantly performed. Therefore, the acquisition strategy must take into account the fact that some queries are pending. An exception to this is for non-deterministic acquisition strategies, for which making no modifications and relying on the stochasticity of the

algorithm may work sufficiently well (Bergstra et al., 2011). One such stochastic acquisition strategy is discussed in Section 2.4.3.

Ginsbourger et al. (2010) develop the multipoints (batch) EI, denoted q-EI, that is alluded to in Schonlau (1997). They develop an analytical form for 2-EI, and suggest an asynchronous approach with Monte Carlo sampling for $q > 2$. They also discuss two heuristic approaches for asynchronous parallel Bayesian optimization, the *kriging believer* and *constant liar*. These approaches fill in the missing (pending) observation values with the posterior predictive GP mean, or a constant value, respectively. The data set is then augmented with these "dummy" values for the purpose of choosing the next query location. Doing so decreases the variance at these locations and thus prevents redundant experiments from being performed. The kriging believer approach is used for GP-based Bayesian optimization in e.g., Bergstra et al. (2011, §6.1) and also in this thesis. Note that the kriging believer can be implemented efficiently because conditioning on an observation equal to the GP posterior predictive mean does not change this mean for any $\mathbf{x}$, and the covariance matrix can be updated with a rank-one update to the Cholesky decomposition of the kernel matrix (Appendix B.10.1).

Ginsbourger and Riche (2010b) introduce another approach, *Expected EI* (EEI). The double expectation in EEI comes from integrating out both the unknown query value y^{query} (as in regular EI) and also the unknown pending observation values $\{y_j^{\text{pend}}\}$. The authors propose to compute the expectation over the pending values using Monte Carlo samples of $\{y_j^{\text{pend}}\}$. This approach was independently proposed in Snoek et al. (2012). Note that EEI is not that same as the kriging believer because the improvement function is nonlinear and therefore

$$\mathbb{E}_{\text{pend}}\mathbb{E}_{\text{query}\,|\,\text{pend}} \max\left(0, \eta - y^{\text{query}}\right) \neq \mathbb{E}_{\text{query}} \max\left(0, \eta - \mathbb{E}_{\text{pend}}[y^{\text{query}}]\right) ,$$

where the left-hand side is EEI and the right-hand side is the kriging believer. In game theory, a similar notion to EEI exists called expected expected utility, in which the former expectation is over the uncertainty about one's utility function, and the latter expectation is over the uncertainty in the outcomes of particular actions (Boutilier, 2001).

Janusevskis et al. (2012) unify the batch (e.g., q-EI) and asynchronous (e.g., EEI) meth-

ods by considering the case of batch size λ with μ jobs currently pending; they name their method $\text{EI}^{(\mu,\lambda)}$. The $\text{EI}^{(\mu,\lambda)}$ acquisition function is a generalization of previous methods and, for example, reduces to standard EI when $\lambda = 1$ and $\mu = 0$. They point out that if the black-box function evaluations are fast compared to the optimization steps then pending queries are unimportant ($\mu = 0$), whereas if function evaluations are very slow then one can take an asynchronous approach ($\lambda = 1$), and that intermediate cases may call for intermediate (λ, μ) values. The authors also suggest a method for online tuning of the number of Monte Carlo samples of $\{y_j^{\text{pend}}\}$.

Clark (2012) considers an alternative approach to q-EI by pointing out that to maximize q-EI one only needs the gradient of q-EI, rather q-EI itself. By switching the expectation and gradient operators, they enable the sampling of stochastic gradient estimates of q-EI, leading to a multistart stochastic gradient ascent approach for maximizing the acquisition function. Finally, a promising fast method for computing q-EI approximately but accurately is introduced in Chevalier and Ginsbourger (2013).

Generally speaking, parallel Bayesian optimization methods are better than sequential methods in terms of wall-clock time (because they can do more queries per unit time) but worse than sequential methods in terms of total evaluations (because they must make decisions with less information at hand). Given this trade-off, Azimi et al. (2010a) try to find the batch policy that, in expectation, best matches a (more informed) sequential policy. They do this by actually simulating sequential policies where the experiment outcomes are Monte Carlo samples from the GP posterior predictive distribution. They then minimize a nearest neighbor objective between the batch and the sampled sequential policies, which is achieved by solving a k-means or k-mediod problem. Building on this work, Azimi et al. (2011, 2012) propose a method to dynamically increase the batch size if the next suggestion $\hat{\mathbf{x}}$ is determined to be mostly independent of the outcome of the current experiment. By doing this, they achieve significant gains with respect to wall-clock time without significantly compromising performance with respect to the number of function evaluations.

2.8 Bayesian optimization with multiple tasks or objectives

At each iteration of Bayesian optimization a suggestion $\hat{\mathbf{x}}$ must be selected from the continuous decision space $\mathcal{X}$. In some problems, there is an additional component to the decision space because there are multiple *tasks* to choose from. If the objective and constraints are decoupled (Chapter 5), then at each iteration one must choose both a location $\mathbf{x} \in \mathcal{X}$ and a task $t \in \mathcal{T}$, where $\mathcal{T}$ is the discrete set of possible tasks $\{f, c_1, \ldots, c_K\}$. More generally, a task could contain one or more objective functions, one or more constraint functions, or, in some cases, functions whose values do not contribute to the overall utility function at all but provide indirect information about the other functions. Multi-task Bayesian optimization is introduced in Swersky et al. (2013), with motivating examples such as minimizing k-fold cross-validation error of a machine learning model. Instead of evaluating all k folds at a given $\mathbf{x}$, the authors reformulate the problem as a multi-task problem in which each of the k folds is treated as its own task that can be evaluated independently. A multi-task GP is used to model the unknown objective function surface for each task, but also the correlations between tasks. Upon discovering that one of the k folds is more informative about the overall objective (the average of the k folds) than the others, one can increase efficiency by evaluating this fold more frequently. Multi-task Bayesian optimization is highly related to the work in this thesis because constraint decoupling (Chapter 5) is a case of multi-task optimization.

Zuluaga et al. (2013) propose the Pareto Active Learning (PAL) method for finding Pareto-optimal solutions when multiple objective functions are present and the input space is a discrete set. Their algorithm classifies each design candidate as either Pareto-optimal or not, and proceeds iteratively until all inputs are classified. The user may specify a confidence parameter determining the trade-off between the number of function evaluations and prediction accuracy.

Constrained optimization and multi-objective optimization are highly related. In particular, constrained optimization can be considered a special case of multi-objective optimization in which the user's utility function for the constraint functions is an infinite step function: constant over the feasible region and negative infinity elsewhere.

2.9 Related research areas

This section described some areas of research in other fields that are related to Bayesian optimization.

2.9.1 Derivative-free optimization

The field of Bayesian optimization often takes a machine learning perspective. In this section we describe some methods related in the area of derivative-free optimization (DFO), which lies within the field of optimization. While its name suggests that DFO methods are intended for problems in which the function derivatives are unknown (Chapter 1, property (A)), they may also apply to problems with any or all of the properties (B) to (D) depending on the particular method. DFO methods are an alternative to Bayesian optimization and can often be applied to the same problems. Loosely speaking, the focus of DFO tends to be more on theoretical soundness and less on efficiency in the case of very expensive functions. See Conn et al. (2009) for an introduction to DFO. As discussed in Conn et al. (2009), the field of DFO for constrained optimization remains relatively unexplored. Below we discuss two DFO methods which can be used for difficult black-box global optimization problems.

Evolution strategies and CMA-ES. One class of DFO methods is the evolution strategy. In evolution strategies, a set of candidate solutions is maintained, and better solutions survive through a process of artificial natural selection (no irony intended). A popular evolution strategy is the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) (Hansen and Ostermeier, 1996). In CMA-ES, at each iteration, the candidate solutions are sampled from a multivariate normal distribution whose mean and covariance are estimated incrementally as optimization proceeds. The unknown function is evaluated at each of these candidate solutions. Then, the solutions are sorted and a fixed proportion of the best solutions are kept (this is the selection step). Finally, these chosen solutions are used to update the estimates of the mean and covariance matrix and optimization proceeds. CMA-ES has been extended to handle noisy function (Hansen et al., 2009) and some evolutionary algorithms have also been extended to handle unknown constraints with Boolean oracles (see e.g., Kramer, 2010; Arnold and Hansen, 2012).

DIRECT. Another global DFO method is the method of DIviding RECTangles (DIRECT) (Jones et al., 1993). DIRECT works by iteratively dividing the search domain into hyper-rectangles and evaluating the unknown function at particular locations within the hyper-rectangles. The goal is to find a hyper-rectangle that contains the solution. Hyper-rectangles are divided further if they are deemed likely to contain the solution, or if they are large; these criteria allow the method to perform both local and global search, respectively. DIRECT has also been extended to address noisy functions (Deng and Ferris, 2007).

Local methods for global optimization Other DFO methods, such as the Nelder-Mead method (Nelder and Mead, 1965), are derivative-free but perform local optimization. A local optimization method may be turned into a global method by randomly restarting the optimization from different locations. However, when function evaluations are expensive such approaches are inefficient in general as information is not shared between these independent runs. On the other hand, if function evaluations are not expensive, the meta-method of random restarts may be appealing. One advantage of the meta-method of random restarts is that it is naturally parallelizable.

2.9.2 Optimal experimental design and active learning

The field of statistics contains the area of *optimal experimental design*, also called the *design of experiments* (DOE). A simple intuitive example is the following: imagine one is trying to fit a line in the unit interval $[0, 1]$ to data that are drawn from the true unknown line plus i.i.d. Gaussian noise. One can only query two points in the interval. Which two points should one query to best determine the line? The answer is that one should query the two endpoints of the interval.

A real-world application of DOE is in geophysics: one must decide on the placement of a limited number of sensors to best understand the properties of the earth, or, similarly, the placement of a limited number of sources whose seismic wave reflections inform us about the properties of the earth. Recent work (Krause, 2008) shows that if the objective function is submodular, a greedy approach performs near-optimally.

The field of *active learning* is highly related to DOE. In active learning, the goal is typically to learn a model or classifier as well as possible through limited sequential queries. Information-based acquisition functions, described in Section 2.4.4, are an interesting class of methods that bridge the gap between active learning / DOE and Bayesian optimization. In information-based methods, the goal is to construct the best estimator of the global minimizer $\mathbf{x}_\star$. Thus, an estimation problem is turned into an optimization problem. In fact, the information-based approach was originally used in the active learning literature, by MacKay (1992). In the field of operations research, this is also called *optimal learning* (see, e.g., Powell and Ryzhov, 2012).

2.9.3 Bandits

The term *bandit* comes from the multi-armed bandit problem, in which one is faced with a number of slot machines, each machine possessing an arm that one can pull. In the bandit setting, one typically searches for an algorithm that minimizes the *regret*, which is the difference between one's reward and the best possible reward at iteration n. Two types of regret are *cumulative regret* and *simple regret*. The cumulative regret R_N is the average of the instantaneous regrets r_n incurred from the start to iteration N, i.e., $R_N = \frac{1}{N}\sum_{t=0}^{N} r_n$. Simple regret, on the other hand, is the regret at the final iteration, namely r_N. In the bandit setting, one often seeks to devise a *no-regret* algorithm, i.e., an algorithm for which $\lim_{N\to\infty} R_N = 0$. Methods that aim to minimize cumulative regret must balance the tradeoff between exploration (sacrificing immediate reward for future gains) and exploitation (greedy behavior). For example, one must decide whether to try a new, unknown, possibly bad slot machine (exploration), or continue playing the best slot machine found so far (exploitation). The notion of simple regret better captures the goal of Bayesian optimization. In Bayesian optimization, we are not penalized for observing very poor function values during the optimization process. Rather, the goal is to find the $\mathbf{x}$ with the lowest function value when optimization terminates.

The bandit setting differs from Bayesian optimization in that set of possible actions is typically discrete, and the goal is often to minimize cumulative regret. Furthermore, the bandit literature places a greater emphasis on theory and comparing convergence rates of different algorithms. However, the two fields are very closely related, and there is a large body of work

spanning the two fields. *Contextual bandits* refer to problems in which each arm of a multi-armed bandit has an associated feature vector or *context*. Over time, one learns to predict the reward as a function of these contexts. This is very similar to the role of the GP in Bayesian optimization: if we think of the input space $\mathcal{X}$ as an continuum of infinitely many arms, each $\mathbf{x}$ is the context and the GP allows us to generalize across contexts and learn global properties of the system. Another bridge between Bayesian optimization and bandits is the GP-UCB acquisition function (Section 2.4.2). GP-UCB has strong theoretical guarantees from the bandit perspective but also works well in practice. This connection is also explored in Bubeck et al. (2009).

2.10 Limitations of Bayesian optimization

Bayesian optimization methods have several limitations and challenges. One challenge is that Bayesian optimization methods typically only work well in low-dimensional search spaces because, in general, the difficulty of the problem can grow exponentially with the dimensionality D of the search space. However, there have been some recent advances in this area. The general approach taken is to assume that the function lies on a lower-dimensional manifold; in other words, that it has an intrinsic dimensionality much smaller than the dimensionality of the search space. Wang et al. (2013) exploit this property by using random linear projections onto a lower-dimensional subspace. Djolonga et al. (2013) attempt to approximate this lower-dimensional subspace and provide sub-exponential regret rates even in the presence of noise. Kandasamy et al. (2015) assume the true function is a sum of lower-dimensional components and introduce the *Add-GP-UCB* algorithm. They provide convergence rates and show that the method works in practice even when the additivity assumption is violated.

Another challenge is that, due to the cubic scaling of Gaussian processes with respect to the data set size (see Section 2.2), Bayesian optimization can become intractably slow once more than a few hundred data are collected. Recent work on using random forests (Hutter et al., 2011) and neural networks (Snoek et al., 2015) instead of Gaussian processes to model the unknown function is a highly promising step in overcoming this limitation.

Another limitation of Bayesian optimization is that it operates under the assumption that

function evaluations are very slow. If the function evaluations turn out to be fast, too much time is wasted on the optimization steps themselves. In unconstrained optimization problems this issue can be avoided by simply using another approach in place of Bayesian optimization. However, in multi-task problems (Section 2.8) or constrained problems with decoupled constraints (Chapter 5), we may be faced with one or more fast functions (e.g., the objective) and one or more slow functions (e.g., a constraint). In this case, neither methods for fast functions (like those discussed in Section 2.9.1) nor methods for slow functions (like Bayesian optimization) are suited to the problem. In Section 4.2 we address this issue for decoupled constraints by introducing a method that can perform both fast and slow optimization steps, adaptively switching between them given the relative speeds of the optimization steps and the function evaluations.

2.11 Bayesian optimization software

Recently there has been a proliferation of publicly available software for Bayesian optimization. The following non-exhaustive list of software may be useful to the reader:

- Spearmint (Snoek et al., 2012). `https://github.com/HIPS/Spearmint/`
- SMAC (Hutter et al., 2011). URL: `http://www.cs.ubc.ca/labs/beta/Projects/SMAC/`
- Auto-WEKA (Thornton et al., 2013). `http://www.cs.ubc.ca/labs/beta/Projects/autoweka/`
- Hyperopt (Bergstra et al., 2011). URL: `http://jaberg.github.com/hyperopt/`
- MOE (Yelp, Inc.). URL: `http://yelp.github.io/MOE/`
- pybo (Hoffman and Shahriari, 2014). URL: `https://github.com/mwhoffman/pybo/`
- BayesOpt (Martinez-Cantin, 2014). URL: `http://rmcantin.bitbucket.org/html/`
- DiceOptim (Roustant et al., 2012). URL: `http://cran.r-project.org/web/packages/DiceOptim/index.html`

The methods described in this thesis are available as part of the Spearmint package.

Chapter 3

Extensions to Expected Improvement with Constraints

This chapter presents extensions to the Expected Improvement with Constraints (EIC) acquisition function described in Section 2.6.1. This work is motivated by the fact that prior descriptions of EIC are inconsistent in two ways: the solution to the problem is not well-defined when constraints are noisy, and EIC is not defined when no feasible point has been found. Furthermore, this chapter presents an extension to EIC in the presence of decoupled constraints.

3.1 Probabilistic constraints

In Bayesian optimization, the objective and constraint functions are in general unknown for two reasons. First, the functions have not been observed everywhere, and therefore we must interpolate or extrapolate their values to new inputs. Second, our observations may be noisy: even after multiple observations at $\mathbf{x}$, we do not necessarily know the true function value at $\mathbf{x}$ with certainty. Therefore, it is impossible to be certain that a constraint $c_k(\mathbf{x}) \geq 0$ is satisfied for any $\mathbf{x}$, even if we have already observed c_k many times at $\mathbf{x}$. As a result, when constraint observations are noisy, it does not make sense to formulate the problem as in Eq. (1.2), since, after any finite number of observations (assuming Gaussian noise), it is impossible to conclude that the problem has been solved. A new formulation is needed.

Optimization problems in which the objective and/or constraints contain uncertain quantities whose probability distributions are known or can be estimated are referred to as *stochastic programming* problems (see e.g., Shapiro et al., 2009). A natural formulation of these problems is to minimize the objective function *in expectation*, while satisfying the constraints *with high probability*. The condition that the constraint be satisfied with high probability is called a *probabilistic constraint* or *chance constraint* (Shapiro et al., 2009). We denote the kth probabilistic constraint as the Boolean function $\mathcal{C}_k(\mathbf{x})$ and the conjunction of all K probabilistic constraints as $\mathcal{C}(\mathbf{x})$ with no subscript. Then,

$$\begin{aligned} \mathcal{C}_k(\mathbf{x}) &\iff \Pr(c_k(\mathbf{x}) \geq 0) \geq 1 - \delta_k \\ \mathcal{C}(\mathbf{x}) &\iff \forall k,\, \mathcal{C}_k(\mathbf{x})\,, \end{aligned} \tag{3.1}$$

for some confidence thresholds $\{\delta_k\}$. In Eq. (3.1), $\mathcal{C}(\mathbf{x})$ is the Boolean function that must be satisfied at the solution to the optimization problem.

This formulation is based on K individual probabilistic constraints, one per original constraint. Another reasonable formulation requires that the joint probability of *all* constraints being satisfied at $\mathbf{x}$ must be above some single threshold δ:

$$\mathcal{C}(\mathbf{x}) \iff \Pr\left(\forall k,\; c_k(\mathbf{x}) \geq 0\right) \geq 1 - \delta\,. \tag{3.2}$$

Here we have overloaded the notation $\mathcal{C}(\mathbf{x})$ with no subscript to mean both the conjunction of K individual probabilistic constraints (Eq. (3.1)) or this second type of probabilistic constraint based on a joint probability (Eq. (3.2)). This notation is convenient because we can now use the notation $\mathcal{C}(\mathbf{x})$ to refer to the Boolean function that encapsulates all constraints, without committing to which type of probabilistic constraint is being used. Note that the two formulations are the same if $K = 1$. The two formulations in Eqs. (3.1) and (3.2) both address the issue of constraint uncertainty and yield a coherent problem formulation: while the model cannot definitively report that the original constraints $c_k(\mathbf{x}) \geq 0$ are satisfied at $\mathbf{x}$, it can definitively report whether or not it believes that the probabilistic constraint(s) are satisfied.

Given these definitions, we can formulate constrained Bayesian optimization problems as,

$$\min_{\mathbf{x}\in\mathcal{X}} \mathbb{E}[f(\mathbf{x})] \text{ s.t. } \mathcal{C}(\mathbf{x})\,. \tag{3.3}$$

For the remainder of this thesis, we will consider Eq. (3.3) rather than Eq. (1.2) as the problem formulation that we wish to solve, regardless of whether or not the constraints are noisy. In the noise-free case, setting $\delta_k = 0$ for all k (or $\delta = 0$) corresponds to demanding that we be completely certain of the constraint satisfaction, which is typically only possible at the observation locations; thus, Eq. (3.3) gracefully reduces to Eq. (1.2). The practical effect of using Eq. (3.3) lies in making recommendations, which is line 8 of Algorithm 1. With probabilistic constraints, the recommendation step is precisely Eq. (3.3):

$$\mathbf{x}_\star = \arg\min_{\mathbf{x}\in\mathcal{X}} \mathbb{E}[f(\mathbf{x})] \text{ s.t. } \mathcal{C}(\mathbf{x})\,. \tag{3.4}$$

When using the EIC acquisition function, we also use Eq. (3.3) as the incumbent η.

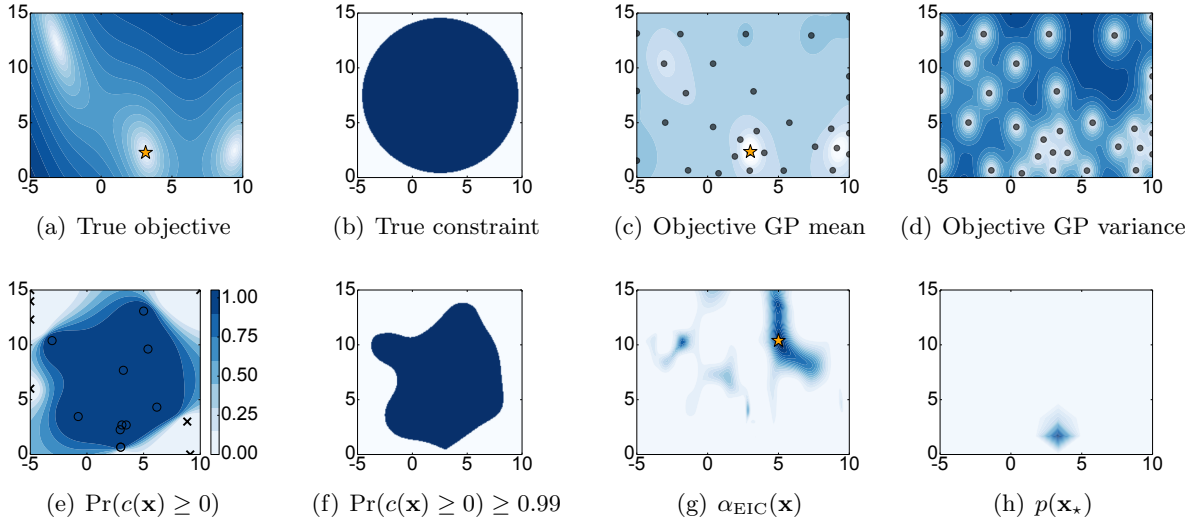
Fig. 3.1 illustrates probabilistic constraints on the Branin-Hoo function, a 2D function with three global minima (Fig. 3.1(a)). We add to this problem the disk constraint

$$c(\mathbf{x}) = 50 - (\mathbf{x}_1 - 2.5)^2 + (\mathbf{x}_2 - 7.5)^2\,,$$

shown in Fig. 3.1(b). This constraint eliminates the upper-left and lower-right solutions, leaving a unique global minimum at $\mathbf{x}_\star = (\pi, 2.275)$ indicated by the orange star in Fig. 3.1(a). Fig. 3.1(e) shows the probability of constraint satisfaction given the 17 observations of the constraint function. The probabilistic constraint $\mathcal{C}(\mathbf{x})$, which is simply this probability surface thresholded at $1 - \delta = 0.99$, is shown in Fig. 3.1(f).

3.2 Finding the feasible region

The EIC acquisition function presented in prior work (Section 2.6.1, Fig. 3.1(g)) relies on computing improvement over the target or incumbent η. However, if no feasible solution has been found (i.e., $\neg\exists\mathbf{x}, \mathcal{C}(\mathbf{x})$), then η does not exist and EIC is not defined. In this case we propose



(a) True objective (b) True constraint (c) Objective GP mean (d) Objective GP variance

(e) $\Pr(c(\mathbf{x}) \geq 0)$ (f) $\Pr(c(\mathbf{x}) \geq 0) \geq 0.99$ (g) $\alpha_{\text{EIC}}(\mathbf{x})$ (h) $p(\mathbf{x}_\star)$

Figure 3.1: Constrained Bayesian optimization on the 2D Branin-Hoo function with a disk constraint, after 50 iterations (33 objective evaluations and 17 constraint evaluations). (a) Branin-Hoo function, (b) true constraint, (c) mean of objective function GP, (d) variance of objective function GP, (e) probability of constraint satisfaction, (f) probabilistic constraint with $\delta = 0.01$, (g) EIC acquisition function, and (h) probability distribution over the location of the minimum, $p(\mathbf{x}_\star)$. Lighter colors indicate lower values. Colorbar in (e) applies to (b), (e), (f), (h); units in (a), (c), (d), (g) are arbitrary. Objective function observations are indicated with black circles in (c) and (d). Constraint observations are indicated with black $\times$'s (violations) and o's (satisfactions) in (e). Orange stars: (a) unique true minimum of the constrained problem, (c) best solution found by Bayesian optimization, (g) input chosen for the next evaluation.

that the acquisition function include only the constraint factors, so that the whole acquisition function is defined as

$$\alpha_{\text{EIC}}(\mathbf{x}) = \begin{cases} \prod_{k=1}^{K} \Pr(c_k(\mathbf{x}) \geq 0) \alpha_{\text{EI}}(\mathbf{x}), & \text{if } \exists \mathbf{x},\, \mathcal{C}(\mathbf{x}) \\ \prod_{k=1}^{K} \Pr(c_k(\mathbf{x}) \geq 0), & \text{otherwise} \end{cases} \tag{3.5}$$

Intuitively, if the probabilistic constraint is violated everywhere, this acquisition function ignores the objective function f and focuses only on searching for a feasible region. This feasibility search is purely exploitative: it searches where the probability of satisfying the constraints is highest. This is possible because the true probability of constraint satisfaction is either zero or one. Therefore, as the algorithm continues to probe a particular region, it will either discover that the region is feasible, or the probability of feasibility will eventually drop and it will move on to a more promising region.

3.3 EIC with decoupled observations

In some problems, the objective and constraint functions may be evaluated independently. We call this property the *decoupling* of the objective and constraint functions (see Section 2.8 and Chapter 5). In decoupled problems, we must choose to evaluate a subset of the tasks (for example, either the objective function or one of the constraint functions) at each iteration of Bayesian optimization. It is important to identify problems with this decoupled structure, because often some of the functions are much more expensive to evaluate than others. In there situations, it is much more efficient to evaluate the cheap function first and then evaluate the expensive function only in those regions that are promising.

One possible acquisition function for decoupled constraints is the expected improvement of individually evaluating each task. However, the myopic nature of the EI criterion causes a pathology in this formulation that prevents exploration of the design space. Consider a situation, with a single constraint, in which some feasible region has been identified and thus the incumbent is defined, but a large unexplored region remains. Evaluating only the objective in this region could not cause improvement as our belief about $\Pr(c(\mathbf{x}) \geq 0)$ will follow the prior and not exceed the threshold $1 - \delta$. Likewise, evaluating only the constraint would not cause improvement because our belief about the objective will follow the prior and is unlikely to become the new best. This is a chicken-and-egg pathology: we must learn that *both* the objective and the constraint are favorable for improvement to occur, but this is not possible when only a single task is observed. This difficulty suggests a non-myopic acquisition function which assesses the improvement after a sequence of objective and constraint observations. However, such a multi-step acquisition function is intractable in general (Ginsbourger and Riche, 2010a).

Instead, to address this pathology, we propose to use the coupled acquisition function (Eq. (3.5)) to select an input $\mathbf{x}$ for evaluation, followed by a second step to determine which task will be evaluated at $\mathbf{x}$. Following Swersky et al. (2013), we select a task with the SUR criterion (Villemonteix et al., 2009, Section 2.4.5 and Eq. (2.10)), but extended to constrained

optimization. Our decision criterion is then

$$\hat{t} = \arg\max_{t \in \mathcal{T}} -\mathbb{E}_{y \mid \mathcal{D}, \mathbf{x}, t} \left[\mathrm{H} \left[\mathbf{x}_\star \mid \mathcal{D} \cup \{(\mathbf{x}, y)\} \right] \right] , \tag{3.6}$$

where t is one of the functions in $\mathcal{T} = \{f, c_1, \ldots, c_K\}$, $\hat{t}$ is the suggested task, $\mathrm{H}(\cdot)$ is the differential entropy functional, y is the unknown function value obtained by observing task t at $\mathbf{x}$, and we have omitted the current entropy term since it does not affect $\hat{t}$. This approach avoids the pathology described above because the choice of $\mathbf{x}$ takes into account both the objective and the constraints and then, given $\mathbf{x}$, the most informative task is selected. We call this two-stage method EIC for decoupled constraints or EIC-D. This method can be implemented using IAGO (Section 2.4.5 and Appendix B.11) where we now generate Monte Carlo samples of the constraint functions in addition to the objective. A general treatment of decoupling is given is Chapter 5.

3.4 Discussion

3.4.1 Interpretation of EIC when using probabilistic constraints

When combined with probabilistic constraints, EIC loses its intuitive definition as the expected improvement of the best feasible solution. This is clear because the EIC does not depend on δ, whereas the incumbent η does. One may then ask why the true EI is not used. The answer lies in the following pathology: a single noisy observation of constraint satisfaction may be insufficient to push the overall posterior probability of satisfaction above $1 - \delta$. Consider, for example, the case of noisy Boolean constraint observations. Observing the constraint at $\mathbf{x}$, the best value one could observe is a Boolean "1" indicating constraint satisfaction. This observation increases our posterior probability that the constraint is satisfied at $\mathbf{x}$, but because these Boolean values are noisy, the probability will not be increased all the way to 1. If the probability is not increased above $1 - \delta$, then $\mathbf{x}$ is not considered feasible according to the probabilistic constraint even after the best possible observation. Thus, no matter the outcome, observing the constraint at $\mathbf{x}$ cannot cause $\mathbf{x}$ to become the new incumbent, and therefore the expected improvement is zero.

This pathology is rooted in the fact that EI is a myopic strategy; indeed, assuming $\mathbf{x}$ really is feasible, then after many evaluations of this noisy Boolean constraint at $\mathbf{x}$ the probabilistic constraint will eventually be satisfied.

3.4.2 Potential inefficiency of EIC

EIC as defined in Eq. (3.5) can become very inefficient when searching for the feasible region. Consider the case of a single constraint with continuous-valued observations and very high noise. Imagine that the initial design points include several points at which the constraint is clearly violated, and one point, $\mathbf{x}$, at which the true constraint value is just over zero, say 0.001. By Eq. (3.5), EIC will continue evaluating the constraint at $\mathbf{x}$ because its probability of satisfaction is slightly higher than the prior value of 0.5. As we continue to repeatedly evaluate the constraint at $\mathbf{x}$, we will eventually satisfy the probabilistic constraint, but this will require many evaluations because the signal-to-noise ratio is very low. These evaluations are wasteful if the true solution is not nearby; and, indeed, there is no particular reason why it should be nearby. This pathology is due to the fact that Eq. (3.5) does not consider the sensitivity of the constraint to a new observation, but rather only its current probability of satisfying the constraints.

3.4.3 Knowledge use in EIC-D

The EIC-D scheme for decoupled constraints avoids the chicken-and-egg pathology but is unsatisfying because of its two-stage nature. By choosing $\mathbf{x}$ without considering the task, we make a sub-optimal choice of $\mathbf{x}$. In the decoupled scenario, our decision space is the product space $\mathcal{T} \otimes \mathcal{X}$. Thus, a more satisfying method would jointly maximize its acquisition function over $\mathcal{T} \otimes \mathcal{X}$, since maximizing first over $\mathbf{x}$ and then over t does not in general yield the joint maximizer. In a sense, then, EIC-D does not make full use of all the information or knowledge available to it. We will address this issue in Chapters 4 and 5.

Chapter 4

Predictive Entropy Search with Constraints

This chapter extends predictive entropy search (PES) (Section 2.4.7, Hernández-Lobato et al., 2014) to solve constrained problems, an approach that I call *predictive entropy search with constraints* (PESC). The extension of information-based methods to constrained problems is natural because the influence of the constraints is entirely accounted for in $p(\mathbf{x}_\star)$; the SUR acquisition function in Eq. (2.10) remains unchanged.

PESC does not suffer from the difficulties underlying EIC discussed in Chapter 3. First, the PESC acquisition function does not depend on the current best feasible solution, so it can operate coherently even when there is not yet a feasible solution. Second, PESC naturally separates the contributions of each task (objective or constraint) in its acquisition function. As a result, no pathology arises in the decoupled case and, thus, no *ad hoc* modifications to the acquisition function are required. This allows us to apply PESC to problems such as those discussed in Chapter 1, namely minimizing the number of calories in a cookie recipe such that it still tastes good, or optimizing the performance of a speech recognition system such that it makes predictions within a prescribed time limit. Furthermore, in addition to its increased generality, our experiments show that PESC performs better than EIC even in the basic setting of joint evaluations to which EIC is most suited (Chapter 6).

4.1 Deriving the PESC acquisition function

Here, as in PES, we commit to zero-mean GP priors on the unknown functions, with i.i.d. Gaussian zero-mean observation noise. PESC is an approximation to the same key quantity as in IAGO, ES and PES, namely Eq. (2.10). We now extend this equation to the setting in which we are optimizing an objective function f subject to K inequality constraints, such that instead of observing one output y we now observe $K+1$ outputs $y^f, y^1, \ldots, y^K$:

$$\alpha_{\mathrm{PESC}}(\mathbf{x}) = \mathrm{H}\left[\mathbf{x}_\star \,|\, \mathcal{D}^f, \mathcal{D}^1, \ldots, \mathcal{D}^K\right] - \mathbb{E}_{p(y^f, y^1, \ldots, y^K \,|\, \mathcal{D}^f, \mathcal{D}^1, \ldots, \mathcal{D}^K, \mathbf{x})}\Big[\mathrm{H}\left[\mathbf{x}_\star \,|\, \mathcal{D}^f \cup \{(\mathbf{x}, y^f)\}, \mathcal{D}^1 \cup \{(\mathbf{x}, y^1)\}, \ldots, \mathcal{D}^K \cup \{(\mathbf{x}, y^K)\}\right] \Big], \quad (4.1)$$

where $\mathcal{D}^f$ is the data set for the objective function, $\mathcal{D}^k$ is the data set for constraint k. As in PES, we then flip this equation using Eq. (2.11), and arrive at a constrained version of Eq. (2.13):

$$\alpha_{\mathrm{PESC}}(\mathbf{x}) = \mathrm{H}\left[y^f, y^1, \ldots, y^K \,|\, \mathcal{D}^f, \mathcal{D}^1, \ldots, \mathcal{D}^K, \mathbf{x}\right] - \mathbb{E}_{p(\mathbf{x}_\star \,|\, \mathcal{D}^f, \mathcal{D}^1, \ldots, \mathcal{D}^K)}\Big[\mathrm{H}\left[y^f, y^1, \ldots, y^K \,|\, \mathcal{D}^f, \mathcal{D}^1, \ldots, \mathcal{D}^K, \mathbf{x}, \mathbf{x}_\star\right]\Big] \quad (4.2)$$

As in PES, the rationale for flipping Eq. (4.1) to Eq. (4.2) is that the predictive entropies in Eq. (4.2) are easier to approximate than the entropies of $\mathbf{x}_\star$ in Eq. (4.1).

As mentioned in Section 2.4.7, the first term on the right-hand side of Eq. (4.2) is straightforward to compute because $p(y^f, y^1, \ldots, y^K \,|\, \mathcal{D}^f, \mathcal{D}^1, \ldots, \mathcal{D}^K, \mathbf{x})$ is a product of independent Gaussians. By Eq. (2.14), the entropy is given by

$$\mathrm{H}\left[y^f, y^1, \ldots, y^K \,|\, \mathcal{D}^f, \mathcal{D}^1, \ldots, \mathcal{D}^K, \mathbf{x}\right] = \frac{1}{2}\log(\sigma_f^2(\mathbf{x}) + \nu_f^2) + \frac{1}{2}\left(\sum_{k=1}^{K}\log(\sigma_k^2(\mathbf{x}) + \nu_k^2)\right) + \frac{K+1}{2}\log(2\pi e), \quad (4.3)$$

where σ_f^2 is the posterior predictive marginal variance of the objective at $\mathbf{x}$, ν_f^2 is the noise variance for the objective, σ_k^2 is the posterior predictive marginal variance of constraint k at $\mathbf{x}$, and ν_k^2 is the noise variance for constraint k. However, the second term on the right-hand side

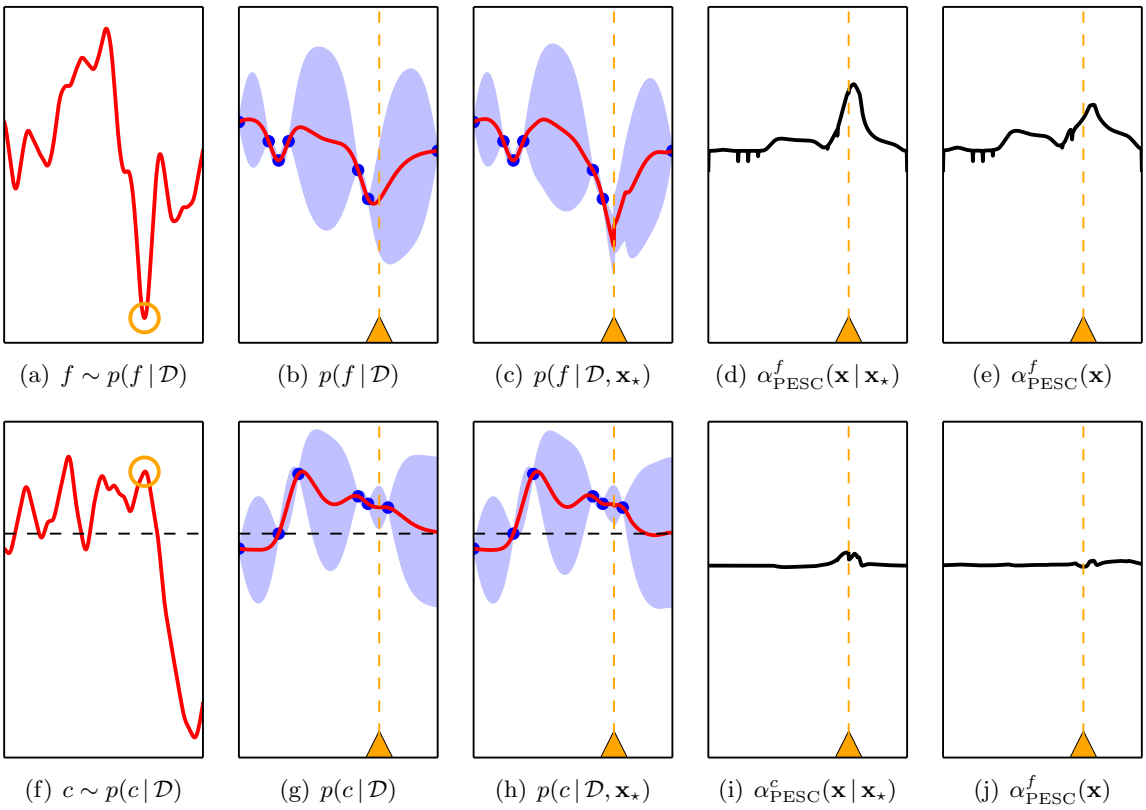
of Eq. (4.2), which we call the expected *conditioned predictive entropy*, has to be approximated. This approximation is described below. For brevity, in the following we use $\mathcal{D}$ as shorthand for $\mathcal{D}^f, \mathcal{D}^1, \ldots, \mathcal{D}^K$.

4.1.1 Strategy for approximating the conditioned predictive entropy

The expected conditioned predictive entropy in Eq. (4.2) can be broken down into two parts: the expectation over $\mathbf{x}_\star$ and the predictive entropy conditioned on $\mathbf{x}_\star$. As in PES, we approximate the expectation in Eq. (4.2) by averaging over samples of $\mathbf{x}_\star$ drawn approximately from $p(\mathbf{x}_\star \mid \mathcal{D})$. To sample $\mathbf{x}_\star$, we first approximately draw f and $c_1, \ldots, c_K$ from their GP posteriors using a finite basis function approximation to the GP kernel (Rahimi and Recht, 2007). This yields a representation of the sample that can be evaluated at any $\mathbf{x}$ with speed that is independent of the data set size and linear in the number of basis features used. We then solve a global constrained optimization problem using the sampled functions. The solution to this problem is the sample of $\mathbf{x}_\star$. This is analogous the approach taken in Hernández-Lobato et al. (2014), but extended to the constrained setting. See Hernández-Lobato et al. (2014, Appendix A) for the derivation of this approximation, and Appendix B.9 for implementation details. This sampling procedure is illustrated in 1-D in Figs. 4.1(a) and 4.1(f).

For each sample of the global minimum, we must compute the entropy of the *conditioned predictive distribution* (CPD), $p(y^f, y^1, \ldots, y^K \mid \mathcal{D}, \mathbf{x}, \mathbf{x}_\star)$. Figures 4.1(b) and 4.1(g) show the predictive distributions for 1-D objective and constraint functions, and Figs. 4.1(c) and 4.1(h) show these same predictive distributions conditioned on a particular value of the global minimizer $\mathbf{x}_\star$. Conditioning on the location of the global minimizer $\mathbf{x}_\star$ pushes the posterior predictions for f down at locations around $\mathbf{x}_\star$ and up in regions away from $\mathbf{x}_\star$. This only occurs in regions where the constraints $c_1, \ldots, c_K$ are likely to be positive; conditioning on $\mathbf{x}_\star$ has little or no effect on f in regions where $c_1, \ldots, c_K$ are presumed to be negative. Furthermore, conditioning on $\mathbf{x}_\star$ pushes the posterior distribution of $c_1, \ldots, c_K$ up near $\mathbf{x}_\star$ because the constraints need to be satisfied at the solution.

Our approach to computing the conditioned predictive entropy is to create a Gaussian approximation to the CPD, and then apply Eq. (2.14) to compute the entropy. Figures 4.1(d)



(a) $f \sim p(f \mid \mathcal{D})$ (b) $p(f \mid \mathcal{D})$ (c) $p(f \mid \mathcal{D}, \mathbf{x}_\star)$ (d) $\alpha^f_{\text{PESC}}(\mathbf{x} \mid \mathbf{x}_\star)$ (e) $\alpha^f_{\text{PESC}}(\mathbf{x})$

(f) $c \sim p(c \mid \mathcal{D})$ (g) $p(c \mid \mathcal{D})$ (h) $p(c \mid \mathcal{D}, \mathbf{x}_\star)$ (i) $\alpha^c_{\text{PESC}}(\mathbf{x} \mid \mathbf{x}_\star)$ (j) $\alpha^f_{\text{PESC}}(\mathbf{x})$

Figure 4.1: Visualization of PESC in a problem with one constraint. The top row represents the objective function f and the bottom row the constraint function c. 1st column: a single sample from the finite basis function approximation of the GP, with the constrained minimum $\mathbf{x}_\star$ marked with an orange circle. 2nd column: the posterior predictive distribution with the mean shown in red, the data as blue circles, the standard deviation with shaded blue, and the sampled $\mathbf{x}_\star$ in orange. 3rd column: the approximated CPD. 4th column: the partial acquisition functions for the objective and the constraint, given the single sample of $\mathbf{x}_\star$. 5th column: the partial acquisition functions for the objective and the constraint, averaged over 100 samples of $\mathbf{x}_\star$. In all columns, the sampled $\mathbf{x}_\star$ is shown in orange.

and 4.1(i) show the PESC acquisition function computed in this manner for one particular sample of $\mathbf{x}_\star$, while Figs. 4.1(e) and 4.1(j) show the PESC acquisition function averaged over 100 samples of $\mathbf{x}_\star$.

4.1.2 Constructing the NFCPD

Let $\mathbf{y} = [y^f, y^1, \ldots, y^K]^\top$ denote the vector of random variables associated with objective and constraint observation values at $\mathbf{x}$ and let $\mathbf{z} = [f(\mathbf{x}), c_1(\mathbf{x}), \ldots, c_K(\mathbf{x})]^\top$ denote the vector of the latent objective and constraint functions at $\mathbf{x}$, such that the entries of $\mathbf{y}$ correspond to the

entries of $\mathbf{z}$ corrupted with observation noise. Then, we can write the CPD as $p(\mathbf{y} \mid \mathcal{D}, \mathbf{x}, \mathbf{x}_\star)$. We approximate the CPD by first approximating the posterior predictive distribution $p(\mathbf{z} \mid \mathcal{D}, \mathbf{x}, \mathbf{x}_\star)$, which we call the *noise-free conditioned predictive distribution* (NFCPD), and then convolving that approximation with additive Gaussian noise of variance $\nu_0^2, \ldots, \nu_K^2$ to form an approximation of the CPD.

We first consider the distribution $p(\mathbf{x}_\star \mid f, c_1, \ldots, c_K)$, which is the distribution of $\mathbf{x}_\star$ given the infinite-dimensional latent objective and constraint functions $f, c_1, \ldots, c_K$. The global minimizer is in fact a deterministic function of $f, c_1, \ldots, c_K$. In particular, $\mathbf{x}_\star$ is the global minimizer if and only if the following two conditions are met:

(i) all constraints are satisfied at $\mathbf{x}_\star$: $\forall k,\ c_k(\mathbf{x}_\star) \geq 0$

(ii) $f(\mathbf{x}_\star)$ is the smallest feasible value: $\forall \mathbf{x}' \in \mathcal{X},$ either $f(\mathbf{x}') \geq f(\mathbf{x}_\star)$ or $\exists k,\ c_k(\mathbf{x}') < 0$

We can informally translate these deterministic conditions into a conditional probability:

$$p(\mathbf{x}_\star \mid f, c_1, \ldots, c_K, \mathcal{D}) = \left[\prod_{k=1}^{K} \Theta\left[c_k(\mathbf{x}_\star)\right]\right] \prod_{\mathbf{x}' \in \mathcal{X}} \Psi(\mathbf{x}', \mathbf{x}_\star, f, c_1, \ldots, c_K)\,, \tag{4.4}$$

where

$$\Psi(\mathbf{x}', \mathbf{x}_\star, f, c_1, \ldots, c_K) = \left\{\prod_{k=1}^{K} \Theta\left[c_k(\mathbf{x}')\right]\right\} \Theta[f(\mathbf{x}') - f(\mathbf{x}_\star)] + \left\{1 - \prod_{k=1}^{K} \Theta\left[c_k(\mathbf{x}')\right]\right\}, \tag{4.5}$$

the symbol Θ denotes the Heaviside step function with the convention that $\Theta(0) = 1$, and we use the fact that $\mathbf{x}_\star$ is conditionally independent of $\mathcal{D}$ given $f, c_1, \ldots, c_K$ to unnecessarily condition on $\mathcal{D}$. The first product in Eq. (4.4) encodes property (i) and the infinite product over $\Psi(\mathbf{x}', \mathbf{x}_\star, f, c_1, \ldots, c_K)$ encodes property (ii). Here, we are being slightly informal in the use of infinite products and infinite dimensional vectors; see Hennig and Schuler (2012) for further discussion of this issue.

Next, we use Bayes' theorem to flip this conditional distribution:

$$p(f, c_1, \ldots, c_K \,|\, \mathbf{x}_\star, \mathcal{D}) \propto p(f, c_1, \ldots, c_K \,|\, \mathcal{D}) \times \left[\prod_{k=1}^{K} \Theta\left[c_k(\mathbf{x}_\star)\right]\right] \left[\prod_{\mathbf{x}'} \Psi(\mathbf{x}', \mathbf{x}_\star, f, c_1, \ldots, c_K)\right] . \tag{4.6}$$

The infinite-dimensional vectors $f, c_1, \ldots, c_K$ are drawn from $p(f, c_1, \ldots, c_K \,|\, \mathcal{D})$, given as infinite-dimensional multivariate Gaussian distributions (i.e., the GPs).

We now return to $\mathbf{z}$ in order to construct the NFCPD. Because $\mathbf{z}$ is simply a vector containing the values of $f, c_1, \ldots, c_K$ at $\mathbf{x}$, $\mathbf{z}$ is also a deterministic function of $f, c_1, \ldots, c_K$ and we can write $p(\mathbf{z} \,|\, f, c_1, \ldots, c_K, \mathbf{x})$ using Dirac delta functions (denoted δ, not to be confused with the probabilistic constraint confidence threshold, also denoted δ in other contexts) to pick out the values at $\mathbf{x}$:

$$p(\mathbf{z} \,|\, f, c_1, \ldots, c_K, \mathbf{x}, \mathbf{x}_\star, \mathcal{D}) = \delta[z_0 - f(\mathbf{x})] \left[\prod_{k=1}^{K} \delta[z_k - c_k(\mathbf{x})]\right] . \tag{4.7}$$

In Eq. (4.7) we have used the fact that $\mathbf{z}$ is conditionally independent of $\mathbf{x}_\star$ and $\mathcal{D}$ given $f, c_1, \ldots, c_K$ to unnecessarily condition on $\mathbf{x}_\star$ and $\mathcal{D}$.

We can now construct the NFCPD $p(\mathbf{z} \,|\, \mathcal{D}, \mathbf{x}, \mathbf{x}_\star)$ by using the identity for conditional probabilities,

$$p(\mathbf{z} \,|\, \mathbf{x}_\star, \mathbf{x}, \mathcal{D}) = \int p(\mathbf{z} \,|\, f, c_1, \ldots, c_K, \mathbf{x}_\star, \mathbf{x}, \mathcal{D})\, p(f, c_1, \ldots, c_K \,|\, \mathbf{x}_\star, \mathbf{x}, \mathcal{D})\; df\, dc_1 \cdots dc_K \,, \tag{4.8}$$

to combine Eqs. (4.6) and (4.7) and marginalize out the infinite-dimensional quantities $f, c_1, \ldots, c_K$ which encode the objective and the constraint functions:

$$p(\mathbf{z} \,|\, \mathcal{D}, \mathbf{x}, \mathbf{x}_\star) \propto \int p(f, c_1, \ldots, c_K \,|\, \mathcal{D})\, \delta[z_0 - f(\mathbf{x})] \left[\prod_{k=1}^{K} \delta[z_k - c_k(\mathbf{x})]\right] \times \left[\prod_{k=1}^{K} \Theta[c_k(\mathbf{x}_\star)]\right] \left[\prod_{\mathbf{x}'} \Psi(\mathbf{x}', \mathbf{x}_\star, f, c_1, \ldots, c_K)\right] df\, dc_1 \cdots dc_K \,. \tag{4.9}$$

We now find a Gaussian approximation to Eq. (4.9) in several steps. The overall approach is

to split Eq. (4.9) into the parts that depend on $\mathbf{x}$ and the parts that do not depend on $\mathbf{x}$. The parts of the computation that depend on $\mathbf{x}$ must be performed repeatedly for each new $\mathbf{x}$ as we maximize the acquisition function. On the other hand, the parts of the approximation that do not depend on $\mathbf{x}$ can be done once, cached, and reused as we maximize the acquisition function. As we will see, most of the factors do not depend on $\mathbf{x}$, leading to crucial speedups over a naive approach that approximates the full Eq. (4.9) for each $\mathbf{x}$.

Because the infinite product over all $\mathbf{x}' \in \mathcal{X}$ in Eq. (4.9) includes $\mathbf{x}$ itself, we first split this product into these two cases, and reorder the factors:

$$p(\mathbf{z} \,|\, \mathcal{D}, \mathbf{x}, \mathbf{x}_\star) \propto \int p(f, c_1, \ldots, c_K \,|\, \mathcal{D}) \left[\prod_{k=1}^{K} \Theta[c_k(\mathbf{x}_\star)]\right] \left[\prod_{\mathbf{x}' \neq \mathbf{x}} \Psi(\mathbf{x}', \mathbf{x}_\star, f, c_1, \ldots, c_K)\right] \times$$
$$\delta[z_0 - f(\mathbf{x})] \left[\prod_{k=1}^{K} \delta[z_k - c_k(\mathbf{x})]\right] \Psi(\mathbf{x}, \mathbf{x}_\star, f, c_1, \ldots, c_K)\, df\, dc_1 \cdots dc_K \,. \quad (4.10)$$

In Eq. (4.10), the factors on the first line do not depend on $\mathbf{x}$, whereas the factors on the second line do depend on $\mathbf{x}$. We now proceed with the approximation, addressing the first and second lines of Eq. (4.10) separately.

4.1.3 Finite-product approximation to the NFCPD

Following Hernández-Lobato et al. (2014), we approximate the infinite product in Eq. (4.10) with one that contains only finitely many factors, in particular only those corresponding to locations where the objective function has been observed and $\mathbf{x}_\star$. We first define the $(N+1)-$dimensional vectors $\mathbf{f}, \mathbf{c}_1, \ldots, \mathbf{c}_K$ as the the concatenation of random variables representing the latent functions at all observed locations, plus $\mathbf{x}_\star$. In symbols,

$$\mathbf{f} = [f(\mathbf{x}_1), \ldots, f(\mathbf{x}_N), f(\mathbf{x}_\star)]$$
$$\mathbf{c}_k = [c_k(\mathbf{x}_1), \ldots, c_k(\mathbf{x}_N), c_k(\mathbf{x}_\star)] \text{ for } k = 1, \ldots, K \,. \quad (4.11)$$

Our strategy will be to change the integration variables in Eq. (4.10) from the infinite-dimensional vectors $f, c_1, \ldots, c_K$ to these finite-dimensional $\mathbf{f}, \mathbf{c}_1, \ldots, \mathbf{c}_K$ and make the associated approxi-

mations to the integrand.

Our approximation to the first line of Eq. (4.10), which we denote $q(\mathbf{f}, \mathbf{c}_1, \ldots, \mathbf{c}_K)$, is given by

$$q(\mathbf{f}, \mathbf{c}_1, \ldots, \mathbf{c}_K) \propto p(\mathbf{f}, \mathbf{c}_1, \ldots, \mathbf{c}_K \,|\, \mathcal{D}) \left[\prod_{k=1}^{K} \Theta[c_{k,N+1}]\right] \left[\prod_{n=1}^{N} \Psi(\mathbf{x}_n, \mathbf{x}_\star, \mathbf{f}, \mathbf{c}_1, \ldots, \mathbf{c}_K)\right], \quad (4.12)$$

where $p(\mathbf{f}, \mathbf{c}_1, \ldots, \mathbf{c}_K \,|\, \mathcal{D})$ is the GP predictive distribution for objective and constraint values, which factors into $p(\mathbf{f} \,|\, \mathcal{D}^f) \prod_{k=1}^{K} p(\mathbf{c}_k \,|\, \mathcal{D}^k)$ by the assumed independence of the objective and constraints, and the other factors in Eq. (4.12) are simply the same as in Eq. (4.10) but only enforced at the finitely many points given by the collected data and $\mathbf{x}_\star$. Note that we have written $\Psi(\mathbf{x}', \mathbf{x}_\star, f, c_1, \ldots, c_K)$ as $\Psi(\mathbf{x}_n, \mathbf{x}_\star, \mathbf{f}, \mathbf{c}_1, \ldots, \mathbf{c}_K)$ to emphasize that the condition Ψ is only enforced at the $\mathbf{x}_n$, and thus we do not rely on values of $f, c_1, \ldots, c_K$ outside of those in $\mathbf{f}, \mathbf{c}_1, \ldots, \mathbf{c}_K$.

We then approximate the second line (not including the differentials) of Eq. (4.10) by

$$p(\mathbf{z} \,|\, \mathbf{f}, \mathbf{c}_1, \ldots, \mathbf{c}_K, \mathbf{x}) \Psi(\mathbf{z}, f_{N+1}) \,. \quad (4.13)$$

While $\mathbf{z}$ is a deterministic function of $f, c_1, \ldots, c_K$ (in fact it is just their values at $\mathbf{x}$), it is not a deterministic function of $\mathbf{f}, \mathbf{c}_1, \ldots, \mathbf{c}_K$. Therefore, in Eq. (4.13), we have approximated the deterministic relationship given by the Dirac deltas with the $(K+1)$-dimensional Gaussian conditional distribution $p(\mathbf{z} \,|\, \mathbf{f}, \mathbf{c}_1, \ldots, \mathbf{c}_K, \mathbf{x})$ given by the $K+1$ GPs. In Eq. (4.13) we have also rewritten $\Psi(\mathbf{x}, \mathbf{x}_\star, f, c_1, \ldots, c_K)$ as $\Psi(\mathbf{z}, f_{N+1})$ to emphasize that it only depends on $\mathbf{z}$ and f_{N+1}. This can be observed by recalling the definition of Ψ (Eq. (4.5)) and observing that $\Psi(\mathbf{x}, \mathbf{x}_\star, f, c_1, \ldots, c_K)$ only depends on $f, c_1, \ldots, c_K$ through their values at $\mathbf{x}$ (this is $\mathbf{z}$) plus $f(\mathbf{x}_\star) = f_{N+1}$.

Combining our approximations to the first and second lines of Eq. (4.10), given by Eqs. (4.12) and (4.13) respectively, we obtain the following approximation to the NFCPD:

$$p(\mathbf{z} \,|\, \mathcal{D}, \mathbf{x}, \mathbf{x}_\star) \approx \frac{1}{Z_1} \int q(\mathbf{f}, \mathbf{c}_1, \ldots, \mathbf{c}_K) p(\mathbf{z} \,|\, \mathbf{f}, \mathbf{c}_1, \ldots, \mathbf{c}_K, \mathbf{x}) \Psi(\mathbf{z}, f_{N+1}) \, d\mathbf{f} \, d\mathbf{c}_1 \cdots d\mathbf{c}_K \,, \quad (4.14)$$

where Z_1 is a normalization constant. We call this the finite-dimensional approximation to the NFCPD. In the next section we proceed with a Gaussian approximation to Eq. (4.14).

4.1.4 Gaussian approximation to the finite-dimensional NFCPD

Because the integrand in Eq. (4.14) is not tractable, we form a Gaussian approximation using EP. Again, we separately address the parts that do and do not depend on $\mathbf{x}$. We begin with $q(\mathbf{f}, \mathbf{c}_1, \ldots, \mathbf{c}_K)$, the part of the NFCPD that does not depend on $\mathbf{x}$. We use EP to find a Gaussian approximation to $q(\mathbf{f}, \mathbf{c}_1, \ldots, \mathbf{c}_K)$, which we denote $\tilde{q}(\mathbf{f}, \mathbf{c}_1, \ldots, \mathbf{c}_K)$, of the form

$$\tilde{q}(\mathbf{f}, \mathbf{c}_1, \ldots, \mathbf{c}_K) = \mathcal{N}(\mathbf{f} \,|\, \mathbf{m}^{\mathbf{f}}, \mathbf{V}^{\mathbf{f}}) \prod_{k=1}^{K} \mathcal{N}(\mathbf{c}_k \,|\, \mathbf{m}^{\mathbf{c}_k}, \mathbf{V}^{\mathbf{c}_k}) \,, \tag{4.15}$$

where $(\mathbf{m}_k, \mathbf{V}_k)$ for $k = 0, \ldots, K$ are the mean and covariance terms determined by EP. The EP update rules for determining these parameters are given in Appendix A.3. This iterative EP procedure is performed only once per iteration of Bayesian optimization because it does not depend on $\mathbf{x}$.

Substituting Eq. (4.15) into Eq. (4.14), our approximation to the NFCPD is now

$$\begin{aligned} p(\mathbf{z} \,|\, \mathcal{D}, \mathbf{x}, \mathbf{x}_\star) \approx \frac{1}{Z_0} \int \mathcal{N}(\mathbf{f} \,|\, \mathbf{m}_0, \mathbf{V}^{\mathbf{f}}) \left[\prod_{k=1}^{K} \mathcal{N}(\mathbf{c}_k \,|\, \mathbf{m}^{\mathbf{c}_k}, \mathbf{V}^{\mathbf{c}_k}) \right] \times \\ p(z_0 \,|\, \mathbf{f}, \mathbf{x}) \left[\prod_{k=1}^{K} p(z_k \,|\, \mathbf{c}_k, \mathbf{x}) \right] \Psi(\mathbf{z}, f_{N+1}) \, d\mathbf{f} \, d\mathbf{c}_1 \cdots d\mathbf{c}_K \,, \end{aligned} \tag{4.16}$$

where Z_0 is a normalization constant and we have split up $p(\mathbf{z} \,|\, \mathbf{f}, \mathbf{c}_1, \ldots, \mathbf{c}_K, \mathbf{x})$ into the product of independent Gaussians by the assumed independence of the objective and constraints.

The integral in Eq. (4.16) can be split up into a product of integrals given its special structure:

$$\begin{aligned} p(\mathbf{z} \,|\, \mathcal{D}, \mathbf{x}, \mathbf{x}_\star) \approx \frac{1}{Z_0} \left[\prod_{k=1}^{K} \int p(z_k \,|\, \mathbf{c}_k, \mathbf{x}) \mathcal{N}(\mathbf{c}_k \,|\, \mathbf{m}^{\mathbf{c}_k}, \mathbf{V}^{\mathbf{c}_k}) \, d\mathbf{c}_k \right] \times \\ \int \left[\int p(z_0 \,|\, \mathbf{f}, \mathbf{x}) \mathcal{N}(\mathbf{f} \,|\, \mathbf{m}_0, \mathbf{V}^{\mathbf{f}}) \, df_1 \cdots df_N \right] \Psi(\mathbf{z}, f_{N+1}) \, df_{N+1} \,. \end{aligned} \tag{4.17}$$

We first treat each of the K integrals in the product on the first line of Eq. (4.17). The

distribution $p(z_k \,|\, \mathbf{c}_k, \mathbf{x})$ is a Gaussian marginal distribution at $\mathbf{x}$ given by the GP conditioned on the data $\mathbf{c}_k$ at the locations $\mathbf{x}_1, \ldots, \mathbf{x}_N$, which is given by Eq. (2.1). This distribution has the special property that its variance only depends on the locations $\mathbf{x}_1, \ldots, \mathbf{x}_N$, and does not depend on the $\mathbf{c}_k$. Furthermore, Eq. (2.1) shows that the mean of this distribution depends linearly on $\mathbf{c}_k$. And, combining the facts that $\mathbf{c}_k$ follows a Gaussian distribution, and that a linear function of a Gaussian random variable is another Gaussian random variable, we have that the mean of $p(z_k \,|\, \mathbf{c}_k, \mathbf{x})$ also follows a Gaussian distribution. To recap, we can think of each integral in the top line of Eq. (4.10) as the distribution of z_k, a Gaussian random variable with fixed variance whose mean is also Gaussian. This is a familiar situation: after marginalizing out the random Gaussian mean, we are left with yet another Gaussian distribution. Thus, the variable $\mathbf{c}_k$ is integrated out analytically. The mean and variance of the resulting univariate Gaussian distribution in z_k, which we call m'_k and v'_k respectively, are given in Appendix A.4.

We now turn our attention to the inner integral in the lower line of Eq. (4.10). The situation is precisely analogous to the above, except that we do not integrate out one of the variables, namely f_{N+1}. In integrating out the variables $f_1, \ldots, f_N$ we are left with a Gaussian distribution with fixed variance and mean given by a linear function of the Gaussian random variable f_{N+1}. The joint distribution $p([z_0, f_{N+1}] \,|\, \mathbf{x})$ is then a bivariate Gaussian, with mean and variance $\mathbf{m}'_\mathbf{f}$ and $\mathbf{V}'_\mathbf{f}$ given in Appendix A.4. We are now left with:

$$p(\mathbf{z} \,|\, \mathcal{D}, \mathbf{x}, \mathbf{x}_\star) \approx \frac{1}{Z} \left[\prod_{k=1}^{K} \mathcal{N}\left(z_k \,|\, m'_k, v'_k\right) \right] \int \mathcal{N}\left([z_0, f_{N+1}] \,|\, \mathbf{m}'_\mathbf{f}, \mathbf{V}'_\mathbf{f}\right) \Psi(\mathbf{z}, f_{N+1}) \, df_{N+1} \,, \quad (4.18)$$

where Z is a normalization constant. Note that the means and variances of the Gaussians in Eq. (4.18) do depend on $\mathbf{x}$. Thus the computations leading from Eq. (4.16) to Eq. (4.18), and all that follow, must be done for every $\mathbf{x}$, whereas the EP approximation to $q(\mathbf{f}, \mathbf{c}_1, \ldots, \mathbf{c}_K)$ need only be performed once per iteration of Bayesian optimization.

We now make the final approximation, which is to approximate the right-hand side of Eq. (4.18) with a product of independent Gaussians:

$$p(\mathbf{z} \,|\, \mathcal{D}, \mathbf{x}, \mathbf{x}_\star) \approx \mathcal{N}\left(z_0 \,|\, \mu_f^{\mathrm{NFCPD}}(\mathbf{x}), v_f^{\mathrm{NFCPD}}(\mathbf{x})\right) \prod_{k=1}^{K} \mathcal{N}\left(z_k \,|\, \mu_k^{\mathrm{NFCPD}}(\mathbf{x}), v_k^{\mathrm{NFCPD}}(\mathbf{x})\right) \,, \quad (4.19)$$

where $\mu_f^{\text{NFCPD}}(\mathbf{x})$ and $v_f^{\text{NFCPD}}(\mathbf{x})$ are the approximate mean and variance of $z_0 = f(\mathbf{x})$ and $\mu_k^{\text{NFCPD}}(\mathbf{x})$ and $v_k^{\text{NFCPD}}(\mathbf{x})$ are the approximate mean and variance of $z_k = c_k(\mathbf{x})$; the corresponding expressions are given in Appendix A.5. Finding these parameters is equivalent to performing one final iteration of EP for a single non-Gaussian factor $\Psi(\mathbf{z}, f_{N+1})$, which we must do for every $\mathbf{x}$. Note: it is also possible to compute the full covariance matrix of the right-hand size of Eq. (4.18) and thus approximate the NFCPD with a multivariate Gaussian, but we elect to approximate Eq. (4.18) with a product of independent Gaussians. This way, the PESC acquisition function is easily decomposed into the separate contributions of the different functions (see Chapter 5).

4.1.5 The PESC acquisition function

We have now approximated the NFCPD with independent Gaussians and are almost ready to compute its entropy with Eq. (2.14). The last step is to translate our approximate NFCPD into an approximate CPD. Since our approximation to the NFCPD is Gaussian and the noise is i.i.d. Gaussian, our approximation to the CPD is also a product of independent Gaussians with means $\mu_f^{\text{CPD}}, \mu_1^{\text{CPD}}, \ldots, \mu_K^{\text{CPD}}$ and variances $v_f^{\text{CPD}}, v_1^{\text{CPD}}, \ldots, v_K^{\text{CPD}}$ given by

$$
\begin{aligned}
v_f^{\text{CPD}} &= v_f^{\text{NFCPD}} + \nu_0 \\
v_k^{\text{CPD}} &= v_k^{\text{NFCPD}} + \nu_k \text{ for } k = 1, \ldots, K\,.
\end{aligned}
$$

The PESC acquisition function, which approximates Eq. (4.1), is then

$$
\alpha_{\text{PESC}}(\mathbf{x}) = \left\{ \log v_f^{\text{PD}}(\mathbf{x}) + \sum_{k=1}^{K} \log v_k^{\text{PD}}(\mathbf{x}) \right\} - \frac{1}{M} \sum_{m=1}^{M} \left\{ \log v_f^{\text{CPD}}\left(\mathbf{x} \,|\, \mathbf{x}_\star^{(m)}\right) + \sum_{k=1}^{K} \log v_k^{\text{CPD}}\left(\mathbf{x} \,|\, \mathbf{x}_\star^{(m)}\right) \right\}, \quad (4.20)
$$

where M is the number of samples drawn from $p(\mathbf{x}_\star \,|\, \mathcal{D})$, $\mathbf{x}_\star^{(m)}$ is the mth of these samples, $v_f^{\text{PD}}(\mathbf{x})$ and $v_k^{\text{PD}}(\mathbf{x})$ are the predictive variances for the noisy evaluations of f and c_k at $\mathbf{x}$, respectively, and $v_f^{\text{CPD}}\left(\mathbf{x} \,|\, \mathbf{x}_\star^{(m)}\right)$ and $v_k^{\text{CPD}}\left(\mathbf{x} \,|\, \mathbf{x}_\star^{(m)}\right)$ are the approximated variances of the CPD for the

noisy evaluations of f and c_k at $\mathbf{x}$ given that $\mathbf{x}_\star^{(m)}$ is the global minimizer. In Eq. (4.20) we have neglected the constant factor of $\frac{(K+1)}{2}\log(2\pi e)$ and the multiplicative factor of $\frac{1}{2}$ for simplicity, since these do not affect the maximizer of the acquisition function.

4.1.6 Summary of approximations made in PESC

To recap, the following approximations were made in going from Eq. (4.1) to Eq. (4.20):

1. The expectation over $\mathbf{x}_\star$ is approximated with Monte Carlo sampling.

2. The Monte Carlo samples of $\mathbf{x}_\star$ are drawn approximately using a finite basis function approximation to the GP kernels.

3. We approximate the NFCPD by replacing the infinite-dimensional integration variables $f, c_1, \ldots, c_K$ with the finite-dimensional integration variables $\mathbf{f}, \mathbf{c}_1, \ldots, \mathbf{c}_K$, and making the corresponding changes to the integrand (Eqs. (4.12) and (4.13)).

4. We further approximate this finite-dimensional approximation to the NFCPD with a product of Gaussians using EP. This is done by first approximating the factors that do not depend on $\mathbf{x}$ and then, for each $\mathbf{x}$, performing the final iteration of the approximation.

In Section 6.2.1, we assess the accuracy of these approximations and show that PESC performs almost as well as a ground-truth method based on rejection sampling in one dimension.

4.1.7 Efficient marginalization of the GP hyperparamters

Marginalization of Eq. (4.20) over the GP hyper-parameters can be done efficiently as in Hernández-Lobato et al. (2014). In particular, one can rewrite Eq. (4.20) with the sum over m on the outside, and showing explicit dependence on the GP hyperparameters θ:

$$\alpha_{\mathrm{PESC}}(\mathbf{x}) = \frac{1}{M}\sum_{m=1}^{M}\Bigg[\left\{\log v_f^{\mathrm{PD}}\left(\mathbf{x}\,|\,\theta^{(m)}\right) + \sum_{k=1}^{K}\log v_k^{\mathrm{PD}}\left(\mathbf{x}\,|\,\theta^{(m)}\right)\right\} - \left\{\log v_f^{\mathrm{CPD}}\left(\mathbf{x}\,|\,\mathbf{x}_\star^{(m)}, \theta^{(m)}\right) + \sum_{k=1}^{K}\log v_k^{\mathrm{CPD}}\left(\mathbf{x}\,|\,\mathbf{x}_\star^{(m)}, \theta^{(m)}\right)\right\}\Bigg] \quad (4.21)$$

where $\theta^{(m)}$ is the mth sample of the GP hyperparameters. Since m is now an index over joint states of the GP hyperparameters and $\mathbf{x}_\star$, we can marginalize over GP hyperparameters without adding additional complexity (in contrast with, for example, ES (Section 2.4.6)).

4.2 PESC for fast black-box functions

One disadvantage of using PESC is that sampling $\mathbf{x}_\star$ and computing the EP approximations can be slow. If PESC is slow compared to the black-box function(s) of interest, the entire Bayesian optimization procedure may be inefficient because too much time is spent in the optimizer and thus too few function evaluations are performed. While this issue can be circumvented in the coupled case by, for example, switching to a faster method like EIC or abandoning Bayesian optimization entirely, it becomes essential the cast of decoupled tasks with varying speeds because we require a method that can straddle this gap: the method must sparingly evaluate the slow task(s) is in standard Bayesian optimization, while also making use of the speed of the fast tasks to evaluate them many times. To address this issue we introduce an (even more) approximate but very fast version of PESC, which we call PESC-F. The two main challenges are then how to make one iteration of PESC faster, and how, at each iteration, to decide which (the slow or the fast) method should be used. The following two subsections address these two issues, respectively.

4.2.1 PESC-F: an even more approximate, but fast, version of PESC

The main operations at each iteration of Bayesian optimization (when using PESC) are

1. Sampling GP hyperparameters and inverting the kernel matrix
2. Sampling $\mathbf{x}_\star$ and performing EP
3. Maximizing the acquisition function

We propose ways to shorten each of these steps. We shorten step 1 by skipping the GP hyperparameter sampling. Rather, we average the acquisition function over the hyperparameter samples already obtained at an earlier iteration (doing this allows for additional speedups using rank-one

updates to the Cholesky decomposition of the kernel matrix; see Appendix B.10.1). Similarly, we shorten step 2 by skipping the sampling of $\mathbf{x}_\star$ and instead using previously obtained samples. We also reuse a previously computed EP approximation instead of recomputing it. In particular, we reuse the approximations $\tilde{h}_n$ and $\tilde{g}_k$ in Eq. (A.9) and then recompute the approximation to $q(\mathbf{f}, \mathbf{c}_1, \ldots, \mathbf{c}_K)$ (Eq. (4.12)) using Eq. (A.11). This allows us to skip all the computations in Appendix A.3. We then perform the remaining calculations given in Section 4.1 in the usual way. Since computing the EP approximation to $q(\mathbf{f}, \mathbf{c}_1, \ldots, \mathbf{c}_K)$ is much more expensive than the ensuing computations, this caching strategy leads to a significant speedup. Finally, we shorten step 3 by using a coarser termination condition tolerance when maximizing the acquisition function, so that optimization converges faster but with reduced precision. Furthermore, if the acquisition function is maximized using a local optimizer with random restarts and/or a grid initialization, we can shorten the computation further by reducing the number of restarts and/or grid size.

4.2.2 Choosing whether to run the full or approximate PESC

The motivation for PESC-F is that optimizer time should be small compared to the time spent evaluating the function(s). Given this, our approach to switching between the full and fast acquisition function is to hold approximately constant the fraction of total wall-clock time spent in the optimizer. To achieve this, at each iteration of Bayesian optimization, we do a full update if and only if

$$\frac{\tau_{\text{current}} - \tau_{\text{last-full}}}{\tau_{\text{full}}} > \gamma\,, \tag{4.22}$$

where τ_{current} is the current time, $\tau_{\text{last-full}}$ is the time since the last full update, τ_{full} is the duration of the last full update (this includes fitting the model and any other extra overhead, i.e., all of lines 3-4 in Algorithm 1), and $\gamma > 0$ is some constant, which we call the rationality level because, the larger the value of γ, the more time spent in rational decision making (i.e., optimizer time). This scheme is illustrated in Algorithm 3.

One could also imagine replacing τ_{full} with an average over the durations of past full updates.

Algorithm 3 Constrained Bayesian optimization for possibly fast tasks with PESC-F.

1: **Inputs:** black-box functions $f, c_1, \ldots, c_K$, model $\mathcal{M}$, initial design $\mathcal{D}$, rationality level γ
2: initialize $\tau_{\text{last-full}} = \tau_{\text{full}} = 0$
3: **repeat**
4: set τ_{current} to the current time
5: **if** $(\tau_{\text{current}} - \tau_{\text{last-full}})/\tau_{\text{full}} > \gamma$ **then**
6: sample GP hyperparameters θ
7: fit GP to the data $\mathcal{D}$
8: sample $\mathbf{x}_\star$ and perform EP
9: set τ_{full} to the current time minus τ_{current} (the elapsed time since executing line 4)
10: set $\tau_{\text{last-full}}$ to the current time
11: **else**
12: fit GP to the data $\mathcal{D}$
13: perform fast update to Gaussian NFCPD approximation as described in Section 4.2.1
14: **end if**
15: maximize the acquisition function: $\hat{\mathbf{x}} = \arg\max_{\mathbf{x}} \alpha_{\text{PESC}}(\mathbf{x} \,|\, \mathcal{M})$
16: evaluate the function: $\hat{y} = f(\hat{\mathbf{x}})$
17: add the new data to the data set: $\mathcal{D} = \mathcal{D} \cup \{(\hat{\mathbf{x}}, \hat{y})\}$
18: **until** termination condition is met
19: **Output:** the recommendation $\mathbf{x}_\star = \arg\min_{\mathbf{x} \in \mathcal{X}} \mathbb{E}_{\mathcal{M}}[f(\mathbf{x})]$ s.t. $\mathcal{C}(\mathbf{x})$

While this approach is less noisy, we opt for using only the duration of the most recent full update since these durations may exhibit deterministic trends. For example, the modeling time tends to increase at each iteration due to the increase in data set size. If indeed the full update duration increases monotonically, then the duration of the most recent update would be a more accurate estimate of the next full update duration than the average duration of all past updates.

PESC-F can be used as a generalization of PESC, since it reduces to PESC in the case of sufficiently slow function evaluations. To see this, we observe that if the black-box function is in fact very slow, then $\tau_{\text{current}} - \tau_{\text{last-full}} > \gamma\tau_{\text{full}}$ will be satisfied at every iteration for a reasonable choice of γ. Thus, the method will always perform the full updates as we would hope. On the other hand, if the black-box function is very fast, this method will mainly perform fast updates but will still occasionally perform full updates occasionally, with a frequency roughly proportional to the function evaluation duration.

4.2.3 Setting the rationality level in PESC-F

Loosely speaking, this algorithm is devised such that the ratio of optimizer time to function evaluation time is at most γ. This notion is approximate because, for example, the time spent in function evaluations actually includes the time spent in fast updates. The ideal setting of γ may depend on the problem, but we propose values on the order of $\gamma = 0.1$ to 1, which correspond to spending roughly $50-90\%$ of the total time performing function evaluations. It is possible that the ideal setting of γ changes as optimization proceeds and it is very likely that it is very problem-dependent; this is a subject for future research. The notion considering the value of computation is called *bounded rationality* and has its roots in the traditional AI literature; for example, in Russell et al. (1991) computation is treated as a possible action that consumes time but increases the expected utility of future actions.

4.3 Discussion

4.3.1 Separability of the acquisition function

The PESC acquisition function is additive in the expected amount of information that is obtained from the evaluation of each function (objective or constraint). For example, the expected information gain obtained from evaluating $f(\mathbf{x})$ is given by the term

$$\log v_f^{\mathrm{PD}}(\mathbf{x}) - \frac{1}{M}\sum_{m=1}^{M} \log v_f^{\mathrm{CPD}}\left(\mathbf{x} \,|\, \mathbf{x}_\star^{(m)}\right)$$

in Eq. (4.20). The other K terms in Eq. (4.20) measure the corresponding contributions from evaluating each of the constraints. This allows PESC to easily address the decoupled scenario when one can independently evaluate the different functions at different locations (Chapter 5). In other words, Eq. (4.20) is a sum of individual acquisition functions, one for each function that we can evaluate. Existing methods for Bayesian optimization with unknown constraints do not possess this desirable separability property. Note: the key property of the PESC acquisition function, from the perspective of decoupled constraints, is that it decomposes into the contributions for each task; the fact that this decomposition is additive is only incidental.

4.3.2 Computational complexity

The complexity of PESC is $\mathcal{O}(MKN^3)$ in the coupled setting, where M is the number of $\mathbf{x}_\star$ samples, K is the number of constraints, and N is the number of collected data for each task. Assuming M is also the number of GP hyperparameter samples, this is the same computational complexity as EIC because the bottleneck is the same $\mathcal{O}(N^3)$ matrix inversion; however, in practice PESC is slower. In the decoupled setting this becomes $\mathcal{O}(M\sum_{k=0}^{K}(N_0+N_k)^3)$ where N_0 is the number of evaluations of the objective and N_k is the number of evaluations for constraint constraint k.

4.3.3 Relationship to PES

If one applies the PESC method described in this Chapter but without constraints (i.e., setting $K=0$), one is left with a method that is very similar to PES (Section 2.4.7) but not exactly identical. In PES, Hernández-Lobato et al. (2014) introduce additional factors into their approximation to $p(f \mid \mathbf{x}_\star)$ in order to enforce the conditions that $\nabla f = 0$ at $\mathbf{x}_\star$ and that the Hessian is positive definite at $\mathbf{x}_\star$. We do not enforce these conditions since the global optimum may be on the boundary of the feasible region, in which case these conditions do not necessarily hold (in fact, even in PES this issue arises because the optimum may be on the boundary of the domain $\mathcal{X}$). Furthermore, in PESC we condition on the entire functions $f, c_1, \ldots, c_K$ and impose the condition that all feasible values must be above the global minimum. In contrast, in PES this conditioning on f is avoided by instead using the following approximate condition: we take the best observed value and the impose a soft condition (to account for noise in the observation) that this must be above $f(\mathbf{x}_\star)$. This amounts to changing a step function Θ to a Gaussian CDF Φ because the noise is Gaussian.

4.3.4 Relationship to Thompson Sampling

In Fig. 4.1, and in particular Figs. 4.1(d) and 4.1(i), the maximizer of the PESC acquisition function is quite close to the sampled $\mathbf{x}_\star^{(m)}$. Is PESC then simply a more complicated version of Thompson sampling (TS) (Section 2.4.3)? In fact, PESC (and PES) are related to TS, but we argue that, computational cost aside, they are better methods. PES/PESC can be seen as

less noisy versions of TS since they average the contributions of M samples of $\mathbf{x}_\star$, whereas TS relies on a single sample and offers no simple way to perform averaging. Furthermore, even with $M = 1$ the global maximum of α_{PESC} is not necessarily near $\mathbf{x}_\star$, since conditioning on $\mathbf{x}_\star$ makes global changes to the predictive distribution as shown in Fig. 4.1; in other words, evaluating where $p(\mathbf{x}_\star)$ is low may still be informative, depending on the correlation structure of the problem. Finally, PESC allows us to separate the contributions of evaluating the objective and constraint(s), whereas TS does not offer this benefit.

4.3.5 PESC and probabilistic constraints

PESC can be used in combination with probabilistic constraints (Section 3.1). Like EIC, the PESC acquisition function does not depend on the confidence threshold(s) associated with the probabilistic constraints; rather, these thresholds only apply when making recommendations.

4.3.6 Bridging the gap between fast and slow

Section 4.2 introduces the PESC-F method that can switch between fast, yet inaccurate, and accurate, but slow, optimizer steps. Of course, if the objective function is extremely fast then Bayesian optimization may not be a good method at all. The goal of PESC-F is to keep the ratio of time spent in the optimization steps to time spent in function evaluations below a user-specified value. However, if the function evaluation is so fast that the time spent in the fast updates becomes significant or even the dominant factor, then this assumption will be violated. We have already defined τ_{full} as the duration of a full update. Let us also define τ_{fast} as the duration of a fast update and τ_{func} as the duration of a fast update. Therefore, we can say that the PESC-F method is most useful when $\tau_{\text{fast}} < \tau_{\text{func}} < \tau_{\text{full}}$. When $\tau_{\text{func}} > \tau_{\text{full}}$, PESC-F also works well because it automatically reverts to standard PESC with full updates at every iteration.

Many aspects of this method are not specific to PESC and could easily be adapted to other acquisition functions like EIC or even unconstrained acquisition functions like PES and EI. In particular, lines 8 and 13 of Algorithm 3 are specific to PESC, whereas all other speedup methods are general. For example, when using vanilla unconstrained EI, the computational bottleneck

is likely to be sampling the GP hyperparameters (Algorithm 3, line 6) and maximizing the acquisition function (Algorithm 3, line 15). The ideas presented above, namely to skip GP hyperparameter sampling and to optimize the acquisition function with a smaller grid and/or coarser tolerances, are applicable in this situation and might be useful in the case of a fairly fast objective function.

In the $\tau_{\text{func}} < \tau_{\text{fast}}$ case, model-based methods are likely not to be the best approach; a methods like CMA-ES (Section 2.9.1) is likely to work better because it will perform many more function evaluations in a fixed time. However, the user may not know these relative time scales *a priori* and thus may not know which algorithm to use. Furthermore, in a decoupled or multi-task scenario, the user might be faced with some very slow tasks and some very fast tasks. In this case, neither a slow model-based method nor a fast method like CMA-ES are well-suited to the entire problem, and thus the problem is more that just a matter of choosing the right algorithm. Given this, an interesting duration of future research is to create algorithms that bridge the gap between fast and slow function evaluations, doing something reasonable in either case.

Chapter 5

Decoupled Constraints and Resource Allocation

5.1 Motivation

Consider the problem discussed in Section 1.1 of optimizing the performance of a speech recognition system such that it must perform its functionality within a specified time limit. The system may be implemented, for example, as a neural network with hyperparameters such as the number of hidden units, learning rate, etc. The objective function is classification error on a validation data set and there is a single constraint imposing a maximum runtime. Prior work on constrained Bayesian optimization (Section 2.6) focuses on the *coupled* scenario, in which the objective and constraint(s) must be evaluated jointly; in other words, in the coupled scenario one cannot evaluate one function without evaluating the other(s). Here we introduce Bayesian optimization with *decoupling*, in which these different tasks (objective and constraints) need not be jointly evaluated. The neural network example is naturally decoupled, because the running time (constraint) can presumably be evaluated without training the network, which must be done to evaluate the objective.

Some problems exhibit decoupling while others do not. An example of a coupled problem is a financial simulation that draws many samples from the distribution of possible outcomes. If the objective function is the expected profit and the constraint is a maximum tolerable probability

of default, then these two functions are computed jointly by the same simulation and are thus coupled to each other. Another example of decoupling is the cookie optimization discussed in Chapter 1: determining the tastiness of a cookie recipe requires a taste test with a group of participants, whereas computing the number of calories can be done independently without such a group.

As discussed in Section 3.3, most existing constrained Bayesian optimization methods, which are based on EI, are not easily extended to decoupled problems. Then, a natural question to ask is, why worry about decoupling? Can we not simply treat decoupled problems as coupled, by always evaluating all tasks at the same locations, thus enabling us to leverage existing methods? We address this question with a thought experiment here, and with empirical evaluations in Chapter 6.

Returning to our time-limited neural network example, let us consider the expense of evaluating each of the two tasks. Evaluating the objective (the classification error on a validation set) requires training the neural network, and perhaps consumes minutes to hours of compute time. On the other hand, evaluating the constraint (running time) requires only to make predictions with the network using random weights, since this will give the same results as with the trained weights. Measuring the running time takes perhaps a few seconds; in short, it is orders of magnitude faster than evaluating the objective. Imagine evaluating both tasks at some $\mathbf{x}$ only to discover immediately that the run time is extremely slow, and thus the run time constraint is violated. To continue training the neural network is then extremely wasteful, because $\mathbf{x}$ (and the nearby region of input space) has already been ruled out. In a decoupled framework, one could first measure the run time at many locations, gaining a sense of the constraint surface. Only then would we incur the significant expense of objective function evaluations, heavily biasing our search towards locations that are deemed probably feasible. The larger the cost differential between the two tasks, the more wasteful it becomes to treat the problem as coupled, potentially incurring enormous expense for values of $\mathbf{x}$ that could immediately be ruled out by a decoupled optimization algorithm.

In other problems, it may be the constraint that is slow and the objective that is fast. Or, we may have multiple tasks, all of which are slow, but some of which are much more informative than

others (an example of a less informative task is a constraint that is satisfied almost everywhere, including at the true solution). In this chapter we aim to build a general framework of Bayesian optimization that encompasses all of these cases.

5.2 Competitive vs. non-competitive decoupling

I divide the class of decoupled problems into two sub-classes, which I call *competitive decoupling* (CD) and *non-competitive decoupling* (NCD). CD is the form of decoupling considered in Section 3.3, in which two or more tasks compete for the same *resource*. In our example, the resource is the CPU being used to perform the optimization. On the other hand, NCD refers to the case in which the different tasks can be evaluated independently, but using different resources. For example, consider the case in which the data used to evaluate our objective function are highly private, and thus the objective function evaluation can only be performed on a particular machine with access to these data. On the other hand, if the constraint evaluation does not require this data set, it could be performed on any machine, or perhaps even in parallel on a large number of available CPUs. In this case the objective and constraint are still decoupled in the sense that they need not be evaluated jointly, but they do not compete for the same resource (CPU) and thus can be run asynchronously but in parallel.

Section 2.7 discusses Bayesian optimization in which multiple function evaluations can be performed in parallel. NCD is a generalization of this notion. With NCD, one can run multiple experiments in parallel, and furthermore these experiments may be of incompatible types that must be run on different resources. Single-task parallel Bayesian optimization can thus be thought of as a specific case of NCD in which there is only one resource, with some capacity given by the maximum number of concurrent experiments. NCD is in particular related to the asynchronous parallel Bayesian optimization methods discussed in Section 2.7. In asynchronous parallelism, jobs are submitted asynchronously, and we must make further suggestions using the information that some experiments are pending. This is precisely the case in NCD, except that the pending experiments are not necessarily the same tasks as that the of next suggestion.

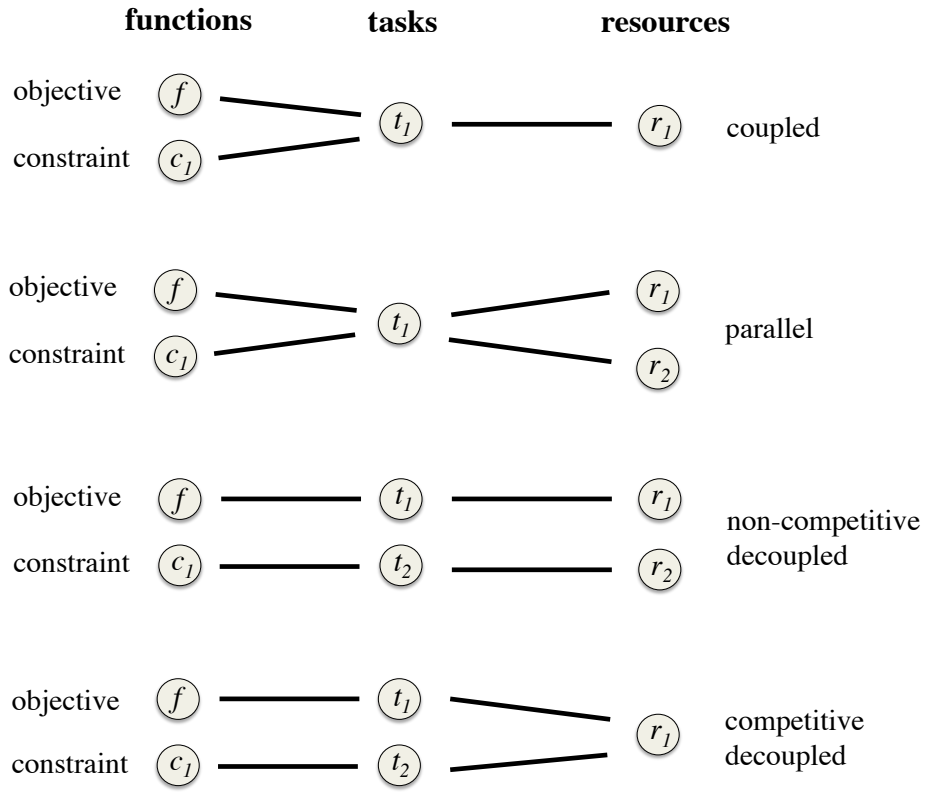


Figure 5.1: Schematic comparing the coupled, parallel, non-competitive decoupled (NCD), and competitive decoupled (CD) scenarios for a problem with a single constraint c_1. In each of the three cases, the mapping between tasks and resources (the right-hand portion of the figure) is the bipartite graph $\mathcal{G}$.

5.3 Formalization

In this section we formalize a general scenario in which the functions $\{f, c_1, \ldots, c_K\}$ in our optimization problem may exhibit a combination of coupling and decoupling. Let $\mathcal{F}$ be the set of functions $\{f, c_1, \ldots, c_K\}$ and let the set of tasks $\mathcal{T}$ be a partition of $\mathcal{F}$ indicating which functions are coupled (i.e. must be jointly evaluated). Let $\mathcal{R} = \{r_1, \ldots, r_{|\mathcal{R}|}\}$ be the set of resources available to solve this problem. Then, we encode the relationship between tasks and resources using a bipartite graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with node set $\mathcal{V} = \mathcal{T} \cup \mathcal{R}$ and edges $\{t \sim r\} \in \mathcal{E}$ such that $t \in \mathcal{T}$ and $r \in \mathcal{R}$. The interpretation of an edge $\{t \sim r\}$ is that task t can be performed on resource r. Note that this framework does not encompass the case in which a task requires multiple resources to be executed; we leave this to future work.

In addition to this basic framework, we also introduce a *capacity* $\omega_{\max}$ for each resource r. The capacity $\omega_{\max}(r) \in \mathbb{Z}^+$ represents how many tasks may be simultaneously executed on resource r; for example, if r represents a cluster of CPUs, $\omega_{\max}(r)$ would be the number of CPUs in the cluster. Introducing the notion of capacity is simply a matter of convenience; in fact, it is equivalent to setting all capacities to one and replicating each resource node in $\mathcal{G}$ according to its capacity.

We now possess a formalism to describe coupling as well as NCD and CD. In particular, coupling describes two functions f_1 and f_2 that belong to the same task t; if this task can be performed on multiple resources (or one resource with $\omega_{\max} > 1$), then this is parallel Bayesian optimization. NCD describes two functions f_1 and f_2 that belong to different tasks t_1 and t_2, which themselves require different resources r_1 and r_2 (i.e. $t_1 \sim r_1$ and $t_2 \sim r_2$). CD describes two functions f_1 and f_2 that belong to *different* tasks t_1 and t_2 (decoupled) requiring the *same* resource r (competitive). These descriptions correspond to the visual representation in Fig. 5.1. These definitions can be trivially extended to the case of more than two functions. The most general case is an arbitrary task-resource graph $\mathcal{G}$ encoding a combination of these situations.

5.4 Algorithm for decoupled optimization

In competitive decoupling, the acquisition function must be capable of comparing the relative benefits of evaluating different tasks. As discussed in Section 4.3.1, the PESC acquisition function conveniently yields the contribution of each function; these are then added to yield the contribution of the task as a whole. On the other hand, the EIC-D method introduced in Section 3.3 is not separable as discussed in Section 3.4.3.

Algorithm 4 is a general algorithm for Bayesian optimization given a set of functions $\mathcal{F}$, tasks $\mathcal{T}$, resources $\mathcal{R}$, and a task-resource graph $\mathcal{G}$. In line 3, we loop through all resources r that are not already at capacity; i.e., those resource whose number of currently running jobs $\omega(r)$ is less than its capacity $\omega_{\max}(r)$. In line 4, we loop through all tasks in $\mathcal{T}$ that can be run on resource r, as dictated by $\mathcal{G}$. At line 6, the acquisition function $\alpha_t(\mathbf{x})$ is maximized over $\mathbf{x}$ for a particular task t. The input $\mathbf{x}_t^*$ is the suggested evaluation location assuming we will

Algorithm 4 A generalized Bayesian optimization framework.

1: **Input:** functions $\mathcal{F}$, tasks $\mathcal{T}$, resources $\mathcal{R}$, task-resource graph $\mathcal{G}$, acquisition function α, search space $\mathcal{X}$, model $\mathcal{M}$, initial data set $\mathcal{D}$, resource query functions $\omega, \omega_{\max}$
2: **repeat**
3: **for** $r \in \mathcal{R}$ s.t. $\omega(r) < \omega_{\max}(r)$ **do**
4: **for** $t \in \mathcal{T}$ s.t. $t \sim r \in \mathcal{G}$ **do**
5: Fit the model $\mathcal{M}$ to the data $\mathcal{D}$
6: Maximize the acquisition function: $\alpha_t^* = \max_{\mathbf{x} \in \mathcal{X}} \alpha_t(\mathbf{x}), \quad \mathbf{x}_t^* = \arg\max_{\mathbf{x} \in \mathcal{X}} \alpha_t(\mathbf{x})$
7: **end for**
8: $t^* = \arg\max_t \alpha_t^*$
9: Submit task t^* at input $\mathbf{x}_{t^*}^*$ for evaluation on resource r
10: **end for**
11: **until** termination condition is met
12: **Output:** the recommendation $\mathbf{x}_\star = \arg\min_{\mathbf{x} \in \mathcal{X}} \mathbb{E}_{\mathcal{M}}[f(\mathbf{x})]$ s.t. $\mathcal{C}(\mathbf{x})$

perform task t, and α_t^* is the associated acquisition function value. At line 8, we then maximize over tasks, selecting the task t^* for which α_t^* is largest. Upon doing so, the pair $(t^*, \mathbf{x}_{t^*}^*)$ now represents the experiment with the highest acquisition function value over all possible $(t, \mathbf{x})$ pairs in the decision space $\mathcal{T} \otimes \mathcal{X}$ that can be run on resource r. At line 9 we then begin this task and repeat this process until the termination condition is met.

I now describe how the algorithm above encompasses the coupled, non-competitive decoupled, and competitive decoupled cases. In the case of a single coupled task running on a single resource, the loops at lines 3 and 4 are both over a single element. At each iteration we simply maximize the acquisition function over $\mathbf{x} \in \mathcal{X}$ at line 6; the maximization over tasks at line 8 is a trivial maximization over only one task (in other words, we recover Algorithm 1). NCD is the same except that we loop over multiple elements at line 3. Finally, in CD the tasks "compete" at line 8 and the task t^* yielding the highest acquisition function is ultimately selected, along with its best location $\mathbf{x}_{t^*}^*$. It is straightforward to apply the fast method PESC-F (Section 4.2) by combining Algorithms 3 and 4. Algorithm 4 is not specific to PESC but can be applied with any separable acquisition function.

5.5 Incorporating cost information

The PESC acquisition function measures the number of bits of information about the location of the solution $\mathbf{x}_\star$ gained by evaluating task t at location $\mathbf{x}$. However, external considerations

may render evaluation of one task more desirable than another, given equal information content. The most salient of these factors is the running time or duration of an experiment. Snoek et al. (2012) consider this issue by automatically measuring the duration of function evaluations and modeling the duration as a function of $\mathbf{x}$ with an independent Gaussian process. Swersky et al. (2013) extend this concept over multiple tasks, so that for each task t an independent GP is used to model the unknown duration as a function of $\mathbf{x}$. This approach can be applied in Algorithm 4 by scaling the acquisition function for task t by the expected duration of t. In particular, we change line 8 to:

$$t^* = \arg\max_t \frac{\alpha_t^*}{\zeta_t}, \tag{5.1}$$

where ζ_t is the expected cost associated with task t. When using an information-based acquisition function such as PESC, this yields an acquisition function in the units of bits per second. When modeling task durations, the total number of models is the number of functions plus the number of tasks, i.e. $|\mathcal{F}|+|\mathcal{T}|$. Alternatively, one could fix the relative costs *a priori* instead of modeling the function evaluation durations. This is especially useful if there are costs other than time present.

In the above discussion we have associated a cost ζ_t to each task t. An even more general approach is to associate a cost to each task-resource allocation, or in other words to each edge of $\mathcal{G}$. If each task is only connected to a single resource in $\mathcal{G}$, these two formulations are the same. However, the weighting by costs need not only apply to selecting one task versus another; rather, we may be selecting one of several possible resources for the same task. This could occur, for example, because a neural network can be trained on either a CPU or GPU, with the former being much slower than the latter. The neural network task is then connected to two different resources (CPU and GPU) with different edge weights representing the different expected run times (either learned or set *a priori*).

5.6 Discussion

In the age of parallel computing, parallel Bayesian optimization algorithms are becoming increasingly important. In this chapter I generalized the notion of parallel Bayesian optimization by introducing the notion of non-competitive decoupling (NCD). We also introduced competitive decoupling (CD), a more complicated variant of decoupling in which multiple tasks must compete for the same resource. I then presented Algorithm 4, which can be used to optimize problems with an arbitrary combination of CD and NCD whose structure is represented by a bipartite graph $\mathcal{G}$. Taking advantage decoupling requires a separable acquisition function. PESC is one such separable acquisition function that performs well in practice. As new acquisition functions are developed in the future, they will hopefully be developed with separability in mind as an important and desirable property.

Chapter 6

Empirical Analyses

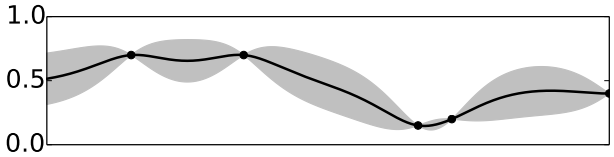
This chapter presents empirical analyses of the methods covered in Chapters 3 to 5. The implementation specifications for the experiments in this chapter are given in Appendix B.

6.1 Comparing EIC with an unconstrained baseline

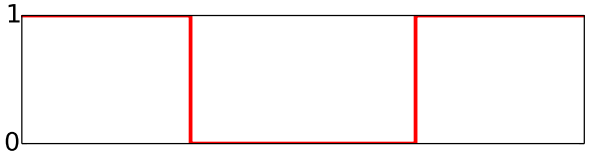
This section examines the need for constrained Bayesian optimization methods in the first place, by comparing EIC (Chapter 3) to a naive baseline in which constraint violations are set to some large penalty value and unconstrained Bayesian optimization is used as the optimizer. Naively applying unconstrained Bayesian optimization to problems with unknown constraints is problematic because the GP is easily confused by the sharp discontinuities that are artificially introduced into the penalized objective. Figure 6.1 illustrates this issue. The objective and constraint are shown in Figs. 6.1(a) and 6.1(b) respectively. When conditioned on "observations" of the penalized objective, the GP learns an artificially short length scale and thus loses its ability to generalize (Fig. 6.1(c)). The following subsections provide a quantitative comparison of EIC and this unconstrained baseline on three machine learning problems.

6.1.1 LDA with sparse topics

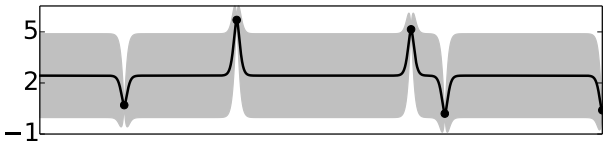
Latent Dirichlet Allocation (LDA) (Blei et al., 2003) is a popular model for learning topics, which are distributions over words, given a corpus of documents. In order for topics to have meaningful



(a) The objective, $f(x)$



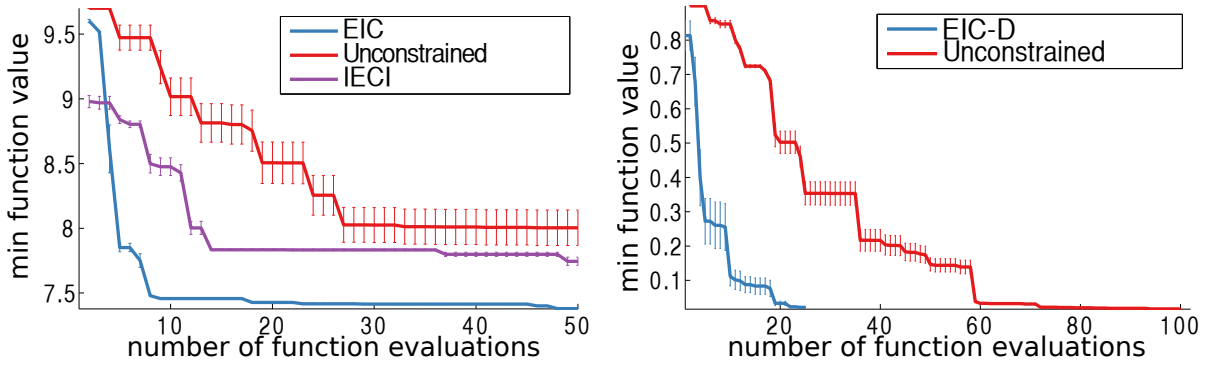
(b) The constraint, $\Theta(c(x))$



(c) The penalized objective, $f(x) + \lambda\Theta(c(x))$

Figure 6.1: Using a simple penalty to augment the objective function in one dimension. (a) The GP marginals (black line and gray fill) conditioned on 5 observations (black dots) of an objective function $f(x)$. (b) A Boolean constraint function $\Theta(c(x))$ that is violated in the middle and satisfied on the two sides (here, Θ denotes the Heaviside step function). (c) The GP marginals (black line and gray fill) when conditioned on the penalized objective, $f(x)+\lambda\Theta(c(x))$ with $\lambda = 5$ (black dots). The learned length scales are 0.09 in (a) and 0.006 in (c).

semantic interpretations, it is desirable for the word distributions to exhibit sparsity. Therefore this experiment considers optimizing the hyperparameters of LDA to minimize the perplexity of held out data subject to the constraint that the entropy of the per-topic word distribution averaged over topics is less than some threshold. This experiment considers LDA with stochastic variational inference (Hoffman et al., 2010) trained with the *vowpal wabbit* (Agarwal et al., 2014) package on 250,000 documents from Wikipedia, with a total vocabulary size of 10,473. The constraint threshold is set to $\log_2 200$ bits. This is achieved, for example, by allocating uniform density over 200 words. The search space consists of five hyperparameters corresponding to the number of topics (from 2 to 100), two Dirichlet distribution prior base measures (from 0 to 2), and two learning rate parameters (rate from 0.1 to 1, decay from 10^{-5} to 1). Figure 6.2(a) compares EIC to the unconstrained baseline on this task. In the baseline, the objective is set to the highest achievable objective function value when the constraint is violated.



(a) Online LDA (b) Neural Network

Figure 6.2: Empirical performance of EIC. (a) Comparing EIC (blue), IECI (purple, Section 2.6.2, Gramacy and Lee (2011)), and the unconstrained baseline (red) on tuning LDA with stochastic variational inference subject to topic sparsity. (b) Comparing EIC-D (blue, Section 3.3) with the unconstrained baseline (red) on tuning a deep neural network subject to a memory constraint. Errors bars indicate standard error from 5 independent runs.

6.1.2 Memory-limited neural net

This experiment considers optimizing the following 11 hyperparameters of a deep neural network on the MNIST handwritten digit classification task: 1 learning rate, 2 momentum parameters (initial and final, with linear interpolation), the number of hidden units per layer (2 layers), the maximum norm on model weights (for 3 sets of weights), and the dropout regularization probabilities (for the inputs and 2 hidden layers). The classification error on the standard validation set is minimized under the constraint that the total number of weights in the network must not exceed 10^6. This constraint is decoupled from the objective because the number of weights can be calculated without training the network. The neural network is trained using momentum-based stochastic gradient descent which is notoriously difficult to tune as training can diverge under various combinations of the momentum and learning rate. When training diverges, the objective function cannot be measured. This is treated as a hidden constraint (Section 2.3.3) with noisy observations. The network is trained for 25,000 weight updates with the *deepnet* package (`https://github.com/nitishsrivastava/deepnet`). Fig. 6.2(b) compares EIC with the unconstrained baseline on this task. Only the objective evaluations are presented, since constraint evaluations are extremely inexpensive compared to an entire training run. In the baseline, constraint violations are set to random performance (90% classification error).

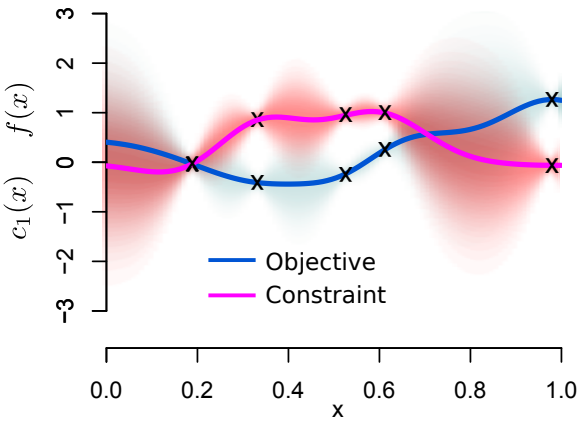
6.2 Analysis of PESC with coupled constraints

I evaluate the performance of PESC through experiments with synthetic functions sampled from the GP prior distribution, analytic benchmark problems previously used in the literature on Bayesian optimization with unknown constraints, and meta-optimization of machine learning algorithms with unknown constraints.

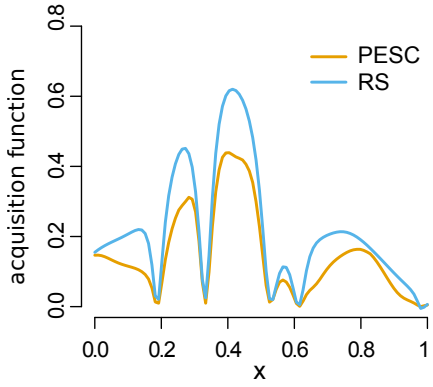
For first case listed above, the synthetic functions sampled from the GP prior are generated following the same experimental set up in Hennig and Schuler (2012); Hernández-Lobato et al. (2014). The search space is the unit hypercube of dimension D, and the ground truth objective f is a sample from a zero-mean GP with a squared exponential covariance function of unit amplitude and length scale $\ell = 0.1$ in each dimension. The function f is represented by first sampling from the GP prior on a grid of 1000 points generated using a Halton sequence (see Leobacher and Pillichshammer, 2014) and then defining f as the resulting GP posterior mean. I use a single constraint function c_1 whose ground truth is sampled in the same way as f. The evaluations for f and c_1 are contaminated with i.i.d. Gaussian noise with variance $\nu_f^2 = \nu_1^2 = 0.01$.

6.2.1 Accuracy of the PESC approximation

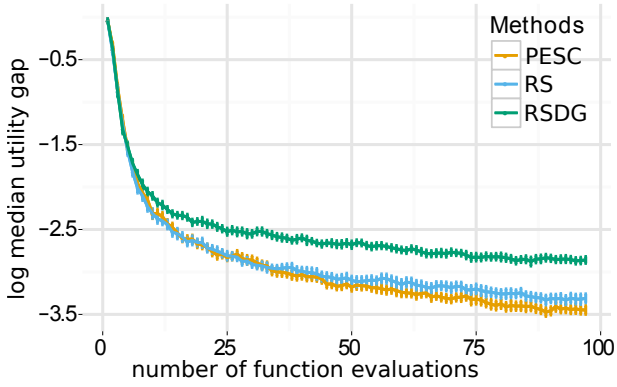
I first analyze the accuracy of the approximation to Eq. (4.1) generated by PESC. I compare the PESC approximation with a ground truth for Eq. (4.1) obtained by rejection sampling (RS). The RS method works by discretizing the search space using a uniform grid. The expectation with respect to $p(\mathbf{x}_\star \,|\, \mathcal{D})$ in Eq. (2.13) is then approximated by Monte Carlo. To achieve this, $f, c_1, \ldots, c_K$ are sampled on the grid and the grid cell with non-negative $c_1, \ldots, c_K$ (feasibility) and the lowest value of f (optimality) is selected. For each sample of $\mathbf{x}_\star$ generated by this procedure, $\mathrm{H}\left[y^f, y^1, \ldots, y^K \,|\, \mathcal{D}, \mathbf{x}, \mathbf{x}_\star\right]$ is approximated by rejection sampling: I sample the functions $f, c_1, \ldots, c_K$ and select those samples whose corresponding feasible optimal solution is the sampled $\mathbf{x}_\star$ and reject the other samples. I then assume that the selected samples for $f, c_1, \ldots, c_K$ are independent and have Gaussian marginal distributions. Under this assumption, $\mathrm{H}\left[y^f, y^1, \ldots, y^K \,|\, \mathcal{D}, \mathbf{x}, \mathbf{x}_\star\right]$ can be approximated using Eq. (2.14), with the variance parameters in this formula being equal to the empirical marginal variances of the selected samples of f and



(a) Marginal posteriors



(b) Acquisition functions



(c) Performance in $1D$

Figure 6.3: Assessing the accuracy of the PESC approximation. (a) Marginal posterior predictive distributions for the objective and constraint given some collected data denoted by $\times$'s. (b) PESC and RS acquisition functions given the data in (a). (c) Median utility gap for PESC, RS and RSDG in the experiments with synthetic functions sampled from the GP prior with $D = 1$.

$c_1, \ldots, c_K$ at $\mathbf{x}$ plus the corresponding noise variances ν_f^2 and $\nu_1^2, \ldots, \nu_K^2$.

Figure 6.3(a) shows the posterior distribution for f and c_1 given 5 observations sampled from the GP prior with $D = 1$. The posterior is computed using the optimal GP hyperparameters. The corresponding approximations to Eq. (4.1) generated by PESC and RS are shown in Fig. 6.3(b). In the figure, both PESC and RS use a total of 50 samples from $p(\mathbf{x}_\star \,|\, \mathcal{D})$ when approximating the expectation in Eq. (4.1). The PESC approximation is quite accurate, and importantly its maximum value is very close to the maximum value of the RS approximation.

One disadvantage of the RS method is its high cost, which scales with the size of the grid

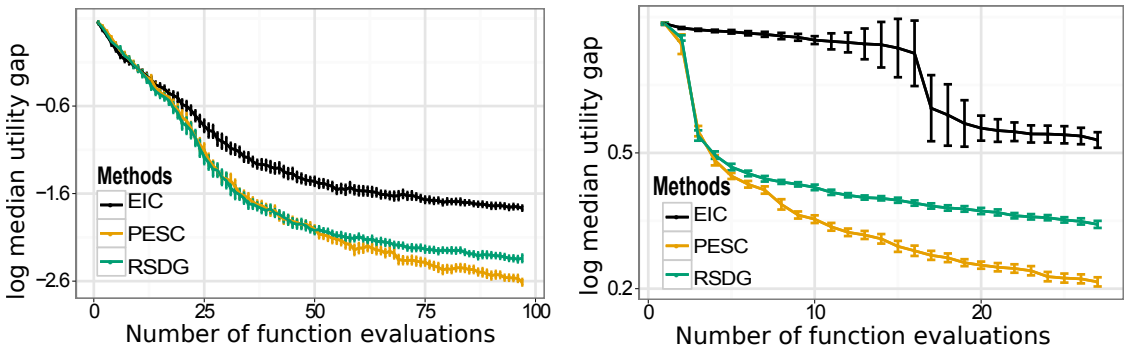
used. This grid has to be large to guarantee good performance, especially when D is large. An alternative is to use a small dynamic grid that changes as data is collected. Such a grid can be obtained by sampling from $p(\mathbf{x}_\star \,|\, \mathcal{D})$ using the same approach as in PESC to generate these samples (a similar approach is taken in Hennig and Schuler (2012), in which the dynamic grid is sampled from the EI acquisition function). The samples obtained then form the dynamic grid, with the idea that grid points are more concentrated in areas that we expect $p(\mathbf{x}_\star \,|\, \mathcal{D})$ to be high. The resulting method is called Rejection Sampling with a Dynamic Grid (RSDG).

I compare the performance of PESC, RS and RSDG in experiments with synthetic data corresponding to 500 pairs of f and c_1 sampled from the GP prior with $D = 1$. At each iteration, RSDG draws the same number of samples of $\mathbf{x}_\star$ as PESC. I fix $\delta_1 = 0.05$ and assume that the GP hyperparameter values are known to each method. For reporting purposes, I set the utility $u(\mathbf{x})$ of a recommendation $\mathbf{x}$ to be $f(\mathbf{x})$ if $\mathbf{x}$ satisfies the constraint, and otherwise a penalty value of the worst (largest) objective function value achievable in the search space. Each method is initialized with the same three random points drawn with Latin hypercube sampling.

Figure 6.3(c) shows the median of the utility gap (between the recommendation and true optimum) for each method across the 500 realizations of f and c_1. I report the median because the empirical distribution of the utility gap is heavy-tailed and in this case the median is more representative of the location of the bulk of the data than the mean. The heavy tails arise because I am averaging over 500 different optimization problems with very different degrees of difficulty. In this and all following experiments, unless otherwise specified, error bars are computed using the bootstrap. The plot shows that PESC and RS are better than RSDG. Furthermore, PESC is very similar to RS, with PESC even performing slightly better perhaps because PESC is not confined to a grid as RS is. These results show that PESC yields a very accurate approximation of the information gain.

6.2.2 Synthetic functions in 2 and 8 input dimensions

I compare the performance of PESC and RSDG with EIC using the same experimental protocol as in the previous section, but with dimensionalities $D = 2$ and $D = 8$. I do not compare with RS here because its use of grids does not scale to higher dimensions. Fig. 6.4 shows the utility



(a) $D = 2$ (b) $D = 8$

Figure 6.4: Optimizing samples from the GP prior with (a) $D = 2$ and (b) $D = 8$.

gap for each method across 500 different samples of f and c_1 from the GP prior with (a) $D = 2$ and (b) $D = 8$. Overall, PESC is the best method, followed by RSDG and EIC. RSDG performs similarly to PESC when $D = 2$, but is significantly worse when $D = 8$. This shows that, when D is high, grid based approaches (e.g. RSDG) are at a disadvantage with respect to methods that do not require a grid (e.g. PESC).

6.2.3 A toy problem

Next, I compare PESC with EIC and AL (Gramacy et al. (2014), Section 2.6.4) on the toy problem described in Gramacy et al. (2014), namely,

$$\min_{\mathbf{x}\in[0,1]^2} f(\mathbf{x}) \;\text{ s.t. }\; c_1(\mathbf{x}) \geq 0,\, c_2(\mathbf{x}) \geq 0\,, \tag{6.1}$$

where the objective $f(\mathbf{x})$ is given by $x_1 + x_2$ and the constraint functions c_1 and c_2 are given by

$$c_1(\mathbf{x}) = 0.5 \sin\left(2\pi(x_1^2 - 2x_2)\right) + x_1 + 2x_2 - 1.5\,,$$
$$c_2(\mathbf{x}) = -x_1^2 - x_2^2 + 1.5\,.$$

This optimization problem has two local minimizers and one global minimizer. At the global solution, which is at $\mathbf{x}_\star \approx [0.1954, 0.4404]$, only one of the two constraints (c_1) is active. Since the objective is linear and c_2 is not active at the solution, learning about c_1 is the main challenge of this problem.

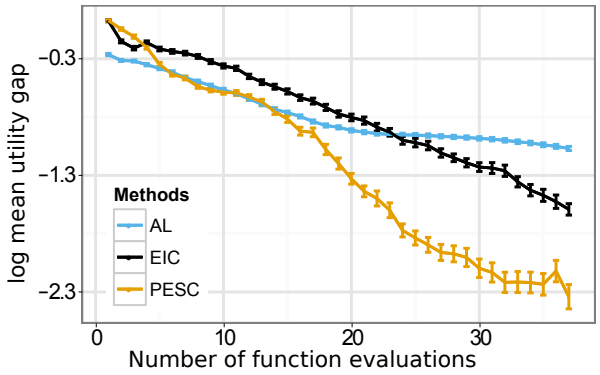


Figure 6.5: Comparing PESC, AL, and EIC in the toy problem from Gramacy et al. (2014).

In this experiment, the evaluations for f, c_1 and c_2 are noise-free. For PESC and EIC, I use $\delta_1 = \delta_2 = 0.025$ and a squared exponential GP kernel. PESC uses 10 samples of $\mathbf{x}_\star$ when approximating the expectation in Eq. (4.1). I use the AL implementation provided by Gramacy et al. (2014) in the R package *laGP* which is based on the squared exponential kernel and assumes the objective f is known. Thus, in order for this implementation to be used, AL has an advantage over other methods in that it has access to the true objective function. In all three methods, the GP hyperparameters are estimated by maximum likelihood.

Fig. 6.5 shows the mean utility gap for each method across 500 independent realizations. Each realization corresponds to a different initialization of the methods with three data points selected with Latin hypercube sampling. The results show that PESC is significantly better than EIC and AL for this problem. EIC is superior to AL, which performs slightly better at the beginning, possibly because it has access to the ground truth objective f.

6.2.4 Finding a fast neural network

In this experiment, I tune the hyperparamters of a three-hidden-layer neural network subject to the constraint that the prediction time must not exceed 2 ms on an NVIDIA GeForce GTX 580 GPU (also used for training). The search space consists of 12 parameters: 2 learning rate parameters (initial and decay rate), 2 momentum parameters (initial and final, with linear interpolation), 2 dropout parameters (input layer and other layers), 2 other regularization parameters (weight decay and max weight norm), the number of hidden units in each of the 3

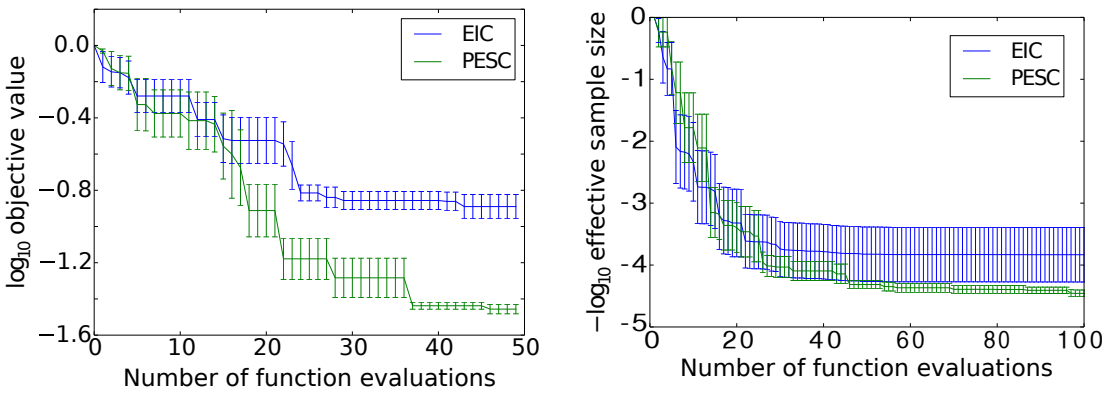
hidden layers, and the activation function (RELU or sigmoid). The network is trained using the *deepnet* package, and the prediction time is computed as the average time of 1000 predictions for minibatches of size 128. The network is trained on the MNIST digit classification task with momentum-based stochastic gradient descent for 5000 iterations. The objective is reported as the classification error rate on the standard validation set. For reporting purposes, I treat constraint violations as the worst possible objective value (a classification error of 1.0).

Figure 6.6(a) shows the results of 50 iterations of Bayesian optimization. In this experiment and the next, the y-axis represents observed function values and $\delta = 0.05$. Curves are the mean over 5 independent experiments. PESC performs significantly better than EIC.

When constraints are noisy, reporting the best observation is an overly optimistic metric (due to "lucky" evaluations); on the other hand, ground-truth is not available. Therefore, to validate our results further, I used the recommendations made at the final iteration of Bayesian optimization for each method (EIC and PESC) and evaluated the function with these recommended parameters. I repeated the evaluation 10 times for each of the 5 repeated experiments to compute a ground-truth score averaged of 50 function evaluations. This procedure yields a score of $7.0 \pm 0.6\%$ for PESC and $49 \pm 4\%$ for EIC (as in the figure, constraint violations are treated as a classification error of 100%). This result is consistent with Fig. 6.6(a) in that PESC performs significantly better than EIC. This experiment is inspired by a real need for networks with fast prediction time, especially in computer vision problems in which the abundance of data can grow faster than a trained network can make predictions (e.g. YouTube, connectomics).

6.2.5 Tuning Markov chain Monte Carlo

Hamiltonian Monte Carlo (HMC) (Duane et al., 1987) is a popular MCMC sampling technique that takes advantage of gradient information for rapid mixing. HMC contains several parameters that require careful tuning. The two basic parameters are the number of leapfrog steps, τ, and the step size, ϵ. HMC may also include a mass matrix which introduces $\mathcal{O}(D^2)$ additional parameters in D dimensions, although the matrix is often chosen to be diagonal (D parameters) or a multiple of the identity matrix (1 parameter) (Neal, 2011). In this experiment, I optimize the performance of HMC using Bayesian optimization. The search space consists of the following



(a) Finding a fast neural network.

(b) Tuning Hamiltonian Monte Carlo

Figure 6.6: Comparing PESC and EIC on machine learning problems with coupled constraints. (a) Tuning a neural network subject to the constraint that it makes predictions in under 2 ms. (b) Tuning Hamiltonian Monte Carlo to maximize the number of effective samples within 5 minutes of compute time, subject to the constraints passing the Geweke and Gelman-Rubin convergence diagnostics and integrator stability.

parameters: the number of leapfrog steps, τ, the step size, ϵ, a mass parameter, and the fraction of the allotted computation time spent burning in the chain.

The objective function is the number of effective samples; this corresponds to finding chains that minimize estimator variance. For each set of inputs, I compute two chains, each with 5 minutes of computation time on a single core of a compute node. I impose the constraints that the samples must pass the Geweke (Geweke, 1992) and Gelman-Rubin (Gelman and Rubin, 1992) convergence diagnostics. In particular, I require the worst (largest absolute value) Geweke test score across all variables and chains to be at most 2.0, and the worst (largest) Gelman-Rubin score between chains and across all variables to be at most 1.2. I use the *coda* R package (Plummer et al., 2006) to compute the effective sample size and the Geweke convergence diagnostic, and the *PyMC* python package (Patil et al., 2010) to compute the Gelman-Rubin diagnostic over two independent traces. The HMC integration may also diverge for large values of ϵ; I treat this as a hidden constraint, and set $\delta_k = 0.05$ for all constraints. I optimize HMC sampling from the posterior of a logistic regression binary classification problem using the German credit data set from the UCI repository (Frank and Asuncion, 2010). The data set contains 1000 data points, and is normalized to have unit variance. Each chain is randomly initialized with $D = 25$

independent draws from a Gaussian distribution with mean zero and standard deviation 10^{-3}.

Figure 6.6(b) compares EIC and PESC on this task, averaged over 10 independent experiments. As above, I perform a ground-truth assessment of the final recommendations. The average effective sample size is 3300 ± 1200 for PESC and 2300 ± 900 for EIC. Here, the difference between the two methods is within the margin of error. Also, in light of the ground truth, we observe that the results in Fig. 6.6(b) are overly optimistic, indicating that this experiment is very noisy. This noise presumably comes from the randomness in initialization and sampling which causes the passing or failure of the convergence diagnostics to be highly stochastic.

6.3 Analyses of PESC with decoupled constraints

6.3.1 Accuracy of the PESC approximation with decoupling

Section 6.2.1 assessed the accuracy of the PESC approximation to a ground truth computation of Eq. (4.1). Here, I assess whether the approximation is accurate for the partial acquisition functions attributed to each function in decoupled PESC. Figure 6.7 shows the PESC acquisition function compared to the RS ground truth in a synthetic $1D$ experiment with an objective and single constraint drawn from the GP prior. The left and right columns of Fig. 6.7 differ only in that three additional constraint observations have been collected in the right column (Figs. 6.7(a) and 6.7(b)). Figures 6.7(c) to 6.7(f) show that the PESC approximation is very accurate even when decomposed into the partial acquisition functions for the objective and the constraint. In the left column of Fig. 6.7, the peak of $\alpha_c(x)$ at about $x = 0.3$ is above the peak of $\alpha_f(x)$ at about $x = 0.4$, which implies that evaluating the constraint is expected to be more valuable than evaluating the objective at this iteration. Indeed, Fig. 6.7(a) shows that the objective is low near $x = 0.3$ but the constraint has not been explored there yet. Figure 6.7(g) shows that $p(\mathbf{x}_\star \,|\, \mathcal{D})$ is high near $x = 0.3$ but also high near the other peak of $\alpha_c(x)$. In the right column of Fig. 6.7, with three additional constraint observations, we now anticipate a larger information gain from evaluating the objective rather than the constraint, since the peak in Fig. 6.7(d) is above the peak in Fig. 6.7(f). Intuitively, as more constraint observations are collected, the constraint becomes well determined and the optimizer turns its attention to the objective.

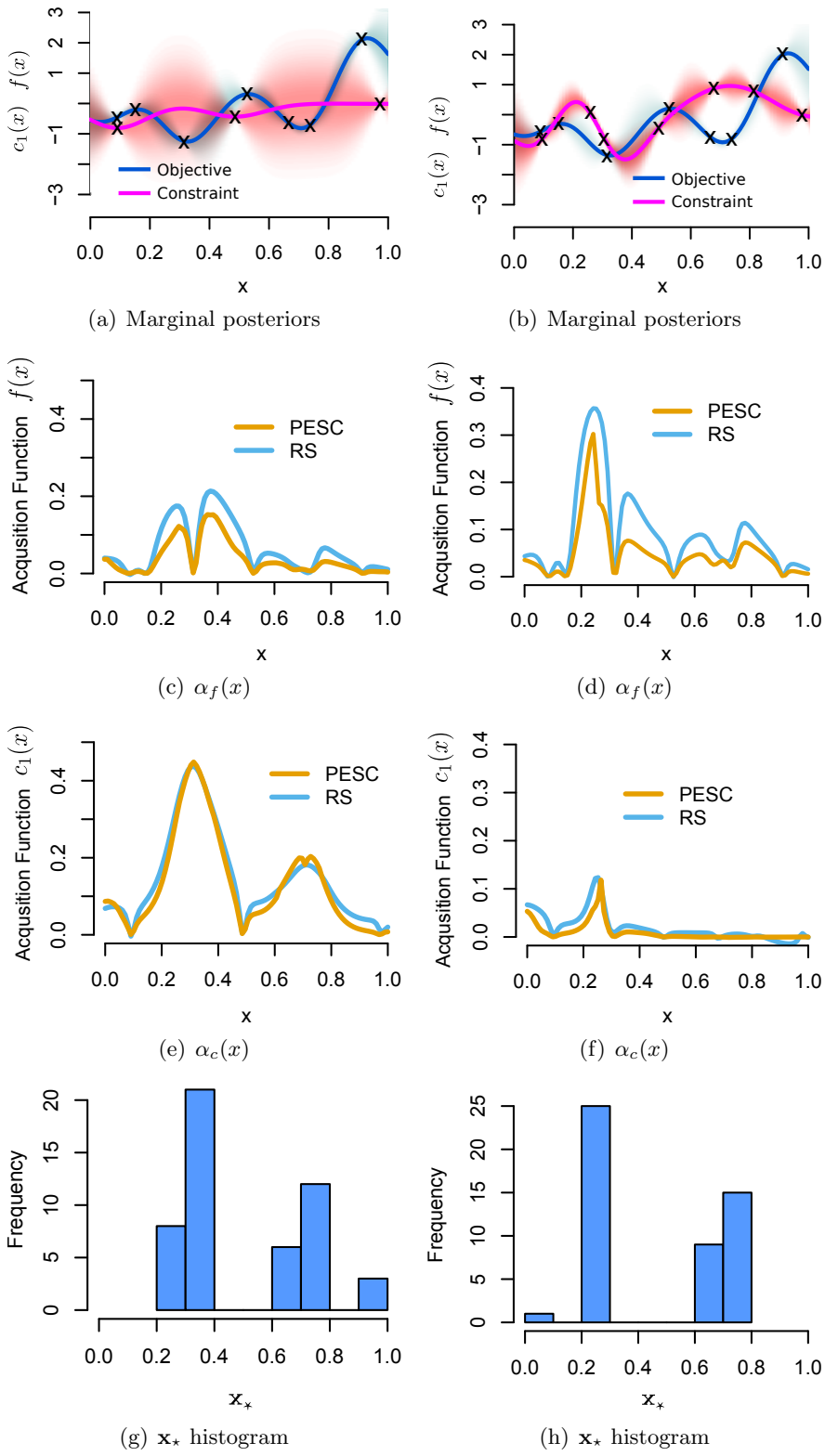
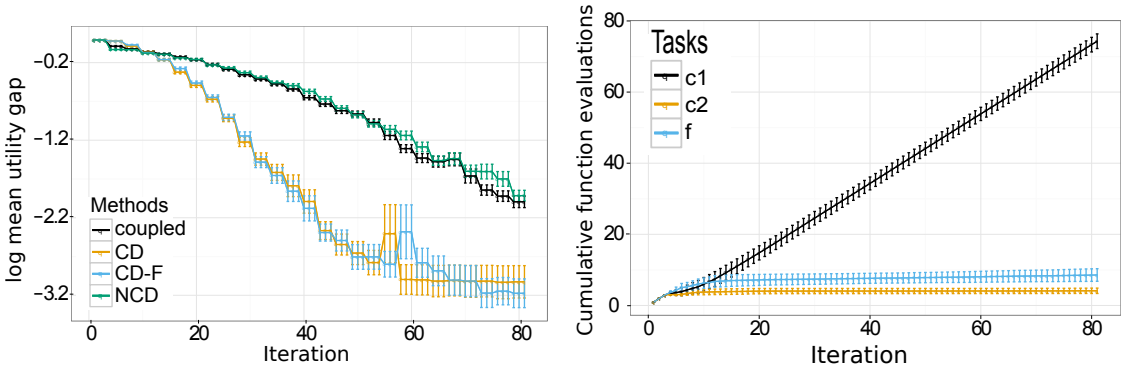


Figure 6.7: Assessing the accuracy of the decoupled PESC approximation for the partial acquisition functions for $\alpha_f(x)$ and $\alpha_c(x)$. Between the left and right columns, three additional observations of the constraint have been made.

6.3.2 Comparing coupled and decoupled PESC

In this section I compare the performance of a coupled and decoupled approach to the same (decoupled) problem, in order to empirically demonstrate the benefits of treating a decoupled problem as such. I first consider the toy problem of Section 6.2.3 with $\delta = 10^{-4}$. For the purposes of this experiment, I assume that I have a resource r with a capacity $\omega_{\max} = 3$ since there are three tasks (one objective and two constraints). Figure 6.8(a) compares the performance of four methods on this problem. The black curve is a coupled approach in which, at each iteration, all three tasks are evaluated together at the same $\mathbf{x}$. The green curve is a NCD approach in which I assume that in fact there are three resources, one per task, such that at each iteration each function is evaluated once but not necessarily at the same $\mathbf{x}$. Here and in the experiments that follow, I use the kriging believer approach (Section 2.7) to condition on pending experiments. The orange curve is a CD approach in which I allow the tasks to compete, such that at each iteration three not necessarily unique functions are evaluated at three not necessarily unique locations. Finally, the blue curve is this same approach but using PESC-F (Section 4.2). Here I do not use the decision criterion given by Eq. (4.22). Rather, when conditioning on pending experiments, fast updates are performed. Once all three new data are collected, then a full update is performed. This is an ideal circumstance for fast updates since, when performing the fast updates, the GP is only conditioning on additional "dummy" data rather than real function observations. In Section 6.4, I show that PESC-F is also effective when conditioning on actual new observations.

I now provide an interpretation of Fig. 6.8(a). The fact that NCD performs about the same as the coupled approach implies that, in this problem, the benefits of decoupling come from choosing an unequal distribution of tasks to evaluate, rather than the additional freedom of evaluating the three tasks at potentially different locations. This hypothesis is corroborated by Fig. 6.8(b), which shows that the decoupled method chose to evaluate the constraint c_1 far more often than the objective or c_2. Recalling that the objective is a linear function and only c_1 is active at the global solution, this makes sense intuitively. Finally, the fact that the blue and orange curves are very similar implies that, when only conditioning on "dummy" data, PESC-F incurs no significant performance loss.



(a) Performance (b) Cumulative task evaluations

Figure 6.8: Toy problem (Section 6.2.3) with 3 cores and 3 tasks using PESC. (a) Performance comparison of coupled (black), NCD (green), CD (orange) and fast CD (blue) approaches.

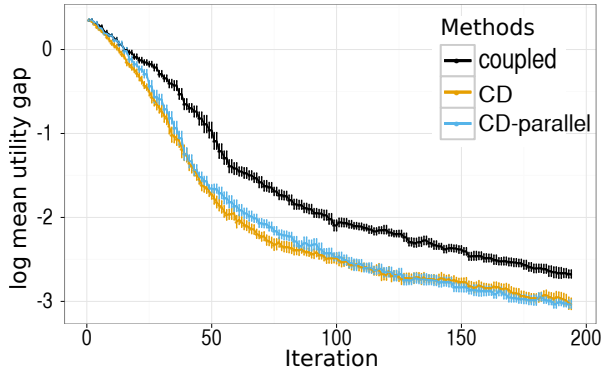
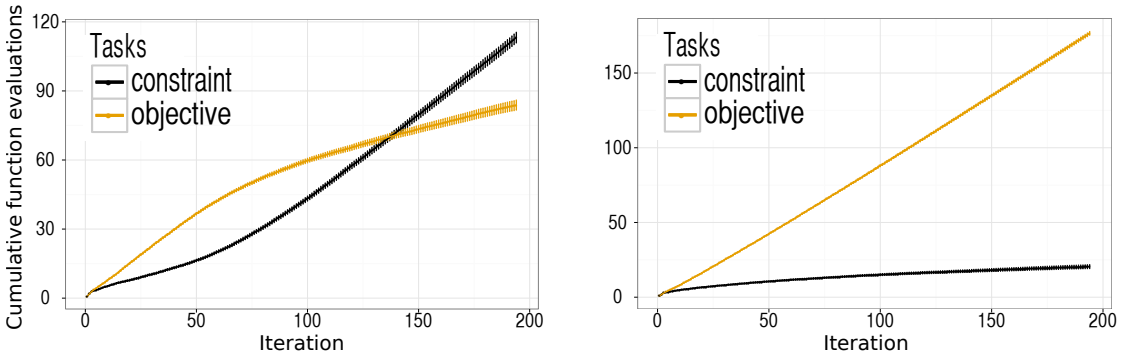


Figure 6.9: Optimizing samples from the GP in $D = 2$ with coupled (black), competitive decoupled (CD, orange) and parallel CD (blue) PESC. Curves reflect the mean over 500 realizations.



(a) Constraint active at solution (b) Constraint not active at solution

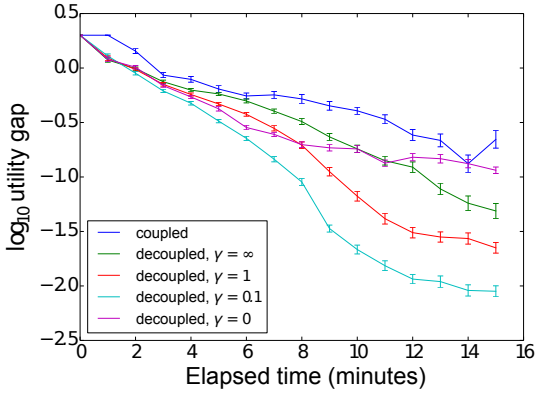
Figure 6.10: Optimizing samples from the GP prior with decoupled PESC in $D = 2$ (Fig. 6.9). Cumulative function evaluations of the objective (orange) and constraint (black) when (a) the constraint is active at the true solution (about 30% of cases) and (b) the constraint is not active at the true solution (about 70% of cases). Curves reflect the mean over 500 realizations.

I now perform another comparison of coupled and decoupled PESC with synthetic data, in which the objective and single constraint functions are drawn from the GP prior as in Section 6.2.2. Here, I assume that I have a computational resource r with capacity $\omega_{\max} = 2$. Figure 6.9 compares three approaches: coupled PESC (black), sequential PESC with competitive decoupling (orange), and parallel PESC with competitive decoupling. As in the toy problem, here the decoupled approach outperforms the coupled approach (Fig. 6.9). The sequential strategy is slightly better than the parallel strategy as it has more information on which to make decisions; however, the difference is very small. As discussed in Section 2.7, sequential methods are generally better in terms of number of function evaluations, whereas parallel methods are generally better in terms of wall-clock time.

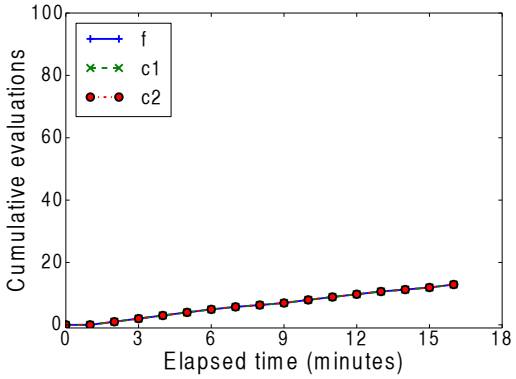
In the toy problem above, the decoupled approach was effective because it learned that evaluating the constraint c_1 is much more useful than evaluating the objective f or the constraint c_2. For this experiment with the objective and single constraint drawn from the GP prior I again plot the cumulative function evaluations of each task (Fig. 6.10). To gain understanding about the decoupled behavior, I divide the 500 realizations into those cases in which the constraint is active at the true solution (Fig. 6.10(a)) and those in which the constraint is not active at the true solution (Fig. 6.10(b)). These figures show that when the constraint is active at the solution, PESC chooses to evaluate the constraint much more frequently. In contrast, when the constraint is not active at the solution, the constraint is evaluated much less. Presumably, in these cases the constraint need only be evaluated until it is determined that it is not active at the solution; after this point, resolving finer detail is not needed. Thus, the partial acquisition functions corresponding to different tasks indeed reflect the relative importance of the tasks.

6.4 Evaluating PESC-F with wall-clock time experiments

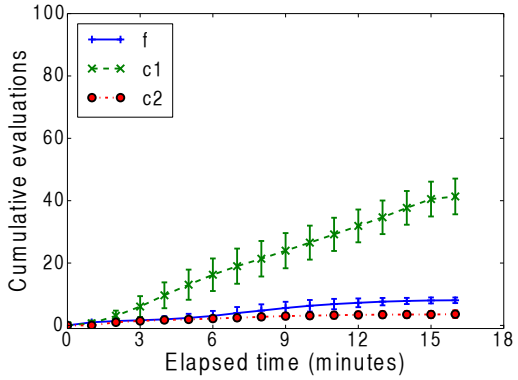
In this section I evaluate PESC with fast updates (PESC-F, Section 4.2) by comparing the performance of several PESC-based methods as a function of wall-clock time. I focus on the toy problem of Section 6.2.3 set up such that evaluating the objective is instantaneous, evaluating c_1 takes 2 seconds, and evaluating c_2 takes 1 minute. Figure 6.11(a) compares the following



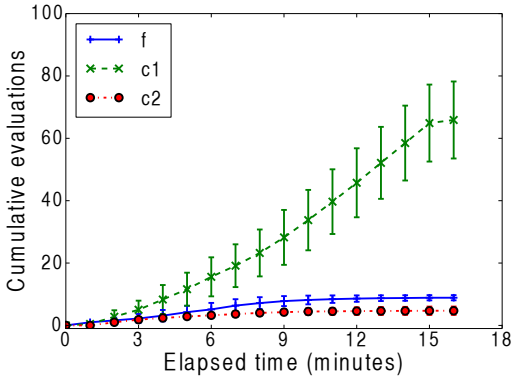
(a) Performance, all approaches



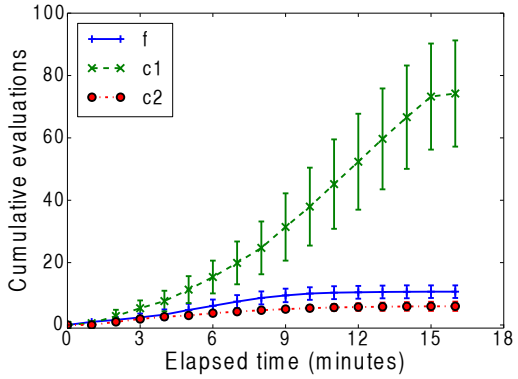
(b) Coupled



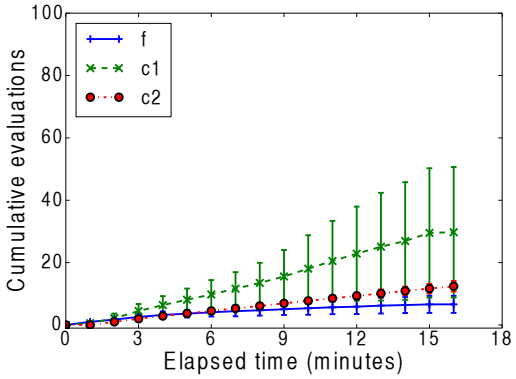
(c) Decoupled, $\gamma = \infty$



(d) Decoupled, $\gamma = 1$



(e) Decoupled, $\gamma = 0.1$



(f) Decoupled, $\gamma = 0$

Figure 6.11: Comparing five variants of PESC on the $2D$ toy problem given in Section 6.2.3, in which objective evaluations are instantaneous, c_1 evaluations take 2 sec, c_2 evaluations take 1 min, and total experiment time is 15 min. (a) log utility gap versus wall-clock time. (b-f) Cumulative function evaluations for (b) coupled, (c) decoupled with $\gamma = \infty$ (no fast updates), (d) decoupled with $\gamma = 1.0$, (e) decoupled with $\gamma = 0.1$, and (f) decoupled with $\gamma = 0$. Curves reflect the mean over 100 trials. Error bars in (b-f) are the standard deviation.

approaches in an experiment limited to 15 minutes of compute time: coupled (blue), and competitive decoupled using PESC-F with rationality levels $\gamma = \infty$ (green), 1 (red), 0.1 (cyan), and 0 (purple). Note that setting $\gamma = \infty$ is simply another way of saying that fast updates are not used (see Eq. (4.22)). Figure 6.11(a) shows that significant gains are made by using PESC-F because, intuitively, less time is spent in the optimizer and more time is spent evaluating functions. This intuition is made concrete through Figures 6.11(b) to 6.11(f) and Table 6.1. I now discuss these results in turn. Figures 6.11(b) to 6.11(f) show the cumulative number of evaluations of each task as a function of wall-clock time. In Fig. 6.11(b), the three curves overlap because the tasks are evaluated together in a coupled fashion. Because c_2 is slow, this method is very inefficient and only evaluates the three tasks 13 times in the allotted 15 minute period. This accounts for its poor performance. Figures 6.11(c) to 6.11(e) proceed as expected: the task c_1 is evaluated most because it is the most informative for this particular problem. Furthermore, as the rationality level γ is decreased, less time is spent in the optimizer, and thus more function evaluations are performed. These correspond to increases in performance in Fig. 6.11(a). However, this trend does not continue indefinitely as γ is decreased. When $\gamma = 0$, performance is significantly diminished (Fig. 6.11(a)). Setting $\gamma = 0$ corresponds to never making a full update after the initial iteration; thus, the algorithm used is, for all intents and purposes, not computing the PESC acquisition function. Interestingly, Fig. 6.11(f) shows that, even though the $\gamma = 0$ method performs only fast updated, the number of c_1 evaluations is not increased relative to approaches with larger values of γ. This is because the $\gamma = 0$ method is not able to learn that c_2 is uninformative and continues to spend time evaluating it, thus performing many fewer evaluations of c_1.

I now address Table 6.1. Whereas Figs. 6.11(b) to 6.11(f) show the number of evaluations of each task, Table 6.1 shows the time spent in these tasks and in the two optimizer modes. (Note that in Table 6.1 the total optimizer time plus the total function evaluation time does not add up to exactly 15 minutes because in this implementation the current iteration is allowed to finish after the 15-minute mark is reached.) As expected, the optimizer time decreases monotonically as γ is decreased. The coupled approach also has low optimizer time, owing simply to its only performing 13 optimization steps. We see that the fifth column of the table, corresponding to

Table 6.1: Time allocations of five PESC variants corresponding to the results in Fig. 6.11 on the $2D$ toy problem (Section 6.2.3). For each method, the table reports the mean time in minutes, over 100 independent runs, spent in standard optimizer steps (Opt-full), fast optimizer steps (Opt-fast), total optimizer steps (Opt-total), evaluating the constraint c_1, evaluating the constraint c_2, and all function evaluations (Evals-total). Time spent evaluating the objective f is not shown because evaluations of f are instantaneous ($< 10^{-3}$ min total).

Method	Opt-full	Opt-fast	**Opt-total**	$c_1(\mathbf{x})$	$c_2(\mathbf{x})$	**Evals-total**
Coupled	2.0	0.0	2.0	0.4	13.0	13.4
Decoupled, $\gamma = \infty$	10.2	0.0	10.2	1.4	3.6	5.0
Decoupled, $\gamma = 1$	5.2	3.0	8.2	2.2	4.7	6.9
Decoupled, $\gamma = 0.1$	1.5	5.1	6.6	2.5	6.0	8.5
Decoupled, $\gamma = 0.0$	0.2	1.8	2.0	1.0	12.5	13.5

time spent in c_1 and thus the number of times c_1 is evaluated, corresponds roughly to the relative performances in Fig. 6.11(a). From the results we may conclude this problem exhibits an optimal value of γ, perhaps near 0.1, and an associated optimal ratio of optimizer time to function time. I leave to future work the issue of tuning γ, but note that it in a highly sophisticated approach this could be achieved in an online fashion using reinforcement learning.

Chapter 7

Conclusion

7.1 Summary

This thesis is about Bayesian optimization with unknown constraints. Because constraint observations may be noisy, I formulated the problem using probabilistic constraints in Chapter 3, allowing the user to directly express the trade-off between cost and risk by specifying the confidence parameter δ. I then extended the constrained Expected Improvement (EIC) acquisition function by addressing the case when the incumbent η cannot be computed, and by extending EIC to the decoupled case in which the objective and constraint(s) may be evaluated independently. I then identified several shortcomings of EIC, especially in the decoupled case. To address these shortcomings, in Chapter 4 we developed Predictive Entropy Search with unknown Constraints (PESC). PESC is based on the theoretically appealing expected information gain heuristic. I showed that the mathematical approximations involved in PESC are quite accurate, and that PESC performs about equally well to a ground truth method based on rejection sampling. I also introduce a fast variant of PESC called PESC-F, and a heuristic for switching between the fast and slow variants such that the total fraction of time spent in the optimization steps (as opposed to function evaluations) is approximately equal to some user-specified constant. One disadvantage of PESC is that it is relatively difficult to implement: in particular, the EP approximation often leads to numerical instabilities. Therefore, we have integrated our implementation, which carefully addresses these numerical issues, into the open-source Bayesian optimization package

Spearmint, available at `https://github.com/HIPS/Spearmint/tree/PESC`. In Chapter 5 we formalized the notion of decoupling and created a taxonomy of decoupled problems including the cases of competitive and non-competitive decoupling. I discussed the notion of incorporating cost information into the optimization problem. Chapter 6 contains empirical evaluations of the methods discussed here. I demonstrated the effectiveness of our algorithms on the meta-optimization of machine learning algorithms and sampling methods. I also showed that PESC outperforms previously existing methods such as EIC and AL over a variety of problems. Furthermore PESC is easily applied to problems with decoupled constraints, without additional computational cost or *ad hoc* modifications.

7.2 Challenges and directions for future research

7.2.1 Scaling up to bigger data

The cubic cost incurred by the kernel matrix inversion in Eq. (2.1) makes GPs prohibitively slow once the data set becomes large. In the decoupled case with fast and slow tasks, this becomes a much more serious problem than in single-task Bayesian optimization, because the fast task may accumulate hundreds or thousands of observations before the slow task is evaluated even a handful of times. In such cases, GPs are inadequate. A highly promising recent development in this area is the work of Snoek et al. (2015), who use deep neural networks with a Bayesian linear regression at the last layer to replace GPs as the model in Bayesian optimization. Whereas the GPs exhibit cubic scaling in the number of data, the neural networks scale linearly and thus are much more tractable with large numbers of observations. One area of future work is to adapt PESC to interface with models other than GPs. Interestingly, the neural network approach not only addresses the issue of scalability, but also provides a natural way to address the bounded rationality issues that PESC-F tries to address. The neural network can be thought of as approximating a GP with a finite number of basis functions. Thus, by changing the number of hidden units in the network (or even the number of layers), one can adjust the trade-off between speed and quality of the modeling. Another interesting direction for future research is a method for performing this trade-off automatically.

7.2.2 Theoretical guarantees

Chapter 2 discusses some theoretical results regarding the convergence properties of Bayesian optimization methods. An important direction for future work is to derive results for more of these methods, such as EIC, PES, PESC, etc. While these methods are already useful from a practical standpoint, one may hesitate to use them without any theoretical guarantees.

7.2.3 Experiments with monetary or other costs

In this thesis we consider time spent to be the cost associated with evaluating the unknown function(s). An interesting extension of this notion is a more general sense of currency or cost incurred by function evaluations. This could simply be a monetary cost to an experiment. The framework of resource allocation (Chapter 5) raises further interesting questions in this direction. For example, in a computational experiment one could consider a commercial computing service such as Amazon's EC2 as a resource. The capacity of this resource would essentially be infinite, but jobs run on this resource incur a real cost in dollars. Then, an obvious question is, when should one use EC2 to perform experiments? A proper treatment of this issue, combined with Algorithm 4, could lead to a very powerful optimization method that effectively compromises time and money to find the best solution within certain time and monetary constraints. This is a direction for future research.

7.2.4 When it makes sense not to run an experiment

Consider the example of a laboratory wishing to maximize the yield of some process. In addition to performing the actual experiment, which takes one month, the laboratory also has access to a computer simulation which takes three hours to perform. The computer simulation provides an estimate of how likely the experiment is to work, although it is not particularly accurate. This is a case of non-competitive decoupling in which there are two tasks and two resources. However, Algorithm 4 is not a good method for approaching this problem. According to Algorithm 4, one begins by immediately filling the capacity of each resource; in this case, this corresponds to starting a computer simulation and starting a physical experiment. However, within a few days, dozens of the computer simulation will have been completed, providing significantly improved

information about the search space. And, yet, the laboratory experiment is already running and thus the information provided by the computer simulations will not be usable until a month has elapsed. Intuitively, a more sensible approach would be to initially not start a laboratory experiment, even though this means leaving a resource idle for some period of time. There should be some optimal number of computer simulations to run before starting the laboratory experiment, such that a balance is struck between, one the one hand, making an informed decision about the parameters of the laboratory experiment and, on the other hand not leaving a resource idle for too long. Studying this issue of when it makes sense *not* to run an experiment is an interesting direction for future research.

7.2.5 Utility functions and output warping

When we formulated the Bayesian optimization problem in Eq. (3.3), in terms of minimizing the expected value of $f(\mathbf{x})$, we implicitly assumed that the user's utility function $u(f)$ is a linear function of $-f(\mathbf{x})$. Even if the user's utility is not linear in the actual outcome of an experiment, we could rely on the user to transform the observations through $u(f)$ before passing the values to the Bayesian optimization method. However, consider the case in which a user's utility function is almost an infinite step function, as it would be with a constraint. In this case, a GP would very poorly model the observations as shown in Fig. 6.1; furthermore, if $u(f)$ is not an invertible function, then information would be lost through the transformation. Thus, there is a distinction between the output space that matches the user's utility function and the output space that is best for modeling purposes. Recent work on *input warping* (Snoek et al., 2014) automatically warps the input space to make it more amenable to GP modeling, in particular by attempting to remove non-stationarity of the objective function. The above discussion calls for research on *output warping* which, given observations in utility space or some other space, warps them to a new space that is more amenable to GP-based modeling.

7.3 Concluding remarks

Constrained black-box optimization is an exciting area of research with great practical value. It is a ubiquitous type of problem with applications in areas such as product design (e.g. designing a low-calorie cookie), machine learning meta-optimization (as in Chapter 6), real-time systems (such as a speech recognition system on a mobile device with speed, memory, and energy usage constraints), or any optimization problem in which the objective function or constraints are expensive to evaluate and possibly noisy. The application to tuning machine learning algorithms and other software is especially compelling. As research progresses further in this area, we will be able to tackle a wider variety of problems with interesting and difficult structures, in an increasingly automated manner. Perhaps in the future it will seem anomalous to have any piece of software with its parameters set by hand rather than by numerical optimization.

Appendix A

Details of the PESC Approximation

A.1 Description of Expectation Propagation

The PESC approximation is based on a Gaussian approximation to the finite-product NFCPD (Eq. (4.12)) using *Expectation Propagation* (EP) (Minka, 2001a,b). EP is a method for approximating a product of factors (typically, a prior and likelihood factors in the computation of a posterior), typically with a Gaussian distribution. EP forms a global Gaussian approximation by approximating each individual factor with an individual Gaussian. This is in contrast to the Laplace approximation which fits single, global Gaussian to the posterior distribution. EP achieves the individual Gaussian approximations by, for each factor, minimizing the Kullback-Leibler (KL) divergence between the exact factor and the Gaussian approximate factor, which corresponds to matching the first and second moments of the Gaussian to those of the exact factor. The insight behind EP is that instead doing this moment matching just between the single exact and approximate factors, we do so in the *context* of all the other factors, since we are ultimately interested in having a good approximation in regions where the overall probability is high. Concretely, assume we wish to approximate the distribution

$$q(\mathbf{x}) = \prod_{n=1}^{N} q_n(\mathbf{x})$$

with the approximate distribution

$$\tilde{q}(\mathbf{x}) = \prod_{n=1}^{N} \tilde{q}_n(\mathbf{x}) \,. \tag{A.1}$$

Consider now that we wish to form a particular approximate factor $\tilde{q}_n(\mathbf{x})$. Then, we define the *cavity* distribution $\tilde{q}^{\backslash n}(\mathbf{x})$ as:

$$\tilde{q}^{\backslash n}(\mathbf{x}) = \frac{\tilde{q}(\mathbf{x})}{\tilde{q}_n(\mathbf{x})} \,. \tag{A.2}$$

Instead of matching the moments of $q_n(\mathbf{x})$ and $\tilde{q}_n(\mathbf{x})$, in EP we instead match the moments of $q_n(\mathbf{x})\tilde{q}^{\backslash n}(\mathbf{x})$ with $\tilde{q}_n(\mathbf{x})\tilde{q}^{\backslash n}(\mathbf{x}) = \tilde{q}(\mathbf{x})$. This causes the approximation quality to be higher in places where the entire distribution $\tilde{q}(\mathbf{x})$ is high, at the expense of approximation quality in less relevant regions where $\tilde{q}(\mathbf{x})$ is close to zero.

To compute the moments of $q_n(\mathbf{x})\tilde{q}^{\backslash n}(\mathbf{x})$ we use Eqs. (5.12) and (5.13) in Minka (2001b), which give the first and second moments of a distribution $p(\mathbf{x})\mathcal{N}(\mathbf{x} \,|\, \mathbf{m}, \mathbf{V})$ in terms of the derivatives of the log partition function of this distribution. This is the desired form, since $\tilde{q}^{\backslash n}(\mathbf{x})$ is Gaussian with known mean and variance (computed in the previous iteration of EP).

Thus, given some initial approximation for all the $\{\tilde{q}_n(\mathbf{x})\}$, the steps of EP are as follows:

1. Choose an n.
2. Compute the cavity distribution $\tilde{q}^{\backslash n}(\mathbf{x})$ given by Eq. (A.2) using the formula for dividing Gaussians.
3. Compute the first and second moments of $q_n(\mathbf{x})\tilde{q}^{\backslash n}(\mathbf{x})$ using Eqs. (5.12) and (5.13) in Minka (2001b). This yields an updated Gaussian approximation $\tilde{q}(\mathbf{x})$ to $q(\mathbf{x})$ with mean and variance given by these moments.
4. Update $\tilde{q}_n(\mathbf{x})$ given by Eq. (A.2) using the formula for dividing Gaussians.
5. Repeat steps 1 to 4 until convergence.

EP updates maybe also be performed in parallel, by performing steps 1 to 4 for $n = 1, \ldots, N$ using the same $\tilde{q}(\mathbf{x})$, and then finally computing the new $\tilde{q}(\mathbf{x})$ (Eq. (A.1)) using the formula

for multiplying Gaussians. Details of the EP implementation used in this thesis are given in Appendix B.8.

A.2 The Gaussian approximation to the NFCPD

In this section we fill in the details of computing a Gaussian approximation to $q(\mathbf{f}, \mathbf{c}_1, \ldots, \mathbf{c}_K)$ given Eq. (4.12). This expression can be written as:

$$q(\mathbf{f}, \mathbf{c}_1, \ldots, \mathbf{c}_K) = \frac{1}{Z_q} \mathcal{N}\left(\mathbf{f} \,|\, \mathbf{m}^{\mathbf{f}}_{\text{pred}}, \mathbf{V}^{\mathbf{f}}_{\text{pred}}\right) \left[\prod_{k=1}^{K} \mathcal{N}\left(\mathbf{c}_k \,|\, \mathbf{m}^{\mathbf{c}_k}_{\text{pred}}, \mathbf{V}^{\mathbf{c}_k}_{\text{pred}}\right)\right] \times \left[\prod_{k=1}^{K} \Theta[c_{k,N+1}]\right] \prod_{n=1}^{N} \left[\left\{\prod_{k=1}^{K} \Theta[c_{k,n}]\right\} \Theta[f_n - f_{N+1}] + \left\{1 - \prod_{k=1}^{K} \Theta[c_{k,n}]\right\}\right], \quad \text{(A.3)}$$

where $\mathbf{m}^{f}_{\text{pred}}$ and $\mathbf{V}^{f}_{\text{pred}}$ are the mean and covariance matrix of the posterior distribution of $\mathbf{f}$ given the data in $\mathcal{D}^f$, and $\mathbf{m}^{\mathbf{c}_k}_{\text{pred}}$ and $\mathbf{V}^{\mathbf{c}_k}_{\text{pred}}$ are the mean and covariance matrix of the posterior distribution of $\mathbf{c}_k$ given the data in $\mathcal{D}^k$. In particular, from Eq. (2.1) we have that

$$\begin{aligned} \mathbf{m}^{\mathbf{f}}_{\text{pred}} &= \mathbf{K}^{f}_{\star}(\mathbf{K}^{f} + \nu_f^2 \mathbb{I})^{-1} \mathbf{y}^{f}, \\ \mathbf{V}^{\mathbf{f}}_{\text{pred}} &= \mathbf{K}^{f}_{\star,\star} - \mathbf{K}^{f}_{\star}(\mathbf{K}^{f} + \nu_f^2 \mathbb{I})^{-1} [\mathbf{K}^{f}_{\star}]^{\top}, \end{aligned}$$

where $\mathbf{K}^{f}_{\star}$ is an $(N+1) \times N$ matrix with the prior cross-covariances between elements of $\mathbf{f}$ and $f_1, \ldots, f_n$ and $\mathbf{K}^{f}_{\star,\star}$ is an $(N+1) \times (N+1)$ matrix with the prior covariances between elements of $\mathbf{f}$. Similarly, we have that

$$\begin{aligned} \mathbf{m}^{\mathbf{c}_k}_{\text{pred}} &= \mathbf{K}^{k}_{\star}(\mathbf{K}^{k} + \nu_k^2 \mathbb{I})^{-1} \mathbf{y}^{k}, \\ \mathbf{V}^{\mathbf{c}_k}_{\text{pred}} &= \mathbf{K}^{k}_{\star,\star} - \mathbf{K}^{k}_{\star}(\mathbf{K}^{k} + \nu_k^2 \mathbb{I})^{-1} [\mathbf{K}^{k}_{\star}]^{\top}, \end{aligned}$$

where $\mathbf{K}^{k}_{\star}$ is an $(N+1) \times N$ matrix with the prior cross-covariances between elements of $\mathbf{c}_k$ and $c_{k,1}, \ldots, c_{k,n}$ and $\mathbf{K}^{k}_{\star,\star}$ is an $(N+1) \times (N+1)$ matrix containing the prior covariances between

the elements of $\mathbf{c}_k$. We will refer to the non-Gaussian factors in Eq. (A.3) as

$$h_n(f_n, f_{N+1}, c_{1,n}, \ldots, c_{k,n}) = \left\{ \prod_{k=1}^{K} \Theta\left[c_{k,n}\right] \right\} \Theta\left[f_n - f_{N+1}\right] + \left\{ 1 - \prod_{k=1}^{K} \Theta\left[c_{k,n}\right] \right\} \tag{A.4}$$

(this is called $\Psi(\mathbf{x}_n, \mathbf{x}_\star, \mathbf{f}, \mathbf{c}_1, \ldots, \mathbf{c}_K)$ in Chapter 4) and

$$g_k(c_{k,N+1}) = \Theta[c_{k,N+1}]\,, \tag{A.5}$$

such that Eq. (A.3) can be written as

$$\begin{aligned} q(\mathbf{f}, \mathbf{c}_1, \ldots, \mathbf{c}_K) \propto \mathcal{N}(\mathbf{f} \,|\, \mathbf{m}^{\mathbf{f}}_{\text{pred}}, \mathbf{V}^{\mathbf{f}}_{\text{pred}}) \left[\prod_{k=1}^{K} \mathcal{N}(\mathbf{c}_k \,|\, \mathbf{m}^{\mathbf{c}_k}_{\text{pred}}, \mathbf{V}^{\mathbf{c}_k}_{\text{pred}}) \right] \\ \left[\prod_{n=1}^{N} h_n(f_n, f_{N+1}, c_{1,n}, \ldots, c_{k,n}) \right] \left[\prod_{k=1}^{K} g_k(c_{k,N+1}) \right]. \end{aligned} \tag{A.6}$$

We approximate these exact factors $h_n(f_n, f_{N+1}, c_{1,n}, \ldots, c_{k,n})$ and $g_k(c_{k,N+1})$ in Eqs. (A.4) and (A.5) with Gaussian approximate factors $\tilde{h}_n(f_n, f_{N+1}, c_{1,n}, \ldots, c_{k,n})$ and $\tilde{g}_k(c_{k,N+1})$ respectively. By the assumed independence of the objective and constraints, these approximate factors take the form

$$\begin{aligned} \tilde{h}_n(f_n, f_{N+1}, c_{1,n}, \ldots, c_{k,n}) \propto \exp\left\{ -\frac{1}{2} [f_n \, f_{N+1}] \mathbf{A}_{h_n} [f_n \, f_{N+1}]^\top + [f_n \, f_{N+1}] \mathbf{b}_{h_n} \right\} \\ \prod_{k=1}^{K} \exp\left\{ -\frac{1}{2} a_{h_n} c_{k,n}^2 + b_{h_n} c_{k,n} \right\} \end{aligned} \tag{A.7}$$

and

$$\tilde{g}_k(c_{k,N+1}) \propto \exp\left\{ -\frac{1}{2} a_{g_k} c_{k,N+1}^2 + b_{g_k} c_{k,N+1} \right\}, \tag{A.8}$$

where $\mathbf{A}_{h_n}, \mathbf{b}_{h_n}, a_{h_n}, b_{h_n}, a_{g_k}$ and b_{g_k} are the natural parameters of the respective Gaussian distributions.

The right-hand side of Eq. (A.6) is then approximated by

$$\tilde{q}(\mathbf{f}, \mathbf{c}_1, \ldots, \mathbf{c}_K) \propto \mathcal{N}\left(\mathbf{f} \,|\, \mathbf{m}^f_{\text{pred}}, \mathbf{V}^f_{\text{pred}}\right) \left[\prod_{k=1}^{K} \mathcal{N}\left(\mathbf{c}_k \,|\, \mathbf{m}^{\mathbf{c}_k}_{\text{pred}}, \mathbf{V}^{\mathbf{c}_k}_{\text{pred}}\right)\right] \left[\prod_{n=1}^{N} \tilde{h}_n(f_n, f_{N+1}, c_{1,n}, \ldots, c_{k,n})\right] \left[\prod_{k=1}^{K} \tilde{g}_k(c_{k,N+1})\right] . \tag{A.9}$$

Because the $\tilde{h}_n(f_n, f_{N+1}, c_{1,n}, \ldots, c_{k,n})$ and $\tilde{g}_k(c_{k,N+1})$ are Gaussian, they can be combined with the Gaussian terms in the first line of Eq. (A.9) and written as

$$\tilde{q}(\mathbf{f}, \mathbf{c}_1, \ldots, \mathbf{c}_K) \propto \mathcal{N}(\mathbf{f} \,|\, \mathbf{m}^{\mathbf{f}}, \mathbf{V}^{\mathbf{f}}) \left[\prod_{k=1}^{K} \mathcal{N}(\mathbf{c}_k \,|\, \mathbf{m}^{\mathbf{c}_k}, \mathbf{V}^{\mathbf{c}_k})\right] , \tag{A.10}$$

where, applying the formula for products of Gaussians, we obtain

$$\begin{aligned} \mathbf{V}^{\mathbf{f}} &= \left[\left(\mathbf{V}^{\mathbf{f}}_{\text{pred}}\right)^{-1} + \widetilde{\mathbf{V}}^{\mathbf{f}}\right]^{-1}, \qquad & \mathbf{m}^{\mathbf{f}} &= \mathbf{V}^{\mathbf{f}} \left[\left(\mathbf{V}^{\mathbf{f}}_{\text{pred}}\right)^{-1} \mathbf{m}^{\mathbf{f}}_{\text{pred}} + \widetilde{\mathbf{m}}^{\mathbf{f}}\right] , \\ \mathbf{V}^{\mathbf{c}_k} &= \left[\left(\mathbf{V}^{\mathbf{c}_k}_{\text{pred}}\right)^{-1} + \widetilde{\mathbf{V}}^{\mathbf{c}_k}\right]^{-1}, \qquad & \mathbf{m}^{\mathbf{c}_k} &= \mathbf{V}^{\mathbf{c}_k} \left[\left(\mathbf{V}^{\mathbf{c}_k}_{\text{pred}}\right)^{-1} \mathbf{m}^{\mathbf{c}_k}_{\text{pred}} + \widetilde{\mathbf{m}}^{\mathbf{c}_k}\right] \end{aligned} \tag{A.11}$$

and with the following definitions for $\widetilde{\mathbf{V}}, \widetilde{\mathbf{m}}^{\mathbf{f}}, \widetilde{\mathbf{V}}^{\mathbf{c}_k}$, and $\widetilde{\mathbf{m}}^{\mathbf{c}_k}$:

- $\widetilde{\mathbf{V}}^{\mathbf{f}}$ is an $(N+1) \times (N+1)$ precision matrix with entries given by
  - $\tilde{v}^{\mathbf{f}}_{n,n} = [\mathbf{A}_{h_n}]_{1,1}$ for $n = 1, \ldots, N$,
  - $\tilde{v}^{\mathbf{f}}_{N+1,n} = \tilde{v}^{\mathbf{f}}_{n,N+1} = [\mathbf{A}_{h_n}]_{1,2}$ for $n = 1, \ldots, N$,
  - $\tilde{v}^{\mathbf{f}}_{N+1,N+1} = \sum_{n=1}^{N} [\mathbf{A}_{h_n}]_{2,2}$,
  - and all other entries are zero.
- $\widetilde{\mathbf{m}}^{\mathbf{f}}$ is an $(N+1)$-dimensional vector with entries given by
  - $\widetilde{m}^{\mathbf{f}}_n = [\mathbf{b}_{h_n}]_1$ for $n = 1, \ldots, N$,
  - $\widetilde{m}^{\mathbf{f}}_{N+1} = \sum_{n=1}^{N} [\mathbf{b}_{h_n}]_2$.
- $\widetilde{\mathbf{V}}^{\mathbf{c}_k}$ is a $(N+1) \times (N+1)$ diagonal precision matrix with entries given by
  - $\tilde{v}^{\mathbf{c}_k}_{n,n} = a_{h_n}$ for $n = 1, \ldots, N$,

– $\tilde{v}^{\mathbf{c}_k}_{N+1,N+1} = a_{g_k}$.

- $\tilde{\mathbf{m}}^{\mathbf{c}_k}$ is an $(N+1)$-dimensional vector such that

  – $\widetilde{m}^{\mathbf{c}_k}_n = b_{h_n}$ for $n = 1, \ldots, N$,

  – $\widetilde{m}^{\mathbf{c}_k}_{N+1} = b_{g_k}$.

In the next section we explain how to actually obtain the values of $\mathbf{A}_{h_n}, \mathbf{b}_{h_n}, a_{h_n}, b_{h_n}, a_{g_k}$ and b_{g_k} by running EP.

A.3 The EP approximation to h_n and g_k

In this section we explain how to iteratively refine the approximate Gaussian factors $\tilde{h}_n$ and $\tilde{g}_k$ with EP.

A.3.1 EP update operations for $\tilde{h}_n$

EP adjusts each $\tilde{h}_n$ by minimizing the KL divergence between

$$h_n(f_n, f_{N+1}, c_{1,n}, \ldots, c_{k,n})\tilde{q}^{\backslash n}(\mathbf{f}, \mathbf{c}_1, \ldots, \mathbf{c}_K) \tag{A.12}$$

and

$$\tilde{h}_n(f_n, f_{N+1}, c_{1,n}, \ldots, c_{k,n})\tilde{q}^{\backslash n}(\mathbf{f}, \mathbf{c}_1, \ldots, \mathbf{c}_K)\,, \tag{A.13}$$

where we define the cavity distribution $\tilde{q}^{\backslash n}(\mathbf{f}, \mathbf{c}_1, \ldots, \mathbf{c}_K)$ as

$$\tilde{q}^{\backslash n}(\mathbf{f}, \mathbf{c}_1, \ldots, \mathbf{c}_K) = \frac{\tilde{q}(\mathbf{f}, \mathbf{c}_1, \ldots, \mathbf{c}_K)}{\tilde{h}_n(f_n, f_{N+1}, c_{1,n}, \ldots, c_{k,n})}\,.$$

Because $\tilde{q}(\mathbf{f}, \mathbf{c}_1, \ldots, \mathbf{c}_K)$ and $\tilde{h}_n(f_n, f_{N+1}, c_{1,n}, \ldots, c_{k,n})$ are both Gaussian, $\tilde{q}^{\backslash n}(\mathbf{f}, \mathbf{c}_1, \ldots, \mathbf{c}_K)$ is also Gaussian. If we marginalize out all variables except those which $\tilde{h}_n$ depends on, namely

$f_n, f_{N+1}, c_{1,n}, \ldots, c_{k,n}$, then $\tilde{q}^{\setminus n}(f_n, f_{N+1}, c_{1,n}, \ldots, c_{k,n})$ takes the form

$$\tilde{q}^{\setminus n}(f_n, f_{N+1}, c_{1,n}, \ldots, c_{k,n}) \propto \mathcal{N}\left([f_n f_{N+1}] \,|\, \mathbf{m}_{n,\text{old}}^{f_n,f_{N+1}}, \mathbf{V}_{n,\text{old}}^{f_n,f_{N+1}}\right) \times \left[\prod_{k=1}^{K} \mathcal{N}\left(c_{k,n} \,|\, m_{n,\text{old}}^{c_{k,n}}, v_{n,\text{old}}^{c_{k,n}}\right)\right], \tag{A.14}$$

where, by applying the forumla for dividing Gaussians, we obtain

$$\mathbf{V}_{n,\text{old}}^{f_n,f_{N+1}} = \left\{\left[\mathbf{V}_{f_n,f_{N+1}}^{\mathbf{f}}\right]^{-1} - \mathbf{A}_{h_n}\right\}^{-1}, \tag{A.15}$$

$$\mathbf{m}_{n,\text{old}}^{f_n,f_{N+1}} = \mathbf{V}_{n,\text{old}}^{f_n,f_{N+1}} \left\{\left[\mathbf{V}_{f_n,f_{N+1}}^{\mathbf{f}}\right]^{-1} \mathbf{m}_{f_n,f_{N+1}}^{\mathbf{f}} - \mathbf{b}_{h_n}\right\}, \tag{A.16}$$

$$v_{n,\text{old}}^{c_{k,n}} = \left\{\left[v_{n,n}^{\mathbf{c}_k}\right]^{-1} - a_{c_{k,n}}^{h_n}\right\}^{-1}, \tag{A.17}$$

$$m_{n,\text{old}}^{c_{k,n}} = \left\{m_n^{\mathbf{c}_k}\left[v_{n,n}^{\mathbf{c}_k}\right]^{-1} - b_{h_n}\right\}^{-1}, \tag{A.18}$$

where $\mathbf{V}_{f_n,f_{N+1}}^{\mathbf{f}}$ is the 2×2 covariance matrix obtained by taking the entries corresponding to f_n and f_{N+1} from $\mathbf{V}^{\mathbf{f}}$, and $\mathbf{m}_{f_n,f_{N+1}}^{\mathbf{f}}$ is the corresponding 2-dimensional mean vector. Similarly, $v_{n,n}^{\mathbf{c}_k}$ is the variance for $c_{k,n}$ in $\tilde{q}$ and $m_n^{\mathbf{c}_k}$ is the corresponding mean.

To minimize the KL divergence between Eqs. (A.12) and (A.13), we match the first and second moments of these two distributions. The moments of Eq. (A.12) can be obtained from the derivatives of its normalization constant. This normalization constant is given by

$$\begin{aligned} Z &= \int h_n(f_n, f_{N+1}, c_{1,n}, \ldots, c_{k,n}) \tilde{q}^{\setminus n}(f_n, f_{N+1}, c_{1,n}, \ldots, c_{k,n})\, df_n, df_{N+1}, dc_{1,n}, \ldots, dc_{k,n} \\ &= \left(\left\{\prod_{k=1}^{K} \Phi\left[\alpha_n^k\right]\right\} \Phi(\alpha_n) + \left\{1 - \prod_{k=1}^{K} \Phi\left[\alpha_n^k\right]\right\}\right), \end{aligned} \tag{A.19}$$

where $\alpha_n^k = m_{n,\text{old}}^{c_{k,n}} / \sqrt{v_{n,\text{old}}^{c_{k,n}}}$ and $\alpha_n = [1, -1]\mathbf{m}_{n,\text{old}}^{f_n,f_{N+1}} / \sqrt{[1, -1]\mathbf{V}_{n,\text{old}}^{f_n,f_{N+1}}[1, -1]^\top}$. We follow Eq. (5.13) Minka (2001b) to update a_{h_n} and b_{h_n}; however, we use the second partial derivatives with respect to $m_{n,\text{old}}^{c_{k,n}}$ rather than first partial derivative with respect to $v_{n,\text{old}}^{c_{k,n}}$ for numerical

robustness. These derivatives are given by

$$\frac{\partial \log Z}{\partial m_{n,\text{old}}^{c_{k,n}}} = \frac{(Z-1)\phi(\alpha_n^k)}{Z\Phi(\alpha_n^k)\sqrt{v_{n,\text{old}}^{c_{k,n}}}}\,, \tag{A.20}$$

$$\frac{\partial^2 \log Z}{\partial [m_{n,\text{old}}^{c_{k,n}}]^2} = -\frac{(Z-1)\phi(\alpha_n^k)}{Z\Phi(\alpha_n^k)} \cdot \frac{\left[\alpha_n^k + \frac{(Z-1)\phi(\alpha_n^k)}{Z\Phi(\alpha_n^k)}\right]}{v_{l,\text{old}}^{c_n^n}}\,. \tag{A.21}$$

The update equations for the parameters a_{h_n} and b_{h_n} of the approximate factor $\tilde{h}$ are then

$$\begin{aligned} a_{h_n}^{\text{new}} &= -\left(\left(\frac{\partial^2 \log Z}{\partial [m_{n,\text{old}}^{c_{k,n}}]^2}\right)^{-1} + v_{n,\text{old}}^{c_{k,n}}\right)^{-1}\,, \\ b_{h_n}^{\text{new}} &= \left\{ m_{n,\text{old}}^{c_{k,n}} - \left[\frac{\partial^2 \log Z}{\partial [m_{n,\text{old}}^{c_{k,n}}]^2}\right]^{-1} \frac{\partial \log Z}{\partial m_{n,\text{old}}^{c_{k,n}}} \right\} a_{h_n}^{\text{new}} \end{aligned} \tag{A.22}$$

We now perform the analogous operations to update $\mathbf{A}_{h_n}$ and $\mathbf{b}_{h_n}$. We need to compute

$$\begin{aligned} \frac{\partial \log Z}{\partial \mathbf{m}_{n,\text{old}}^{f_n,f_{N+1}}} &= \frac{\left\{\prod_{k=1}^K \Phi[\alpha_n^k]\right\}\phi(\alpha_n)}{Z\sqrt{s}}[1,\,-1]\,, \\ \frac{\partial \log Z}{\partial \mathbf{V}_{n,\text{old}}^{f_n,f_{N+1}}} &= -\frac{1}{2}[1,\,-1]^\top[1,\,-1]\frac{\left\{\prod_{k=1}^K \Phi[\alpha_n^k]\right\}\phi(\alpha_n)\alpha_n}{Zs}\,, \end{aligned}$$

where

$$s = [-1\,1]\mathbf{V}_{n,\text{old}}^{f_n,f_{N+1}}[-1\,1]^\top\,. \tag{A.23}$$

We then compute the optimal mean and variance (those that minimize the KL-divergence) of the product in Eq. (A.13):

$$\begin{aligned} [\mathbf{V}_{f_n,f_{N+1}}^{\mathbf{f}}]_{\text{new}} &= \mathbf{V}_{n,\text{old}}^{f_n,f_{N+1}} - \mathbf{V}_{n,\text{old}}^{f_n,f_{N+1}}\left[\frac{\partial \log Z}{\partial \mathbf{m}_{n,\text{old}}^{f_n,f_{N+1}}}\left(\frac{\partial \log Z}{\partial \mathbf{m}_{n,\text{old}}^{f_n,f_{N+1}}}\right)^\top - 2\frac{\partial \log Z}{\partial \mathbf{V}_{n,\text{old}}^{f_n,f_{N+1}}}\right]\mathbf{V}_{n,\text{old}}^{f_n,f_{N+1}}\,, \\ [\mathbf{m}_{f_n,f_{N+1}}^{\mathbf{f}}]_{\text{new}} &= \mathbf{m}_{n,\text{old}}^{f_n,f_{N+1}} + \mathbf{V}_{n,\text{old}}^{f_n,f_{N+1}}\frac{\partial \log Z}{\partial \mathbf{m}_{n,\text{old}}^{f_n,f_{N+1}}}\,. \end{aligned} \tag{A.24}$$

Next we need to divide the Gaussian distribution in Eq. (A.24) by the Gaussian cavity distribu-

tion $\tilde{q}^{\backslash n}(f_n, f_{N+1}, c_{1,n}, \ldots, c_{k,n})$. Therefore the new parameters $\mathbf{A}_{h_n}$ and $\mathbf{b}_{h_n}$ of the approximate factor $\tilde{h}$ are obtained using the formula for the ratio of two Gaussians:

$$
\begin{aligned}
[\mathbf{A}_{h_n}]^{\text{new}} &= \left[\mathbf{V}^{\mathbf{f}}_{f_n,f_{N+1}}\right]^{-1}_{\text{new}} - \left[\mathbf{V}^{f_n,f_{N+1}}_{n,\text{old}}\right]^{-1} . \\
[\mathbf{b}_{h_n}]^{\text{new}} &= \left[\mathbf{m}^{\mathbf{f}}_{f_n,f_{N+1}}\right]_{\text{new}} \left[\mathbf{V}^{\mathbf{f}}_{f_n,f_{N+1}}\right]^{-1}_{\text{new}} - \mathbf{m}^{f_n,f_{N+1}}_{n,\text{old}} \left[\mathbf{V}^{f_n,f_{N+1}}_{n,\text{old}}\right]^{-1} .
\end{aligned} \tag{A.25}
$$

A.3.2 EP approximation to g_k

We now show how to refine the $\{\tilde{g}_k\}$. We adjust each $\tilde{g}_k$ by minimizing the KL divergence between

$$
g_k(c_{k,N+1})\tilde{q}^{\backslash k}(\mathbf{f}, \mathbf{c}_1, \ldots, \mathbf{c}_K) \tag{A.26}
$$

and

$$
\tilde{g}_k(c_{k,N+1})\tilde{q}^{\backslash k}(\mathbf{f}, \mathbf{c}_1, \ldots, \mathbf{c}_K) , \tag{A.27}
$$

where

$$
\tilde{q}^{\backslash k}(\mathbf{f}, \mathbf{c}_1, \ldots, \mathbf{c}_K) = \frac{\tilde{q}(\mathbf{f}, \mathbf{c}_1, \ldots, \mathbf{c}_K)}{\tilde{g}_k(c_{k,N+1})} .
$$

Because $\tilde{q}(\mathbf{f}, \mathbf{c}_1, \ldots, \mathbf{c}_K)$ and $\tilde{g}_k(c_{k,N+1})$ are both Gaussian, $\tilde{q}^{\backslash k}(\mathbf{f}, \mathbf{c}_1, \ldots, \mathbf{c}_K)$ is also Gaussian. If we marginalize out all variables except those which $\tilde{g}_k$ depends on, namely $c_{k,N+1}$, then $\tilde{q}^{\backslash k}(\mathbf{f}, \mathbf{c}_1, \ldots, \mathbf{c}_K)$ takes the form $\tilde{q}^{\backslash k}(c_{k,N+1}) = \mathcal{N}\left(c_{k,N+1} \,|\, m^{c_{k,n}}_{k,\text{old}}, v^{c_{k,n}}_{k,\text{old}}\right)$, where, by appling the formula for the ratio of Gaussians, $m^{c_{k,n}}_{k,\text{old}}$ and $v^{c_{k,n}}_{k,\text{old}}$ are given by

$$
\begin{aligned}
v^{c_{k,n}}_{k,\text{old}} &= \left\{[v^{\mathbf{c}_k}_{n,n}]^{-1} - a_{g_k}\right\}^{-1} , \\
m^{c_{k,n}}_{k,\text{old}} &= \left\{m^{\mathbf{c}^k}_n [v^{\mathbf{c}^k}_{n,n}]^{-1} - b_{g_k}\right\}^{-1} .
\end{aligned} \tag{A.28}
$$

As in Appendix A.3.1, to minimize the KL divergence between Eqs. (A.26) and (A.27), we match the first and second moments of these two distributions. The moments of Eq. (A.26) can be obtained from the derivatives of its normalization constant. This normalization constant is

given by $Z = \Phi(\alpha)$ where $\alpha = m_{k,\text{old}}^{c_{k,n}} / \sqrt{v_{k,\text{old}}^{c_{k,n}}}$. Then, the derivatives are

$$\begin{aligned}
\frac{\partial \log Z}{\partial m_{k,\text{old}}^{c_{k,n}}} &= \frac{\phi[\alpha]}{\Phi[\alpha]\sqrt{v_{k,\text{old}}^{c_{k,n}}}} \\
\frac{\partial^2 \log Z}{\partial [m_{k,\text{old}}^{c_{k,n}}]^2} &= -\frac{\phi[\alpha]}{v_{k,\text{old}}^{c_{k,n}}\Phi[\alpha]} \cdot \left\{ \alpha + \frac{\phi[\alpha]}{\Phi[\alpha]} \right\}
\end{aligned} \tag{A.29}$$

The update rules for a_{g_k} and b_{g_k} are then

$$\begin{aligned}
[a_{g_k}]^{\text{new}} &= -\left(\left[\frac{\partial \log Z}{\partial m_{k,\text{old}}^{c_{k,n}}} \right]^{-1} + v_{k,\text{old}}^{c_{k,n}} \right)^{-1} \\
[b_{g_k}]^{\text{new}} &= \left\{ m_{k,\text{old}}^{c_{k,n}} - \left(\frac{\partial^2 \log Z}{\partial [m_{k,\text{old}}^{c_{k,n}}]^2} \right)^{-1} \frac{\partial \log Z}{\partial m_{k,\text{old}}^{c_{k,n}}} \right\} [a_{g_k}]^{\text{new}} \, .
\end{aligned} \tag{A.30}$$

Once EP has converged we can approximate the NFCPD as

$$\begin{aligned}
&p\left(\mathbf{z} \,|\, \mathcal{D}, \mathbf{x}, \mathbf{x}_\star\right) \approx \int p(z_0 \,|\, \mathbf{f}) p(z_1 \,|\, \mathbf{c}_1) \cdots p(z_K \,|\, \mathbf{c}_K)\, \tilde{q}(\mathbf{f}, \mathbf{c}_1, \ldots, \mathbf{c}_K) \\
&\left(\left\{ \prod_{k=1}^{K} \Theta[c_k(\mathbf{x})] \right\} \Theta[f(\mathbf{x}) - f_{N+1}] + \left\{ 1 - \prod_{k=1}^{K} \Theta[c_k(\mathbf{x})] \right\} \right) d\mathbf{f}\, d\mathbf{c}_1 \cdots d\mathbf{c}_K,
\end{aligned} \tag{A.31}$$

where we have used the assumed independence of the objective and constraints to split the distribution $p(\mathbf{z} \,|\, \mathbf{f}, \mathbf{c}_1, \ldots, \mathbf{c}_K)$ into the product of factors $p(f(\mathbf{x}) \,|\, \mathbf{f})$, $p(c_1(\mathbf{x}) \,|\, \mathbf{c}_1), \ldots, p(c_K(\mathbf{x}) \,|\, \mathbf{c}_K)$. This is Eq. (4.16).

A.4 Performing the integration in Eq. (A.31)

Because the constraint factor on the right-hand side of Eq. (A.31) only depends on f_{N+1} out of all the integration variables, we can rewrite Eq. (A.31) as

$$\begin{aligned}
&p\left(\mathbf{z} \,|\, \mathcal{D}^f, \mathcal{D}^1, \ldots, \mathcal{D}^K, \mathbf{x}, \mathbf{x}_\star^{(m)}\right) \approx \frac{1}{Z} \int \left(\left\{ \prod_{k=1}^{K} \Theta[z_k] \right\} \Theta[z_0 - f_{N+1}] + \left\{ 1 - \prod_{k=1}^{K} \Theta[z_k] \right\} \right) \\
&\left(\int p(z_0 \,|\, \mathbf{f})\, p(z_1 \,|\, \mathbf{c}_1) \cdots p(z_K \,|\, \mathbf{c}_K)\, \tilde{q}(\mathbf{f}, \mathbf{c}_1, \ldots, \mathbf{c}_K)\, df_1 \cdots df_N\, d\mathbf{c}_1 \cdots d\mathbf{c}_K \right) df_{N+1} \, ,
\end{aligned} \tag{A.32}$$

where Z is a normalization constant.

We now compute the inner integral in A.32 analytically. This is possible because the marginal posterior predictive distributions on $\mathbf{z}$ are Gaussian and we also have a Gaussian approximation $\tilde{q}(\mathbf{f}, \mathbf{c}_1, \ldots, \mathbf{c}_K)$. We first rewrite the inner integral in Eq. (A.32) by substituting the definition of $\tilde{q}$ in Eq. (4.15):

$$\int p(z_0 \,|\, \mathbf{f})\, p(z_1 \,|\, \mathbf{c}_1) \cdots p(z_K \,|\, \mathbf{c}_K) \mathcal{N}(\mathbf{f} \,|\, \mathbf{m}_0, \mathbf{V}_0) \prod_{k=1}^{K} \mathcal{N}(\mathbf{c}_k \,|\, \mathbf{m}_k, \mathbf{V}_k) df_1 \cdots df_N \, d\mathbf{c}_1 \cdots d\mathbf{c}_K \tag{A.33}$$

For each $\mathbf{c}_k$, the product of the conditional Gaussian distribution $p(c_k(\mathbf{x}) \,|\, \mathbf{c}_k)$ with the Gaussian $\mathcal{N}(\mathbf{c}_k \,|\, \mathbf{m}_k, \mathbf{V}_k)$ is an $(N+2)$-dimensional multivariate Gaussian distribution. All variables in $\mathbf{c}_k$ are then integrated out leaving only a univariate Gaussian on $c_k(\mathbf{x})$ with mean and variance m'_k and v'_k respectively. For the variables $f(\mathbf{x})$ and $\mathbf{f}$, the product of $p(f(\mathbf{x}) \,|\, \mathbf{f})$ and $\mathcal{N}(\mathbf{f} \,|\, \mathbf{m}_0, \mathbf{V}_0)$ yields an $(N+2)$-dimensional multivariate Gaussian distribution. The variables $f_1, \ldots, f_N$ are integrated out leaving only a bivariate Gaussian on the two-dimensional vector $\mathbf{f}' = (f(\mathbf{x}), f_{N+1})$ with mean vector $\mathbf{m}'_\mathbf{f}$ and covariance matrix $\mathbf{V}'_\mathbf{f}$. Thus, the result of the inner integral in Eq. (A.32) is

$$\begin{aligned} &\int p(z_0 \,|\, \mathbf{f})\, p(z_1 \,|\, \mathbf{c}_1) \cdots p(z_K \,|\, \mathbf{c}_K) \mathcal{N}(\mathbf{f} \,|\, \mathbf{m}_0, \mathbf{V}_0) \prod_{k=1}^{K} \mathcal{N}(\mathbf{c}_k \,|\, \mathbf{m}_k, \mathbf{V}_k) df_1 \cdots df_N \, d\mathbf{c}_1 \cdots d\mathbf{c}_K \\ &\quad = \mathcal{N}\left(\mathbf{f}' \,|\, \mathbf{m}'_\mathbf{f}, \mathbf{V}'_\mathbf{f}\right) \prod_{k=1}^{K} \mathcal{N}\left(c_k(\mathbf{x}) \,|\, m'_k, v'_k\right) , \end{aligned} \tag{A.34}$$

where, using Eqs. (3.22) and (3.24) of Rasmussen and Williams (2006) for the means and

variances repectively, we have the definitions

$$
\begin{aligned}
\left[\mathbf{m}'_{\mathbf{f}}\right]_1 &= \mathbf{k}^f_{\text{final}}(\mathbf{x})^\top \left[\mathbf{K}^f_{\star,\star}\right]^{-1} \mathbf{m}^{\mathbf{f}}\,, \\
\left[\mathbf{m}'_{\mathbf{f}}\right]_2 &= \left[\mathbf{m}^{\mathbf{f}}\right]_{N+1}\,, \\
\left[\mathbf{V}'_{\mathbf{f}}\right]_{1,1} &= k_f(\mathbf{x},\mathbf{x}) - \mathbf{k}^f_{\text{final}}(\mathbf{x})^\top \left\{ \left[\mathbf{K}^f_{\star,\star}\right]^{-1} + \left[\mathbf{K}^f_{\star,\star}\right]^{-1} \mathbf{V}^{\mathbf{f}} \left[\mathbf{K}^f_{\star,\star}\right]^{-1} \right\} \mathbf{k}^f_{\text{final}}(\mathbf{x})\,, \\
\left[\mathbf{V}'_{\mathbf{f}}\right]_{2,2} &= \left[\mathbf{V}^{\mathbf{f}}\right]_{N+1,N+1}\,, \\
\left[\mathbf{V}'_{\mathbf{f}}\right]_{1,2} &= k_f(\mathbf{x},\mathbf{x}^{(m)}_\star) - \mathbf{k}^f_{\text{final}}(\mathbf{x})^\top \left\{ [\mathbf{K}^f_{\star,\star}]^{-1} + [\mathbf{K}^f_{\star,\star}]^{-1} \mathbf{V}^{\mathbf{f}} [\mathbf{K}^f_{\star,\star}]^{-1} \right\} \mathbf{k}^f_{\text{final}}(\mathbf{x}^{(m)}_\star)\,, \\
m'_k &= \mathbf{k}^k_{\text{final}}(\mathbf{x})^\top \left[\mathbf{K}^k_{\star,\star}\right]^{-1} \mathbf{m}^{\mathbf{c}_k}\,, \\
v'_k &= k_k(\mathbf{x},\mathbf{x}) - \mathbf{k}^k_{\text{final}}(\mathbf{x})^\top \left\{ \left[\mathbf{K}^k_{\star,\star}\right]^{-1} + \left[\mathbf{K}^k_{\star,\star}\right]^{-1} \mathbf{V}^{\mathbf{c}_k} \left[\mathbf{K}^k_{\star,\star}\right]^{-1} \right\} \mathbf{k}^k_{\text{final}}(\mathbf{x})\,,
\end{aligned}
$$

and

- $\mathbf{m}^{\mathbf{f}}$ and $\mathbf{V}^{\mathbf{f}}$ are the posterior mean and posterior covariance matrix of $\mathbf{f}$ given by $\tilde{q}$.
- $\mathbf{k}^f_{\text{final}}(\mathbf{x})$ is the $(N+1)$-dimensional vector with the prior cross-covariances between $f(\mathbf{x})$ and elements of $\mathbf{f}$, i.e., $f(\mathbf{x}_1), \ldots, f(\mathbf{x}_N), f(\mathbf{x}^{(m)}_\star)$.
- $\mathbf{K}^f_{\star,\star}$ is an $(N+1) \times (N+1)$ matrix with the prior covariances between elements of $\mathbf{f}$.
- $\mathbf{k}^f_{\text{final}}(\mathbf{x}^{(m)}_\star)$ is the $(N+1)$-dimensional vector with the prior cross-covariances between $f(\mathbf{x}^{(m)}_\star)$ and $f(\mathbf{x}_1), \ldots, f(\mathbf{x}_N), f(\mathbf{x}^{(m)}_\star)$.
- $\mathbf{k}^k_{\text{final}}(\mathbf{x}^k)$ is an $(N+1)$ vector with the cross-covariances between $c_k(\mathbf{x})$ and the elements of $\mathbf{c}_k$.
- $\mathbf{K}^k_{\star,\star}$ is an $(N+1) \times (N+1)$ covariance matrix with the prior covariances between $c_k(\mathbf{x}_1), \ldots, c_k(\mathbf{x}_n), c_k(\mathbf{x}^{(m)}_\star)$.

We have now computed the inner integral in Eq. (A.31), leaving us with our next approximation

of the NFCPD,

$$p(f(\mathbf{x}), c_1(\mathbf{x}), \ldots, c_K(\mathbf{x}) \,|\, \mathcal{D}^f, \mathcal{D}^1, \ldots, \mathcal{D}^K, \mathbf{x}, \mathbf{x}_\star^{(m)}) \approx$$
$$\frac{1}{Z} \int \left(\left\{ \prod_{k=1}^{K} \Theta[c_k(\mathbf{x})] \right\} \Theta\left[f(\mathbf{x}) - f_{N+1}\right] + \left\{ 1 - \prod_{k=1}^{K} \Theta\left[c_k(\mathbf{x})\right] \right\} \right)$$
$$\left\{ \prod_{nk=1}^{K} \mathcal{N}\left(c_k(\mathbf{x}) \,|\, m'_k, v'_k\right) \right\} \mathcal{N}\left(\mathbf{f}' \,|\, \mathbf{m}'_{\mathbf{f}}, \mathbf{V}'_{\mathbf{f}}\right) \, df_{N+1} \,. \tag{A.35}$$

Eq. (A.35) is the same as Eq. (4.18) in the main text. Note that the computations performed to obtain m'_k, v'_k, $\mathbf{m}'_{\mathbf{f}}$, and $\mathbf{V}'_{\mathbf{f}}$ do not depend on $\mathbf{x}$. In the following section we make a final Gaussian approximation to Eq. (A.35), which must be performed for every $\mathbf{x}$.

A.5 Final Gaussian approximation to the NFCPD, for each x

Because the right-hand side of Eq. (A.35) cannot be computed analytically, we approximate it with a product of Gaussians that have the same marginal means and variances. In particular, we approximate it as Eq. (4.19) in the main text, namely,

$$p\left(\mathbf{z} \,|\, \mathcal{D}, \mathbf{x}, \mathbf{x}_\star^{(m)}\right) \approx \mathcal{N}\left(f(\mathbf{x}) \,|\, \mu_f^{\text{NFCPD}}(\mathbf{x}), v_f^{\text{NFCPD}}(\mathbf{x})\right) \prod_{k=1}^{K} \mathcal{N}\left(c_k(\mathbf{x}) \,|\, \mu_k^{\text{NFCPD}}(\mathbf{x}), v_k^{\text{NFCPD}}(\mathbf{x})\right) \,,$$

where $\mu^f_{\text{NFCPD}}(\mathbf{x})$, $v^f_{\text{NFCPD}}(\mathbf{x})$ and $\mu^k_{\text{NFCPD}}(\mathbf{x})$ and $v^k_{\text{NFCPD}}(\mathbf{x})$ are the marginal means and marginal variances of $f(\mathbf{x})$ and $c_k(\mathbf{x})$ according to the right-hand-side of Eq. (A.35). Using Eq. (5.13) in Minka (2001b) we can compute these means and variances in terms of the derivatives of the normalization constant Z in Eq. (A.35), which is given by

$$Z = \left\{ \prod_{k=1}^{K} \Phi[\alpha_k] \right\} \Phi[\alpha] + \left\{ 1 - \prod_{k=1}^{K} \Phi[\alpha_k] \right\} \tag{A.36}$$

where

$$
\begin{aligned}
s &= \left[\mathbf{V}'_{\mathbf{f}}\right]_{1,1} + \left[\mathbf{V}'_{\mathbf{f}}\right]_{2,2} - 2\left[\mathbf{V}'_{\mathbf{f}}\right]_{1,2} \\
\alpha &= \frac{[1,\,-1]\mathbf{m}'_{\mathbf{f}}}{\sqrt{s}} \\
\alpha_k &= \frac{m'_k}{\sqrt{v'_k}}\,.
\end{aligned}
$$

Doing so yields

$$
\begin{aligned}
v_f^{\mathrm{NFCPD}}(\mathbf{x}) &= \left[\mathbf{V}'_{\mathbf{f}}\right]_{1,1} - \frac{\beta}{s}\left(\beta+\alpha\right)\left(\left[\mathbf{V}'_{\mathbf{f}}\right]_{1,1} - \left[\mathbf{V}'_{\mathbf{f}}\right]_{1,2}\right)^2 \\
\mu_f^{\mathrm{NFCPD}}(\mathbf{x}) &= \left[\mathbf{m}'_{\mathbf{f}}\right]_1 + \left(\left[\mathbf{V}'_{\mathbf{f}}\right]_{1,1} - \left[\mathbf{V}'_{\mathbf{f}}\right]_{1,2}\right)\frac{\beta}{\sqrt{s}} \\
v_k^{\mathrm{NFCPD}}(\mathbf{x}) &= \left\{[v'_k]^{-1} + \tilde{a}\right\}^{-1} \\
\mu_k^{\mathrm{NFCPD}}(\mathbf{x}) &= v_{N,\mathrm{const}}^k(\mathbf{x})\left\{[m'_k]^{-1}[v'_k]^{-1} + \tilde{b}\right\}\,,
\end{aligned}
\tag{A.37}
$$

where

$$
\begin{aligned}
\beta &= \frac{\left\{\prod_{k=1}^K \Phi[\alpha_k]\right\}\phi(\alpha)}{Z} \\
\tilde{a} &= -\left\{\frac{\partial^2 \log Z}{\partial\left[m'_k\right]^2} + v'_k\right\}^{-1} \\
\tilde{b} &= \tilde{a}\left\{m'_k + \frac{\sqrt{v'_k}}{\alpha_k+\beta_k}\right\} \\
\beta_k &= \frac{\phi(\alpha_n)}{Z\Phi(\alpha_n)}(Z-1)\,, \\
\frac{\partial^2 \log Z}{\partial\left[m'_k\right]^2} &= -\frac{\beta_k\left\{\alpha_k+\beta_k\right\}}{v'_k}\,.
\end{aligned}
$$

Appendix B

Implementation Considerations

The following are details of the implementation used in the experiments in Chapter 6 of this thesis, as well as general implementation considerations for constrained Bayesian optimization and the methods presented in this thesis.

B.1 Kriging believer implementation

The kriging believer method for asynchronous parallel Bayesian optimization (Section 2.7) can be made to run faster in two ways. The kriging believer works by attributing to a pending experiment the mean value according to the GP. We then must condition on this "dummy" observation and update the GP posterior. The first implementation trick is that the GP posterior mean does not need to be updated after conditioning on this extra observation equal to the posterior mean. We can see this by Eq. (4) in Hennig and Schuler (2012), which we repeat here (with the notation changed slightly to be consistent with this thesis):

$$\mu(\mathbf{x}') = \mu_0(\mathbf{x}') + \Sigma_0(\mathbf{x}', \mathbf{x})\sigma_0^{-1}(\mathbf{x})(y - \mu_0(\mathbf{x}))\,. \tag{B.1}$$

Here, X_0 is the original design matrix, $\mathbf{x}$ is the location of data collection with corresponding output y, Σ_0 is the posterior predictive covariance matrix conditioned on the original data, σ_0 is the posterior predictive marginal variance conditioned on the original data, μ is the new posterior predictive mean, μ_0 is the original posterior predictive mean, and $\mathbf{x}'$ is some test point.

From Eq. (B.1) we see that if $y = \mu_0(\mathbf{x})$, i.e. if the observation at $\mathbf{x}$ falls exactly at the posterior predictive mean value, then $\mu(\mathbf{x}') = \mu_0(\mathbf{x}')$, i.e. the posterior predictive mean is unchanged for all $\mathbf{x}'$.

The second implementation trick for the kriging believer method is based on the observation is that to compute the posterior variance, we need to compute the Cholesky decomposition of the kernel matrix $K(X, X)$. This kernel matrix is precisely the original kernel matrix $K(X_0, X_0)$ plus an additional row and column corresponding to the new point $\mathbf{x}$. Thus, we can compute the Cholesky decomposition of $K(X, X)$ using a *rank-one update* to the Cholesky decomposition of $K(X_0, X_0)$ (see Dongarra et al. (1979)). This reduces the computational burden from $\mathcal{O}(N^3)$ to $\mathcal{O}(N^2)$ where N is the number of collected data.

B.2 Optimizing the acquisition function

Optimizing the acquisition function in Bayesian optimization is a bound-constrained global optimization problem. Often, derivative information is available. The implementation used in this thesis uses implementations in the NLopt package (Johnson) of the Method of Moving Asymptotes (MMA) (Svanberg, 2002) when derivative information is present and Bound Optimization by Quadratic Optimization (BOBYQA) (Powell, 2009) when derivative information is not present. In either case the optimizer is initialized with the point giving the best acquisition function value from a grid of 10^4 points from the low-discrepancy Sobol sequence plus the locations of collected data and 10 additional points drawn i.i.d. from a Gaussian distribution with mean equal to the current recommendation and variance 10^{-8}. One could also generate this grid through Latin hypercube sampling, Halton sequences, or a number of other methods. The tolerance for NLopt convergence is 10^{-4} in the scaled units.

B.3 Making recommendations

There are several ways to make recommendations in Bayesian optimization. A principled approach is to follow the problem formulation that is being optimized. For example, if we are using probabilistic constraints and thus formulating the problem as in Eq. (3.3), then we make

a recommendation following Eq. (3.3) as well. Namely, we do a constrained minimization on the GP predictive mean subject to the probabilistic constraints being satisfied. Fortunately, gradient information is available both for the GP predictive mean and for the probabilistic constraints. To perform the actual optimization, we use the same methods as when optimizing the acquisition function (Appendix B.2). For EIC, we use this procedure to make a recommendation $\widetilde{\mathbf{x}}$ at every iteration, and then use the GP mean value $\mathbb{E}[f(\widetilde{\mathbf{x}})]$ as the incumbent η.

B.4 The initial design

Typically Bayesian optimization schemes start by running some initial experiments before beginning the model-based approach. These initial experiments are also called the *initial design*. Here we use as the initial design just one random point (in the decoupled case, we pick the objective task). Difficulty arises when dealing with *hidden constraints* that, when violated, prevent us from observing the objective function. In this case, it is possible to enter a loop in which a random point is queried, the constraint is violated, no objective function value is reported, and thus a random point is queried again. In such a situation one ideally begins the model-based optimization immediately even though zero objective function observations are available.

B.5 Normalizing the input and output spaces

We normalize the inputs by linearly mapping the input space into the unit hypercube. We normalize the output space as follows. The observations of the objective function are shifted to have zero mean and then scaled to have unit variance. Observations of constraint functions cannot be shifted (because the value zero has special meaning) and require more care in normalization. Here, we normalize constraint observations by scaling them such that the maximum absolute value of the scaled observations is one. For binary or binomial constraints, no normalization is used.

B.6 GP implementation

Efficient and stable implementations of the key GP computations such as Eqs. (2.1) and (2.4) are given in Rasmussen and Williams (2006), Algorithm 2.1 and Appendix A.4. For numerical stability, in our implementation we add a small constant *jitter* in the range of $10^{-9} - 10^{-6}$ to the diagonal of the kernel matrix $K(X, X)$ before taking its Cholesky decomposition. This is equivalent to adding the same amount to the assumed noise variance.

Unless otherwise specified, we use the Matérn 5/2 kernel (Section 2.2.1) in experiments. By default we use slice sampling (Neal, 2000) to take 10 samples of the GP kernel hyperparameters after burning 20 samples at each iteration. The hyperparameters are initialized as follows: length scales are 0.1, mean is zero, amplitude is 1, noise is 10^-4. Each length scale has a Beta prior on $\ell/5$ with $\alpha = 1.5, \beta = 7.0$. The amplitude has a rectified Gaussian prior with $\mu = 1, \sigma^2 = 1$, the mean has a Gaussian prior with $\mu = 1, \sigma^2 = 1$, and the noise has a rectified Horseshoe prior (Carvalho et al., 2009) with parameter 0.1.

When the posterior distribution cannot be computed in closed form due to a non-Gaussian likelihood, we use elliptical slice sampling (Murray et al., 2010) to sample the latent functions. We also use the prior whitening procedure described in Murray and Adams (2010) to avoid poor mixing due to the strong coupling of the latent values and the kernel hyperparameters. Following Snoek et al. (2012), we use the Matérn 5/2 kernel for the GP prior, which corresponds to the assumption that the function being modeled is twice differentiable. We assume the objective and all constraints are independent and model them with independent GPs. When constraints are modeling with GP classifiers, we use the Gaussian CDF as the response function. Note that since the objective and constraints are all modeled independently, they need not all be modeled with GPs or even with the same types of models as each other. Any combination of models suffices, so long as each one represents its uncertainty about the true function values. However, the PESC acquisition function is specific to GP models.

B.7 Sampling in EIC-D

Section 3.3 introduces EIC for decoupled constraints, or EIC-D. The implementation of the sampling approach in Eq. (3.6) is similar to Villemonteix et al. (2009), but must be extended to constraints. First, estimating $p(\mathbf{x}_\star)$ requires a discretization of the space. In the spirit of Hennig and Schuler (2012), we form a discretization of N_d points by taking the top N_d points according to the EIC criterion. Then, we extend the method of Villemonteix et al. (2009) to constrained optimization: we sample from the objective function GP and all K constraint GPs, and then find the minimum of the objective for which the constraint is satisfied for all K constraint samples.

B.8 EP implementation for PESC

B.8.1 Initialization and convergence

Initially EP sets the parameters of all the approximate factors to be zero. We use at the convergence criterion that the absolute change in all parameters should be below 10^{-4}.

B.8.2 Damping

To improve convergence we use damping (Minka and Lafferty, 2002). If $\tilde{h}_n^{\text{new}}$ is the minimizer of the KL-divergence, damping entails using instead $\tilde{h}_n^{\text{damped}}$ as the new factor, as defined below:

$$\tilde{h}_n^{\text{damped}} = [\tilde{h}_n^{\text{new}}]^\epsilon + \tilde{h}_n^{1-\epsilon}, \tag{B.2}$$

where $\tilde{h}_n$ is the factor at the previous iteration. We do the same for $\tilde{g}_k$. The parameter ϵ controls the amount of damping, with $\epsilon = 1$ corresponding to no damping. We initialize ϵ to 1 and multiply it by a factor of 0.99 at each iteration. Furthermore, the factors $\tilde{h}_n$ and $\tilde{g}_k$ are updated in parallel (i.e. without updating $\tilde{q}$ in between) in order to speed up convergence (Gerven et al., 2009). During the execution of EP, some covariance matrices may become non positive definite due to an excessively large step size (i.e. large ϵ). If this issue is encountered during an EP iteration, the damping parameter is reduced by half and the iteration is repeated. The EP algorithm is terminated when the absolute change in all the parameters in $\tilde{q}$ is less than

10^{-4}.

B.8.3 Numerical stability

The EP approximation in PESC can easily become numerically unstable unless care is taken. The calculations involve many matrix inversions. These should be avoided whenever possible by replacing the inversion with a Cholesky decomposition plus solving a triangular system. However, matrix inversion cannot be avoided entirely. As mentioned in Appendix B.6, it is useful to add jitter to the diagonal of the kernel matrix before computing its Cholesky decomposition when making GP predictions. We found that this jitter was not needed in other places; it is sufficient to add it when making GP predictions.

One case of interest is robustly computing $\log(1 + \exp(x))$. We compute this as follows:

$$[\log(1 + \exp(x))]_{\text{robust}} = \begin{cases} \exp(x), & \text{if } \exp(x) < \epsilon \\ x, & \text{if } x > \log(\xi) \\ \log(1 + \exp(x)), & \text{otherwise} \end{cases} \tag{B.3}$$

for some small constant ϵ and some large constant ξ. Eq. (B.3) is derived as follows. If $z \equiv \exp(x)$ is small, then we can apply the Taylor expansion $\log(1 + z) \approx z$ so $\log(1 + \exp(x)) \approx \exp(x)$. If $\exp(x)$ is very large, then we can say that 1 is negligible compared to $\exp(x)$, and thus $\log(1 + \exp(x)) \approx \log(\exp(x)) = x$. Otherwise, we simply return $\log(1 + \exp(x))$. In our implementation we use $\epsilon = 10^{-6}$ and $\xi = 100$.

Another case of interest is robustly computing $\log(1 - \exp(x))$ when $x < 0$. We compute this as follows:

$$[\log(1 - \exp(x))]_{\text{robust}} = \begin{cases} -\exp(x), & \text{if } \exp(x) < \epsilon \\ \log(-x), & \text{if } -\epsilon < x < 0 \\ \log(1 - \exp(x)), & \text{otherwise} \end{cases} \tag{B.4}$$

for some small constant $\epsilon > 0$. Eq. (B.4) is derived as follows. If $z \equiv \exp(x)$ is small, then

we can apply the Taylor expansion $\log(1-z) \approx -z$ so $\log(1-\exp(x)) \approx -\exp(x)$. If $x > -\epsilon$, meaning that $|x|$ is very small, then we can use the Taylor approximation $\exp(x) \approx 1+x$, so that $\log(1-\exp(x)) \approx \log(1-(1+x)) = \log(-x)$. If neither of these cases holds, we simply return $\log(1-\exp(x))$. In our implementation we use $\epsilon = 10^{-6}$.

A third case of interest is robustly computing $\log(\exp(x)+\exp(y)$. One can easily derive this for a sum of arbitrarily many terms but here we use only two terms for simplicity. We compute this as follows:

$$[\log(\exp(x)+\exp(y))]_{\text{robust}} = \begin{cases} x + [\log(1+\exp(y-x))]_{\text{robust}}, & \text{if } x > y \\ y + [\log(1+\exp(x-y))]_{\text{robust}}, & \text{otherwise} \end{cases} \tag{B.5}$$

This works by factoring out the larger value to prevent computing the exp of a very large or small value, if both x and y are very large or small. For example, if $x > y$, then

$$\log(\exp(x)+\exp(y)) = \log(\exp(x)[1+\exp(y-x)]) = x + \log(1+\exp(y-x)),$$

and vice versa when $x < y$.

Finally, the PESC algorithm requires computing the log of the Gaussian CDF (denoted $\Phi(x)$) accurately for large values of x. Some implementations, such as the SciPy python implementation, saturate to 1 for arguments as small as 9.0 (SciPy version 0.14.0). Thus $\log\Phi(9.0)$ is computed as exactly 0.0. Curiously, this situation is not the same when x is negative. For example, in the same version of SciPy $\Phi(-9)$ is computed without saturating to zero, and then $\log\Phi(-9)$ can be computed with no issue. To make use of this, we do the following:

$$[\log\Phi(x)]_{\text{robust}} = \begin{cases} -\log\Phi(-x), & \text{if } x > 5 \\ -\Phi(-x), & \text{otherwise} \end{cases} \tag{B.6}$$

The logic is as follows. When $x > 5$, $\Phi(x)$ is very close to 1. Thus, $1-\Phi(x)$ is a very small positive quantity. Then, we use the Taylor expansion approximation $\log(1-z) \approx -z$ for small

z to yield

$$\log(\Phi(x)) = \log(1 - (1 - \Phi(x))) \approx -(1 - \Phi(x)) = -\Phi(-x)\,.$$

When the standard implementation of the Gaussian CDF saturates easily only on the positive end, Eq. (B.6) can make use of the CDF of negative values in this way to accurately compute the log Gaussian CDF for relatively large positive values.

Finally, if $\mathbf{x}$ is close to $\mathbf{x}_\star$, then the quantity s (Eq. (A.23)) may be very close to zero, causing numerical instability. To avoid this, whenever $s < 10^{-10}$ we scale the covariance term $\left[\mathbf{V}_{\mathbf{f}'}^{\text{pred}}\right]_{1,2}$ down until $s \geq 10^{-10}$.

B.9 Sampling $\mathbf{x}_\star$ in PESC

The procedure for sampling $\mathbf{x}_\star$ is a constrained extension to the method outlined in Hernández-Lobato et al. (2014). We perform a finite basis function GP approximation to the objective and each constraint, and then sample all of these functions. We then minimize these analytical samples, but now using a constrained optimization method. In particular we use the Method of Moving Asymptotes (MMA) (Svanberg, 2002) as implemented in the NLopt package (Johnson). We use a grid size of 10^3; the number of random features is 10^3; the NLopt convergence tolerance is 10^{-6} in the scaled input space units.

As described in Hernández-Lobato et al. (2014), if the Gaussian covariance matrix is the sum of a low rank matrix and a diagonal matrix, we can use a faster method to sample from it. This method is outlined in Appendix B.2 of Seeger (2008). The cost of this method is is $\mathcal{O}(N^2M)$ where N is the number of collected data and M is the number of random features. Sampling with the naive method takes $\mathcal{O}(M^3)$ operations because we must take the Cholesky decomposition of an $M \times M$ matrix. Given that in our implementation $M = 1000$ and typically $N \lesssim 100$, this method can speed up this sampling procedure by orders of magnitude. This method could be sped up further by using a quasi-random numbers to sample the random features, thus reducing the number of random features needed to attain the same approximation quality.

B.10 PESC-F implementation details

B.10.1 Rank-one Cholesky update in PESC-F

In PESC-F because the GP hyperparameters (and in particular the length scales) are not changed since the last iteration, the kernel matrix $K(X, X)$ from Eq. (2.1) is unchanged except for the addition of a new row and column. Given this, we can compute the Cholesky decomposition of the new kernel matrix with a rank-one update of the Cholesky decomposition of the current kernel matrix, as discussed in Appendix B.1. As this $\mathcal{O}(N^3)$ operation is the main bottleneck for GP-based Bayesian optimization as N gets large, this trick can significantly speed up the fast updates in PESC-F. In fact, this trick applies more generally beyond PESC-F or even any fast-update method: any Bayesian optimization method that does not update the GP hyperparameters at every iteration can take advantage of the rank-one Cholesky update.

B.10.2 Speeding up the acquisition function maximization

In our implementation of PESC-F, the grid size for maximizing the acquisition function is reduced from 10^4 to 10^3, and the tolerance for the convergence of the optimization routine is increased from 10^{-4} to 10^{-3}.

B.11 Fast implementation of the IAGO algorithm

The IAGO algorithm Section 2.4.5 can be sped up significantly by using the implementation described below and illustrated in Algorithm 5. The implementation relies on the fact that Algorithm 2 is dominated by the cost of sampling the functions f_j on line 6. This is true even if we pre-compute the Cholesky decomposition of the covariance matrix of f, which we can do because the covariance matrix does not depend on j or y_i. Given the Cholesky decomposition of the covariance matrix, sampling a multivariate normal vector of length N_3 requires N_3^2 operations because we must multiply a $N_3 \times N_3$ matrix with a vector of length N_3. Thus the total cost of the method in Algorithm 2, for each $\mathbf{x}$, is $\mathcal{O}(N_1 N_2 N_3^2)$, where N_1 is the number of samples of y, N_2 is the number of samples of f in estimating $p(\mathbf{x}_\star \,|\, \mathcal{D} \cup \{(\mathbf{x}, y)\})$, and N_3 is the size of the grid on which samples of f are drawn.

Algorithm 5 The IAGO algorithm (Villemonteix et al., 2009), faster implementation

1: **Inputs:** data set $\mathcal{D}$, candidate $\mathbf{x}$, # of y samples N_1, # of f samples N_2, grid size N_3
2: initialize the grid X' of size $N_3 \times D$
3: compute the posterior predictive mean, μ_0, on the grid using Eq. (2.1)
4: compute posterior predictive covariance matrix, Σ, on the grid, adding $\mathbf{x}$ to design matrix
5: compute the Cholesky decomposition of the covariance matrix: $\Sigma^{1/2} = \text{chol}(\Sigma)$
6: **for** $j = 1$ **to** N_2 **do**
7: sample N_3 i.i.d. standard normals: $z_k \sim \mathcal{N}(0, 1)$ for $k = 1, \ldots, N_3$
8: sample a zero-mean function: $f_j = \Sigma^{1/2}\mathbf{z}$
9: **end for**
10: **for** $i = 1$ **to** N_1 **do**
11: sample $y_i \sim p(y|\mathbf{x}, \mathcal{D})$ on the grid
12: update the mean on the grid: $\mu(X') = \mu_0(X') + \Sigma(X', X)\Sigma^{-1}(X, X)(y_i - \mu_0(X'))$
13: initialize $p_{\text{empirical}} = \mathbf{0}$ for grid of size N_3
14: **for** $j = 1$ **to** N_2 **do**
15: shift f_j to have the correct mean: $f_j = f_j + \mu(X')$
16: compute the grid index $k = \arg\min f$
17: record the result: $p_{\text{empirical}}[k] = p_{\text{empirical}}[k] + \frac{1}{N_2}$
18: **end for**
19: compute the entropy $\mathrm{H}_i = -\sum_{j=1}^{N_2} p_{\text{empirical}}[j] \log p_{\text{empirical}}[j]$
20: **end for**
21: compute the average entropy $\mathrm{H} = \frac{1}{N_1}\sum_{i=1}^{N_1} \mathrm{H}_i$
22: **Output:** $\alpha_{\text{SUR}}(\mathbf{x}) = -\mathrm{H}$

The trick to speed up this computation relies on two properties of GPs and Gaussian random variables. The first property is that the posterior predictive covariance in a GP depends only on the collection data locations $\{\mathbf{x}_i\}$, not on the collected data values $\{y_i\}$ (see Section 2.2). The second property is that a Gaussian random variable with mean μ and covariance matrix Σ can be sampled by first sampling a Gaussian random variable with covariance matrix Σ and mean zero, and then adding μ to this sample. Because the sampling step takes $\mathcal{O}(N_3^2)$ time and adding the mean only takes $\mathcal{O}(N_3)$ time, we can interpret this property as allowing us to break up the sampling into the slow step and the fast step.

In line 6 of Algorithm 2, f is sampled conditional on the fantasy observation y_i. But, we can split up the sampling into two steps: first, generate a zero-mean sample with the correct covariance, and, second, add the correct mean (the second property described above). However, only the mean depends on y_i; this is the first property described above. Therefore, we can do the first step of the sampling without conditioning on y_i. This is precisely the trick employed in Algorithm 5. In lines 2-9, we pre-compute the first step of the f sampling. In line 15-17, we

re-use these pre-computed samples for all i. Thus, we perform the expensive $\mathcal{O}(N_3^2)$ operation only N_2 times instead of $N_1 N_2$ times. This reduces the entire computational cost of the method from $\mathcal{O}(N_3^3 + N_1 N_2 N_3^2)$ to $\mathcal{O}(N_3^3 + N_2 N_3^2 + N_1 N_2 N_3)$, where the N_3^3 terms come from computing the Cholesky decomposition of the posterior predictive covariance matrix. For reasonable choices of N_1, N_2, and N_3 such as $N_1 = 10$, $N_2 = 1000$, $N_3 = 1000$, this fast method achieves an order of magnitude speedup.

Note: it may seem that computing the GP mean conditioned on observing a sample y_i also requires $\mathcal{O}(N_3^2)$ operations, thus rendering Algorithm 5 no better than Algorithm 2. However, Eq. (2.1) shows us that the posterior predictive mean is a *linear* function of the observation vector $\mathbf{y}$. Thus, augmenting the collected data with a single observation fantasy y_i only constitutes an additional term in the posterior predictive mean. Thus updating the mean is faster than recomputing it entirely, and only requires $\mathcal{O}(N_3)$ operations. This is another trick, and is implemented in lines 3, 12 and 15. Note also that the Cholesky decomposition in line 4 of Algorithm 5 can be performing using a rank-one update to the Cholesky decomposition of the kernel matrix evaluated at the non-augmented design matrix; see Appendix B.10.1 for more details.

Bibliography

Alekh Agarwal, Olivier Chapelle, Miroslav Dudík, and John Langford. A reliable effective terascale linear learning system. *The Journal of Machine Learning Research*, 15(1):1111–1133, 2014.

Dirk V. Arnold and Nikolaus Hansen. A (1+1)-CMA-ES for constrained optimisation. In *Genetic and Evolutionary Computation Conference*, pages 297–304, 2012.

Javad Azimi. *Bayesian optimization with empirical constraints*. PhD thesis, Oregon State University, 2012.

Javad Azimi, Alan Fern, and Xiaoli Z. Fern. Batch Bayesian optimization via simulation matching. In *Advances in Neural Information Processing Systems*, pages 109–117. 2010a.

Javad Azimi, Xiaoli Fern, Alan Fern, Elizabeth Burrows, Frank Chaplen, Yanzhen Fan, Hong Liu, Jun Jaio, and Rebecca Schaller. Myopic policies for budgeted optimization with constrained experiments. In *AAAI-10: 24th Conference on Artificial Intelligence*, 2010b.

Javad Azimi, Ali Jalali, and Xiaoli Fern. Dynamic batch Bayesian optimization. In *NIPS Workshop on Bayesian optimization*, 2011.

Javad Azimi, Ali Jalali, and Xiaoli Zhang Fern. Hybrid batch bayesian optimization. In *International Conference on Machine Learning*, pages 1215–1222, 2012.

James S. Bergstra, Rémi Bardenet, Yoshua Bengio, and Bálázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, pages 2546–2554. 2011.

Dimitri P. Bertsekas. *Constrained Optimization and Lagrange Multiplier Methods*. Athena Scientific, 1996.

David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent Dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

Craig Boutilier. On the foundations of expected expected utility, 2001. URL `http://www.cs.utoronto.ca/kr/papers/foundations.pdf`.

Eric Brochu, Tyson Brochu, and Nando de Freitas. A Bayesian interactive optimization approach

to procedural animation design. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2010a.

Eric Brochu, Vlad M. Cora, and Nando de Freitas. A tutorial on Bayesian optimization of expensive cost functions, 2010b. arXiv:1012.2599 [cs.LG].

Sébastien Bubeck, Rémi Munos, and Gilles Stoltz. Pure exploration in multi-armed bandits problems. In *Proceedings of the 20th International Conference on Algorithmic Learning Theory*, pages 23–37, Berlin, Heidelberg, 2009.

Adam D. Bull. Convergence rates of efficient global optimization algorithms. *Journal of Machine Learning Research*, (3-4):2879–2904, 2011.

Carlos M. Carvalho, Nicholas G. Polson, and James G. Scott. Handling sparsity via the horseshoe. In *International Conference on Artificial Intelligence and Statistics*, pages 73–80, 2009.

Olivier Chapelle and Lihong Li. An empirical evaluation of Thompson sampling. In *Advances in Neural Information Processing Systems 24*, pages 2249–2257. 2011.

Clment Chevalier and David Ginsbourger. Fast computation of the multi-points expected improvement with applications in batch selection. In *Learning and Intelligent Optimization*, volume 7997, pages 59–69. 2013.

Scott Clark. *Parallel Machine Learning Algorithms in Bioinformatics and Global Optimization*. PhD thesis, Cornell University, 2012.

Andrew R. Conn, Katya Scheinberg, and Luis N. Vicente. *Introduction to Derivative-Free Optimization*, volume 8. Society for Industrial and Applied Mathematics, 2009.

Nando de Freitas, Alex Smola, and Masrour Zoghi. Exponential regret bounds for Gaussian process bandits with deterministic observations. In *International Conference on Machine Learning*, pages 1743–1750, 2012.

Geng Deng and Michael C. Ferris. Extension of the DIRECT optimization algorithm for noisy functions. In *Winter Simulation Conference*, pages 497–504, 2007.

Josip Djolonga, Andreas Krause, and Volkan Cevher. High dimensional Gaussian Process bandits. In *Advances in Neural Information Processing Systems*, pages 1025–1033, 2013.

Jack J. Dongarra, James R. Bunch, Cleve B. Moler, and Gilbert W. Stewart. *LINPACK User's Guide*. Philadelphia, PA, 1979.

Simon Duane, Anthony D. Kennedy, Brian J. Pendleton, and Duncan Roweth. Hybrid Monte Carlo. *Physics letters B*, 195(2):216–222, 1987.

Alexander Forrester, Andras Sobester, and Andy Keane. *Engineering Design via Surrogate*

Modelling. John Wiley and Sons, 2008.

Andrew Frank and Arthur Asuncion. UCI machine learning repository, 2010.

Michael P Friedlander and Paul Tseng. Exact regularization of convex programs. *SIAM Journal on Optimization*, 18(4):1326–1350, 2007.

Jacob R. Gardner, Matt J. Kusner, Zhixiang (Eddie) Xu, Kilian Q. Weinberger, and John P. Cunningham. Bayesian optimization with inequality constraints. In *International Conference on Machine Learning*, pages 937–945, 2014.

Andrew Gelman and Donald R. Rubin. A single series from the Gibbs sampler provides a false sense of security. In *Bayesian Statistics*, pages 625–32. Oxford University Press, 1992.

Donald Geman and Bruno Jedynak. An active testing model for tracking roads in satellite images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(1):1–14, 1996.

Marcel V. Gerven, Botond Cseke, Robert Oostenveld, and Tom Heskes. Bayesian source localization with the multivariate Laplace prior. In *Advances in Neural Information Processing Systems*, pages 1901–1909, 2009.

John Geweke. Evaluating the accuracy of sampling-based approaches to the calculation of posterior moments. In *Bayesian Statistics*, pages 169–193. University Press, 1992.

David Ginsbourger and Rodolphe Riche. Towards Gaussian process-based optimization with finite time horizon. In *Advances in Model-Oriented Design and Analysis*, pages 89–96. 2010a.

David Ginsbourger and Rodolphe Le Riche. Dealing with asynchronicity in parallel Gaussian process based global optimization. `http://hal.archives-ouvertes.fr/hal-00507632`, 2010b.

David Ginsbourger, Rodolphe Le Riche, and Laurent Carraro. Kriging is well-suited to parallelize optimization. In *Computational Intelligence in Expensive Optimization Problems*, pages 131–162. Springer-Verlag, 2010.

Robert B. Gramacy and Herbert K. H. Lee. Optimization under unknown constraints. *Bayesian Statistics*, 9, 2011.

Robert B. Gramacy, Genetha A. Gray, Sebastien Le Digabel, Herbert K. H. Lee, Pritam Ranjan, Garth Wells, and Stefan M. Wild. Modeling an augmented Lagrangian for improved blackbox constrained optimization, 2014. arXiv:1403.4890v2 [stat.CO].

Nikolaus Hansen and Andreas Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In *International Conference on Evolutionary Computation*, pages 312–317, 1996.

Nikolaus Hansen, Andr S. P. Niederberger, Lino Guzzella, and Petros Koumoutsakos. A method

for handling uncertainty in evolutionary optimization with an application to feedback control of combustion. *IEEE Transactions on Evolutionary Computation*, 13(1):180–197, 2009.

Philipp Hennig and Christian J. Schuler. Entropy search for information-efficient global optimization. *Journal of Machine Learning Research*, 13(1):1809–1837, 2012.

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.

Matthew D. Hoffman, David M. Blei, and Francis Bach. Online learning for latent Dirichlet allocation. In *Advances in Neural Information Processing Systems*, pages 856–864, 2010.

Matthew W. Hoffman and Bobak Shahriari. Modular mechanisms for Bayesian optimization. In *NIPS Workshop on Bayesian Optimization*, 2014.

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

Janis Janusevskis, Rodolphe Le Riche, David Ginsbourger, and Ramunas Girdziusas. Expected improvements for the asynchronous parallel global optimization of expensive functions: Potentials and challenges. In *International Conference on Learning and Intelligent Optimization*, pages 413–418. Springer, 2012.

Steven G. Johnson. The NLopt nonlinear-optimization package. URL `http://ab-initio.mit.edu/nlopt`.

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

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.

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.

Kirthevasan Kandasamy, Jeff Schneider, and Barnabas Poczos. High dimensional bayesian optimisation and bandits via additive models, 2015. arXiv:1503.01673 [stat.ML].

Oliver Kramer. A review of constraint-handling techniques for evolution strategies. *Applied Computational Intelligence and Soft Computing*, 3:1–19, 2010.

Andreas Krause. *Optimizing Sensing: Theory and Applications*. PhD thesis, Carnegie Mellon University, December 2008.

Gunther Leobacher and Friedrich Pillichshammer. *Introduction to quasi-Monte Carlo integration and applications*. Springer, 2014.

Dan Lizotte. *Practical Bayesian Optimization*. PhD thesis, University of Alberta, Edmonton, Alberta, 2008.

David J. C. MacKay. Information-based objective functions for active data selection. *Neural Computation*, 4(4):590–604, 1992.

Ruben Martinez-Cantin. BayesOpt: A Bayesian optimization library for nonlinear optimization, experimental design and bandits. *Journal of Machine Learning Research*, 15:3735–3739, 2014.

Thomas P. Minka. Expectation propagation for approximate Bayesian inference. In *Uncertainty in Artificial Intelligence*, pages 362–369, 2001a.

Thomas P. Minka. *A family of algorithms for approximate Bayesian inference*. PhD thesis, Massachusetts Institute of Technology, 2001b.

Thomas P. Minka and John Lafferty. Expectation-propagation for the generative aspect model. In *Uncertainty in Artificial Intelligence*, pages 352–359, 2002.

Jonas Mockus, Vytautas Tiesis, and Antanas Zilinskas. The application of bayesian methods for seeking the extremum. *Towards Global Optimization*, 2(117-129):2, 1978.

Iain Murray and Ryan P. Adams. Slice sampling covariance hyperparameters of latent Gaussian models. In *Advances in Neural Information Processing Systems*, pages 1723–1731, 2010.

Iain Murray, Ryan P. Adams, and David J. C. MacKay. Elliptical slice sampling. *Journal of Machine Learning Research*, 9:541–548, 2010.

Radford M. Neal. Slice sampling. *Annals of Statistics*, 31:705–767, 2000.

Radford M. Neal. MCMC using Hamiltonian dynamics. In *Handbook of Markov Chain Monte Carlo*, volume 2. Chapman and Hall/CRC, 2011.

John A. Nelder and Roger Mead. A simplex method for function minimization. *Computer Journal*, 7(4):308–313, 1965.

Jorge Nocedal and Steve J. Wright. *Numerical optimization*. Springer Series in Operations Research and Financial Engineering. 2006.

James M. Parr, Carren M. E. Holden, Alexander I. J. Forrester, and Andy J. Keane. Review of efficient surrogate infill sampling criteria with constraint handling. In *2nd International Conference on Engineering Optimization*, pages 1–10, 2010.

Anand Patil, David Huard, and Christopher Fonnesbeck. PyMC: Bayesian stochastic modelling

in Python. *Journal of Statistical Software*, 35(4):1–81, 2010.

Victor Picheny. A stepwise uncertainty reduction approach to constrained global optimization. In *International Conference on Artificial Intelligence and Statistics*, pages 787–795, 2014.

Martyn Plummer, Nicky Best, Kate Cowles, and Karen Vines. CODA: Convergence diagnosis and output analysis for MCMC. *R News*, 6(1):7–11, 2006.

Michael J. D. Powell. The BOBYQA algorithm for bound constrained optimization without derivatives. Technical report, Department of Applied Mathematics and Theoretical Physics, Cambridge, England, 2009.

Warren B. Powell and Ilya O. Ryzhov. *Optimal learning*, volume 841. John Wiley & Sons, 2012.

Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems 20*, pages 1177–1184. 2007.

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

R. Tyrrell Rockafellar. Augmented lagrange multiplier functions and duality in nonconvex programming. *SIAM Journal on Control*, 12(2):268–285, 1974.

Olivier Roustant, David Ginsbourger, and Yves Deville. DiceKriging, DiceOptim: Two R packages for the analysis of computer experiments by kriging-based metamodeling and optimization. *Journal of Statistical Software*, 51(1):1–55, 2012.

Stuart Russell, Eric Wefald, Maurice Karnaugh, Richard Karp, David McAllester, Devika Subramanian, and Michael Wellman. Principles of Metareasoning. In *Artificial Intelligence*, volume 49, pages 400–411, 1991.

Michael J. Sasena, Panos Papalambros, and Pierre Goovaerts. Exploration of metamodeling sampling criteria for constrained global optimization. *Engineering Optimization*, 34:263–278, 2002.

Matthias Schonlau. *Computer Experiments and Global Optimization*. PhD thesis, University of Waterloo, 1997.

Matthias Schonlau, William J Welch, and Donald R Jones. Global versus local search in constrained optimization of computer models. *Lecture Notes-Monograph Series*, pages 11–25, 1998.

Matthias W Seeger. Bayesian inference and optimal design for the sparse linear model. *Journal of Machine Learning Research*, 9:759–813, 2008.

Alexander Shapiro, Darinka Dentcheva, and Andrzej Ruszczyński. *Lectures on stochastic pro-*

gramming: modeling and theory. Society for Industrial and Applied Mathematics, Philadelphia, USA, 2009.

Jasper Snoek. *Bayesian Optimization and Semiparametric Models with Applications to Assistive Technology*. PhD thesis, University of Toronto, Toronto, Canada, 2013.

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.

Jasper Snoek, Kevin Swersky, Richard S. Zemel, and Ryan P. Adams. Input warping for Bayesian optimization of non-stationary functions. In *International Conference on Machine Learning*, pages 1674–1682, 2014.

Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Md. Mostofa Ali Patwary, Prabhat, and Ryan P. Adams. Scalable Bayesian optimization using deep neural networks, 2015. arXiv:1502:05700 [stat.ML].

Andras Sobester, Stephen J. Leary, and Andy J. Keane. On the design of optimization strategies based on global response surface approximation models. *Journal of Global Optimization*, 33: 31–59, 2005.

Niranjan Srinivas, Andreas Krause, Sham Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: no regret and experimental design. In *International Conference on Machine Learning*, pages 1015–1022, 2010.

Richard S. Sutton and Andrew G. Barto. *Introduction to reinforcement learning*. MIT Press, 1998.

Krister Svanberg. A class of globally convergent optimization methods based on conservative convex separable approximations. *SIAM Journal on Optimization*, 12:555–573, 2002.

Kevin Swersky, Jasper Snoek, and Ryan P. Adams. Multi-task Bayesian optimization. In *Advances in Neural Information Processing Systems*, pages 2004–2012, 2013.

Matthew Tesch, Jeff G. Schneider, and Howie Choset. Expensive function optimization with stochastic binary outcomes. In *International Conference on Machine Learning*, pages 1283–1291, 2013.

Madeleine B. Thompson and Radford M. Neal. Slice sampling with adaptive multivariate steps: the shrinking-rank method. In *Joint Statistical Meetings*, pages 3890–3896, 2010.

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

Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Auto-WEKA: Com-

bined selection and hyperparameter optimization of classification algorithms. In *International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 847–855, 2013.

Julien Villemonteix, Emmanuel Vázquez, and Eric Walter. An informational approach to the global optimization of expensive-to-evaluate functions. *Journal of Global Optimization*, 44(4): 509–534, 2009.

Ziyu Wang, Masrour Zoghi, Frank Hutter, David Matheson, and Nando de Freitas. Bayesian optimization in high dimensions via random embeddings. In *International Joint Conferences on Artificial Intelligence*, pages 1778–1784, 2013.

Yelp, Inc. Metric Optimization Engine (MOE), GitHub repository. URL `http://github.com/Yelp/MOE`.

Marcela Zuluaga, Andreas Krause, Guillaume Sergent, and Markus Püschel. Active learning for multi-objective optimization. In *International Conference on Machine Learning*, pages 462–470, 2013.