Automating Inference, Learning, and Design using Probabilistic Programming

Tom Rainforth

Wolfson College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Trinity 2017

Abstract

Imagine a world where computational simulations can be inverted as easily as running them forwards, where data can be used to refine models automatically, and where the only expertise one needs to carry out powerful statistical analysis is a basic proficiency in scientific coding. Creating such a world is the ambitious long-term aim of *probabilistic programming*.

The bottleneck for improving the probabilistic models, or simulators, used throughout the quantitative sciences, is often not an ability to devise better models conceptually, but a lack of expertise, time, or resources to realize such innovations. Probabilistic programming systems (PPSs) help alleviate this bottleneck by providing an expressive and accessible modeling framework, then automating the required computation to draw inferences from the model, for example finding the model parameters likely to give rise to a certain output. By decoupling model specification and inference, PPSs streamline the process of developing and drawing inferences from new models, while opening up powerful statistical methods to non-experts. Many systems further provide the flexibility to write new and exciting models which would be hard, or even impossible, to convey using conventional statistical frameworks.

The central goal of this thesis is to improve and extend PPSs. In particular, we will make advancements to the underlying inference engines and increase the range of problems which can be tackled. For example, we will extend PPSs to a mixed inference-optimization framework, thereby providing automation of tasks such as model learning and engineering design. Meanwhile, we make inroads into constructing systems for automating adaptive sequential design problems, providing potential applications across the sciences. Furthermore, the contributions of the work reach far beyond probabilistic programming, as achieving our goal will require us to make advancements in a number of related fields such as particle Markov chain Monte Carlo methods, Bayesian optimization, and Monte Carlo fundamentals.

Acknowledgements

I would first like to thank my partner Sophie for her support and understanding through this process and for all the sacrifices she has made on my behalf. I would similarly like to thank my family for their unwavering support and making me who I am today. I would like to thank Frank Wood, Jan-Willem van de Meent, and Brooks Paige for their guidance and support, both academic and personal. I owe them all, particularly Frank and Jan-Willem, a huge debt of gratitude for teaching me most of what I know, nurturing me from an arrogant fool who asked questions like *Metropolis what?* to an even more arrogant fool who thinks he knows everything. This work would have been a terrible failure without them. I would like to thank my friends and colleagues Atılım Güneş Baydin, Rob Cornish, Piotr Czaban, Neil Dhir, Jack Fitzsimons, Adam Goliński, Bradley Gram-Hansen, Max Igl, David Janz, Tom Jin, Tuan Anh Le, Mario Lezcano, Aravindh Mahendran, David Martínez Rubio, Siddharth N, Nantas Nardelli, Michael Osborne, Nick Palmius, Yura Perov, David Tolpin, Andrea Vedaldi, Andrew Warrington, Stefan Webb, Hongseok Yang, Yuan Zhou, and Rob Zinkov, for making this time in Oxford some of the happiest of my life. I would like to thank my previously unmentioned coauthors Arnaud Doucet, Fredrik Lindsten, Christian A. Naesseth, and Benjamin Vincent, for being an absolute pleasure to work with. Finally, I would like to thank BP for providing the funding for this research.

Much of the novel material in this thesis is from coauthored work and it would be fabrication to claim it all as my own. In particular, a lot of the work is based around the probabilistic programming system *Anglican*, for which all the credit must go to Frank Wood, Jan-Willem van de Meent, and David Tolpin. Elsewhere, citations are made to the original coauthored papers upon which this work is based, for which credit must go to all my coauthors.

Contents

1 Introduction

How come a dog is able to catch a frisbee in mid-air? How come a batsman can instinctively predict the flight of a cricket ball, moving at over 100km/h, sufficiently accurately and quickly to hit it when there is not even time to consciously make a prediction? Clearly, neither can be based on a deep explicit knowledge of the laws of physics or some hard-coded model for the movement of objects; we are not even born with the knowledge that unsupported objects will fall down [Baillargeon, 2002]. The only reasonable explanation for these abilities is that the batsmen and the dog have *learned from experience*. We do not have all the knowledge we require to survive from birth, but we are born with the ability to learn and adapt, making observations about the world around us and using these to refine our cognitive models for everything from the laws of physics to social interaction. Classically the scientific method has relied on human interpretation of the world to formulate explicit models to explain our internal intuitions, which we then test through experimentation. However, even as a whole scientific society, our models are often terribly inferior to the subconscious models of animals and children, such as for most tasks revolving around social interaction. This leads one to ask, is there something fundamentally wrong with this hand-crafted modeling approach? Is there another approach that better mimics the way humans themselves learn?

Machine learning is an appealing alternative, and often complementary, approach that focuses on constructing algorithms and systems that can adapt, or learn, from data in order to make predictions that have not been explicitly programmed. This is exciting not only because of the potential it brings to automate and improve a wide array of computational tasks, but because it allows us to design systems capable of going beyond the boundaries of human understanding, reasoning about and making predictions for tasks we cannot solve directly ourselves. As a field, machine learning is very wide ranging, straddling computer science, statistics, engineering, and beyond. It is perhaps most closely related to the field of computational statistics, differing predominantly in its emphasis on prediction rather than understanding. Despite the current hype around the field, most of the core ideas have existed for some time, often under the guise of pattern recognition, artificial intelligence, or computational statistics. Nonetheless, the explosion

in the availability of data and in computational processing power in recent years has led to a surge of interest in machine learning by academia and industry alike, particularly in its application to real world problems. This interest alone is enough to forgive the hype, as the spotlight is not only driving the machine learning community itself forward, but helping identify huge numbers of applications where existing techniques can be transferred to fantastic effect. From autonomous vehicles [Lefèvre et al., 2014], to speech recognition [Jurafsky and Martin, 2014], and designing new drugs [Burbidge et al., 2001], machine learning is rapidly becoming a crucial component in many technological and scientific advancements.

In many machine learning applications, it is essential to use a principled *probabilistic* approach [Ghahramani, 2015], incorporating uncertainty and utilizing all the information at hand, particularly when data is scarce. The *Bayesian paradigm* provides an excellent basis upon which to do this: an area specialist constructs a probabilistic model for data generation, conditions this on the actual observations received, and, using Bayes' rule, receives an updated model incorporating this information. This allows information from both existing expertise and data to be combined in a statistically rigorous fashion. As such, it allows us to use machine learning to complement the conventional scientific approach, rather than directly replacing it: we can construct models in a similar way to that which is already done, but then improve and refine these using data.

Unfortunately, there are two key stumbling blocks that often make it difficult for this idealized view of the Bayesian machine learning approach to be realized in practice. Firstly, a process known as *Bayesian inference* is required to solve the specified problems. This is typically a challenging task, closely related to integration, which is often computationally intensive to solve. Furthermore, it often requires significant statistical expertise to implement effectively, creating a substantial barrier to entry. Secondly, it can be challenging to specify models that are true to the assumptions the user wishes to make and the prior information available. It can again require statistical expertise to abstract application specific knowledge to a valid statistical model. Furthermore, assumptions are often made in the interest of the tractability of inference, rather than the fidelity of the model. Perhaps because of these drawbacks, there is often a reliance on off-the-shelf solutions, even when these models are somewhat inappropriate for the task at hand.

Probabilistic programming systems (PPS) [Goodman et al., 2008b] are an attempt to overcome this dichotomy between the Bayesian ideal and common practice. Their core philosophy is to decouple model specification and inference, the former corresponding to the user-specified program code, composing of a generative model and statements for conditioning on data, and the latter to an inference engine capable of operating on arbitrary programs. This abstraction barrier

allows users with domain specific knowledge to write models naturally, as if they were writing a simulator, without worrying about the inference, which becomes the job of the developer. Informally one can think of PPS as operating as inverse probability engines, outputting the conditional probability distribution implied by the generative model coupled with the observed data. Removing the need for users to worry about the required inference significantly reduces the burden of developing new models and makes effective statistical methods accessible to non-experts. From a developer's perspective, the abstraction can also aid in the design and testing of new inference algorithms. Furthermore, the availability of the target source code, with known semantics, opens up many opportunities for new methods that would not otherwise be possible.

The underlying theme of this thesis is improving and extending probabilistic programs. However, doing this will involve making a number of noticeable advancements in distinct research areas such as particle Markov chain Monte Carlo methods [Andrieu et al., 2010], Bayesian optimization [Shahriari et al., 2016b], and Monte Carlo fundamentals [Owen, 2013]. Along with the direct merit of these advancements, our general aims from a probabilistic programming perspective can be broken down into two themes: improving the underlying inference engines of PPSs and increasing the range of problems they can tackle.

Improving PPS inference engines is necessary both to increase their efficiency and, perhaps more importantly, because the limits on the complexity of models that can be tackled by a system are predominantly dictated by the effectiveness of its inference engine(s). Though existing PPSs provide an effective means of performing inference in simple models or those with easily exploitable structures, they still tend to be some way off what is achievable using bespoke inference methods. They are even further away from what would be required to achieve our lofty ambitions of providing automated inference for the vast array of stochastic simulators used by the scientific community. Improving inference engines is, therefore, essential for making PPSs practical and applicable to a wider array of models. Our main contribution to this long term goal is in introducing the *interacting particle Markov chain Monte Carlo* algorithm [Rainforth et al., 2016c], which represents a state-of-the-art inference method suitable for PPS and, arguably, the current go-to algorithm for the PPS Anglican [Wood et al., 2014].

Our second aim, of increasing the range of problems that can be tackled by PPSs, is all about going beyond the standard Bayesian inference setting. As powerful as the Bayesian paradigm is, it is far from the only tool in the machine learning arsenal and there are many problems that fall outside of its framework. For example, by extending PPSs to a more general mixed inference-optimization framework using BOPP [Rainforth et al., 2016b], we open the door

to a number of fascinating opportunities such as learning models themselves or constructing principled, probabilistic, engineering design pipelines that explicitly incorporate the uncertainty in the task at hand. We will also investigate the opportunities and pitfalls of *nesting* inference problems and nesting estimation more generally [Rainforth et al., 2017a]. Such nesting is essential for expressing a wide array of problems from experimental design [Chaloner and Verdinelli, 1995] to theory-of-mind [Stuhlmüller and Goodman, 2014]. PPSs provide a highly convenient means of encoding such problems, but previous results confirming the statistical validity of doing so remain substantially lacking. We provide many of the required foundational theoretical results and demonstrate their importance by giving guidelines and precautions for carrying out such nesting, both in the specific probabilistic programming context [Rainforth, 2018] and more generally [Rainforth et al., 2017a], highlighting potential shortcomings in how this is dealt with by existing systems. We also use our findings to start taking steps towards potentially building a general purpose system for automated adaptive design, realizing this potential in the restricted setting of psychological trials investigating the impact of delays and uncertainty on the subjective values people place on rewards [Vincent and Rainforth, 2017].

1.1 Thesis Aims and Layout

This thesis contains a mixture of pedagogical material, literature review, and cutting-edge research. Although there will inevitably be gaps, the aim is to take the reader from the fundamentals of probability and machine learning, all the way through to highly advanced topics such as designing automatic mixed inference-optimization engines for probabilistic programming languages. In addition to trying to make the work as self-contained as is realistically possible, there are three key rationales why we have decided to start from the fundamentals. Firstly, it is our aim to make the work as accessible as possible and hopefully provide a useful guide to those new to the field. Probabilistic programming, and machine learning more generally, draws researchers from an exceptionally diverse pool of backgrounds and what is rudimentary to, for example, the statistics community is often bewildering to the programming languages community and vice-versa. Secondly, because probabilistic programing, and in particular the more advanced content of this thesis, goes beyond conventional notions of Bayesian modeling, a thorough understanding of the core concepts is essential. Thirdly, because machine learning moves so quickly, or perhaps simply because of the nature of research itself, it can be easy to become fixated on a small area of research and lose sight of the big picture. For good research it is important to not only question one's contribution to a particular subfield, but also the justification

for that subfield itself. With the current hype surrounding machine learning and, in particular, deep learning, returning to these fundamental principles and core concepts is more important than ever so that we can assert the right approach for a particular problem and appreciate what we can realistically hope to achieve with machine learning.

More precisely, the thesis can be broken down as follows. Chapters 2, 3, and 5 are all explicitly pedagogical and cover fundamentals that should be understood in depth by anybody researching or applying Bayesian modeling and arguably machine learning more generally. Further breaking these down, Chapter 2 provides a short, but hopefully gentle, primer on probability theory and thereby lays out the foundations upon which all probabilistic machine learning is based. Chapter 3 introduces the Bayesian approach to machine learning upon which most of this work is based, giving motivation for the framework, but also a critique exposing some of its weaknesses. Chapter 5 introduces the problem of *Bayesian inference*, required to solve all Bayesian modeling problems, focusing on a *Monte Carlo* approach, providing, in particular, a detailed introduction to some of the foundational methods.

Chapter 4 provides an introduction to probabilistic programming from the perspective of writing models. Though its earlier sections are still predominantly pedagogical, much of the core notation for the rest of the thesis is laid out in its latter sections, some of which is somewhat distinct to that conventionally used in the literature. Chapter 6 considers a more advanced class of Bayesian inference methods, namely sequential Monte Carlo [Doucet and Johansen, 2009] and particle Markov chain Monte Carlo methods [Andrieu et al., 2010], building up to our recent contribution to the literature [Rainforth et al., 2016c]. Chapter 7 though not explicitly new research, being in particular an ode to the work of Tolpin et al. [2016], still constitutes relatively advanced and cutting-edge material in the form of the intricate workings of a probabilistic programming system back-end. Chapter 8 is a return to slightly more pedagogical material in the form of an introduction to Bayesian optimization.

Chapters 9, 10, and 11 all outline new contributions to the literature both as presented in recent publications of ours [Rainforth et al., 2016b, 2017a; Rainforth, 2018; Vincent and Rainforth, 2017] and newly introduced material for this thesis. Chapter 9 introduces a mixed inference-optimization framework for PPSs using a mixture of Bayesian optimization, code transformations, and existing inference engines. Chapter 10 considers the statistical implications of nesting Monte Carlo estimators and the application of these results to the nesting of probabilistic programs. Chapter 11 outlines a method for automating adaptive experiments for a certain class of psychological trials and takes steps towards designing a general purpose PPS for experimental design.

2

A Brief Introduction to Probability Theory

Before going into the main content of the paper, we first provide a quick primer on probability theory, giving some essential background and outlining conventions that we will use throughout the thesis. Readers familiar with the differences between a probability and a probability density and between a random variable and an outcome may want to skip to this chapter, referring back as needed for clarification on any conventions undertaken. Others will hopefully find it to be a gentle introduction to the key concepts that will be needed to be understood to follow the rest of the thesis. Notation has been chosen with accessibility as the primary aim and we will avoid the use measure theory where possible. Nonetheless, a measure theoretic approach to probability is essential for a more rigorous understanding and we refer the interested reader to Durrett [2010].

2.1 Random Variables, Outcomes, and Events

A *random variable* is a variable whose realization is currently unknown, such that it can take on multiple different values or *outcomes*.[1] A set of one or more outcomes is known as an *event*. For example, if we roll a fair six-sided dice then the result of the roll is a random variable while rolling a 4 is both a possible outcome and a possible event. Rolling a number greater or equal to 5, on the other hand, is a possible event but not a possible outcome: it is a set of two individual outcomes, namely rolling a 5 and rolling a 6. Outcomes are *mutually exclusive*, that is, it is not possible for two separate outcomes to occur for a particular trial, e.g. we cannot roll both a 2 and 4 with a single throw. Events, on the other hand, are not. For example, it is possible for both the events that we roll and even number and we roll a number greater than 3 to occur.

2.2 Probabilities

A *probability* is the chance of an event occurring. For example, if we denote the output of our dice roll as X, then we can say that $P(X = 4) = 1/6$ or that $P(X \leq 3) = 0.5$. Here $X = 4$ and $X \leq 3$ are events for the random variable X with probabilities of $1/6$ and 0.5 respectively. A probability of 0 indicates that an event has no chance of happening, for example

[1] In a more formal, measure-theoretic framework, outcomes are points in *sample space* and random variables are measurable functions from outcomes to a measurable space.

the probability that we role an 8, while a probability of 1 indicates it is certain to happen, for example, the probability that we roll a positive number. All probabilities must thus lie between 0 and 1 (inclusively). A *distribution* of a random variable provides the probabilities of each possible outcome for that random variable occurring.

Though, we will regularly use the shorthand $P(x)$ to denote the probability of the event $P(X = x)$, we reiterate the important distinction between the random variable X and the outcome x: the former has an unknown value (e.g. the result of the dice roll) and the latter is a fixed possible realization of the random variable (e.g. rolling a 4). All the same, we will at times be intentionally carefree about delineating between random variables and outcomes, except for when the distinction is explicitly necessary.

Somewhat surprisingly, there are two competing (and sometimes incompatible) interpretations of probability. The frequentist interpretation of probability is that it is the average proportion of the time an event will occur if a trial is repeated infinitely many times. The Bayesian interpretation of probability is that it is the subjective belief that an event will occur in the process of incomplete information. Both viewpoints have strengths and weaknesses and we will avoid being drawn into one of the biggest debates in science, noting only that the philosophical differences between the two are typically completely detached from the practical differences between the resulting machine learning or statistics methods (see Section 3.4), despite these philosophical differences all too often being used to argue the superiority of the resultant algorithms [Gelman et al., 2011; Steinhardt, 2012].

2.3 Conditioning and Independence

A *conditional probability* is the probability of an event given that another event has occurred. For example, the conditional probability that we roll a 4 with a dice given that we have rolled a 3 or higher is $P(X = 4|X \geq 3) = 0.25$. More typically, we will condition upon events that are separate but correlated to the event we care about. For example, the probability of dying of lung cancer is higher if you smoke. The process of updating a probability using the information from another event is known as conditioning on that event. For example, one can condition the probability that a football team will win the league on the results from their first few games.

Events are *independent* if the occurrence of one event does not affect the probability of the occurrence of the other event. Similarly, random variables are independent if the outcome of one random variable does not affect the distribution of the other. Independence of random variables indicates the probability of each variable is the same as the conditional probability given the other

variable, i.e. if X and Y are independent, $P(X = x) = P(X = x|Y = y)$ for all possible y. Note that independence does not necessarily carry over when adding or removing a conditioning: if X and Y are independent, this does not necessarily mean that $P(X = x|A) = P(X = x|A, Y = y)$ for some event A. For example, the probability that the next driver to pass a speed camera is speeding and that the speed camera is malfunctioning can be reasonably presumed to be independent. However, conditioned on the event that the speed camera is triggered, the two are clearly not independent as if the camera is working, this would indicate that the driver is speeding. If $P(X = x|A) = P(X = x|A, Y = y)$ holds, then X and Y are known as conditionally dependent given A. In the same way that independence does not imply conditional independence, conditional independence does not imply non-conditional independence.

2.4 The Laws of Probability

Though not technically axiomatic, the mathematical laws of probability can be summarized by the *product rule* and the *sum rule*. Remarkably, almost all of probability stems from these two simple rules. The product rule states that the probability of two events occurring is the probability of one of the events occurring times the conditional probability of the other event happening given the first event happened, namely

$$P(A, B) := P(A \cap B) = P(A|B)P(B) = P(B|A)P(A) \tag{2.1}$$

where we have introduced $P(A, B)$ as a shorthand for the probability that both the events A and B occur. An immediate consequence of the product rule is Bayes' rule,

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}, \tag{2.2}$$

which we will to return at length in Section 3.2. Another is that for independent random variables, the joint distribution is the product of the individual probabilities: $P(A, B) = P(A)P(B)$.

The sum rule has a number of different representations, the most general of which is that the probability that either A or B occurs, $P(A \cup B)$, is given by

$$P(A \cup B) = P(A) + P(B) - P(A, B). \tag{2.3}$$

The intuition of the sum rule is perhaps easiest to see by considering that

$$P(B) - P(A, B) = P(B)(1 - P(A|B)) = P(B, \neg A)$$

is the probability of B and not A. Now $A \cup B$ can only occur if A occurs or if B occurs and not A. As it is not possible for both these events to occur, the probability of either event must

be the sum of the probability of each separate event, leading to (2.3). There are a number of immediate consequences of the sum rule. For example, if A and B are mutually exclusive then $P(A \cup B) = P(A) + P(B)$. As outcomes are mutually exclusive, it follows from the sum rule and the axioms of probability that the sum of the probabilities for each possible outcome is equal to 1. We can also use this to define the concept of *marginalizing* out a random variable Y as

$$P(X = x) = \sum_i P(X = x, Y = y_i) \tag{2.4}$$

where the sum is over all the possible outcomes of Y. Here $P(X = x)$ is known as the *marginal probability* of X and $P(X = x, Y = y)$ as the *joint probability* of X and Y.

Conditional probabilities follow the same key results as unconditional probabilities, but it should be noted that they do not define probability distributions over the conditioning term. For example, $P(A|B)$ is a probability distribution over A with all the corresponding requirements, but is not a distribution over B. Therefore, for example, it is possible to have $\sum_i P(A|B = b_i) > 1$. We instead refer to $P(A|B)$ as the *likelihood* of B, given the occurrence of event A.

2.5 Probability Densities

Thus far we have presumed that our random variables are discrete, i.e. that there is some fixed number of possible outcomes.[2] Things get somewhat more complicated if our variables are continuous. Consider for example the probability that a runner takes exactly π (i.e. $3.14159265\ldots$) hours to run a marathon $P(X = \pi)$. Clearly, the probability of this particular event is zero, $P(X = \pi) = 0$, as is the probability of the runner taking any other exact time to complete the race: we have an infinite number of possible outcomes, each with zero probability (presuming the runner finishes the race). Thankfully, the notion of an event that we previously introduced comes to our rescue. For example, the event that the runner takes between 3 and 4 hours has non-zero probability: $P(3 \leq X \leq 4) \neq 0$. Here our event includes an infinite number of possible outcomes and even though each individual outcome had zero probability, the combination of *uncountably infinitely* many such outcomes need not also have zero probability. To more usefully characterize probability in such cases, we can define a *probability density function* which reflects the relative probability of areas of the space of outcomes. We can informally define this by considering the probability of being some small area of the space of size δx. Presuming that the probability density $p_X(x)$ is roughly constant within our small area, we can say that

[2] Technically speaking, discrete random variables can also take on a *countably infinite* number of values, e.g. the Poisson distribution is defined over $0, 1, 2, \ldots, \infty$. However, this countable infinity is much smaller than the *uncountably infinite* number of possible outcomes for continuous random variables.

$p_X(x)\delta x \approx P(x \leq X < x + \delta x)$, thus giving the informal definition $p_X(x) = \lim_{\delta \to 0} \frac{P(x \leq X < x+\delta x)}{\delta x}$. More precisely we can define the probability density as satisfying

$$P(X \in \mathcal{A}) = \int_{x \in \mathcal{A}} p_X(x) dx \tag{2.5}$$

where $X \in \mathcal{A}$ means the event that X is in $\mathcal{A}$. We can similarly define the *cumulative distribution function* $P(X \leq x)$, which is the probability that X is less than equal to the outcome x

$$P(X \leq x) = \int_{-\infty}^{x} p_X(u) du, \tag{2.6}$$

where u is a dummy variable. The fundamental laws of probability discussed in the last section apply equally well to probability densities, replacing summations with integrals as and when required.

In the rest of the thesis will drop the notation $p_X(x)$, using simply $p(x)$ instead. The main rationale for this is that we will regularly use probability density functions that we do not actually sample from. For example, in importance sampling, we will sample from one distribution but evaluate its density under another. In these scenarios, it may not be possible to link a random variable to each density. We will instead make it explicit what distribution a random variable is drawn from using the notation $X \sim p(x)$. However, we will regularly be carefree about distinguishing between random variables and outcomes by using loose notations such as $x \sim p(x)$ when the delineation is not necessary in the context.

2.6 Measures

Consider now if there is also a probability that the runner does not finish the race which we denote as the outcome $X = \infty$. As we have thus-far introduced them, neither the concept of a probability or a probability density seem to be suitable for this case: every outcome other than $X = \infty$ has zero probability, but $X = \infty$ seems to have infinite probability density. To solve this conundrum we have to briefly break our promise to avoid measure theory. A *measure* can be thought of as something that assigns a size to a set of objects. *Probability measures* assign probabilities to events, remembering that events represent sets of outcomes, and thus are used to define a more formal notion of probability that we have previously discussed. The measure assigned to an event including all possible outcomes is thus 1, while the measure assigned to the empty set is 0.

We can generalize the concept of a probability density to arbitrary random variables by formalizing its definition as being with respect to an appropriate *reference measure*. Somewhat confusingly, this reference measure is typically not a probability measure. Consider the case of the continuous densities examined in Section 2.5. Here we have implicitly used the *Lebesgue*

measure as the reference measure, which corresponds to the standard Euclidean notion of size, coinciding with the concepts of length, area, and volume in 1, 2, and 3 dimensions respectively. In (2.5) then dx indicated integration with respect to a Lebesgue measure, with $\int_{x\in\mathcal{A}} dx$ being equal to the hypervolume of $\mathcal{A}$ (e.g. area of $\mathcal{A}$ in two dimensions). Our reference measure is, therefore, clearly not a probability measure as $\int_{x\in\mathbb{R}} dx = \infty$. Our probability measure here can be informally thought of as $p(x)dx$, so that $\int_{x\in\mathcal{A}} p(x)dx = P(x \in \mathcal{A})$.[3] In the discrete cases, we can define a probability density $p(x) = P(X = x)$ by using the notion of a *counting measure* for reference, which simply counts the number of outcomes which lead to a particular event. Note that we were not free to choose any arbitrary measure for any given random variable. We cannot use a counting measure as reference for continuous random variables or the Lebesgue measure for discrete random variables because, for example, the Lebesgue measure would assign zero measure to all possible events for the latter. Nonetheless, the reference measure we use is not necessarily unique either. For example, we could apply a constant c scaling the reference measure and then scale the density by $1/c$. For notional convenience, we will refer to dx (or equivalent) as our reference measure elsewhere in the thesis.

For the example where the runner might not finish, we can use a *mixed measure*. Perhaps the easiest way to think about this is to think about the runner's time X as being a deterministic function of two random variables: the first a discrete random variable $Y \in \{0, 1\}$ that dictates where the runner finishes ($Y = 1$) or not ($Y = 0$) and the second a continuous random variable $Z \in \mathbb{R}^+$ that specifies a distribution over run times assuming the runner finishes (such that Z always exists but only equals X if the runner finishes). Now X is a function of Y and Z and we can define the probability of the event $X \in \mathcal{A}$ as follows

$$\begin{aligned} P(X \in \mathcal{A}) &= \sum_{y\in\{0,1\}} \int_{z\in\mathbb{R}} P(Y = y)p(z)\mathbb{I}(X(y, z) \in \mathcal{A})dz \\ &= P(Y = 0)\mathbb{I}(\infty \in \mathcal{A}) + P(Y = 1) \int_{z\in\mathcal{A}\setminus\infty} p(z)dz \end{aligned}$$

where $\mathbb{I}(\cdot)$ is the identity function evaluating to 1 if its input holds true and 0 otherwise, and we have used the independence between Y and Z. By definition of a probability density, we also have that $P(X \in \mathcal{A}) = \int_{x\in\mathcal{A}} p(x)d\mu(x)$ for density $p(x)$ with respect to measure $d\mu(x)$, where we switched notation from dx to $d\mu(x)$ to express the fact that the measure now depends explicitly on the value of x, i.e. our measure is non-stationary in x. For $x \neq \infty$ then it is natural for $d\mu(x)$ to correspond to the Lebesgue measure, noting $\int_{x\in\mathbb{R}\setminus\infty} p(x)d\mu(x) = 1 - P(X = \infty)$.

[3] More formally, the density is derived from the probability measure and reference measure rather than the other way around: it is the Radon-Nikodym derivative of the probability measure with respect to the reference measure.

For $x = \infty$ it is natural for $d\mu(x)$ to correspond to some scaling of the counting measure. For discrete distributions, it is natural to use the counting measure exactly (i.e. set the scaling to one), but here this could lead to a slightly misleading density given that $X = \infty$ is infinitely times more probable than any other X. For example, this could mean that $\arg\max_x p(x) \neq \infty$ which could quickly lead to confusion. A more illustrative choice might use the counting measure scaled by an arbitrarily small value (such that the density is scaled by large value), though strictly speaking this scaling should not be 0.

For most applications,[4] the choice between valid reference measures will turn out to be inconsequential (provided we are consistent in our choice when considering multiple densities), as it does not affect the distribution of random variables themselves or affect things such as expectations (see Section 2.7). The key point we wish to instead make is that, regardless of variable type, we can (implicitly) always define a probability density function.[5] The generalization also allows us to put probability distributions of variables that are neither discrete nor continuous type. For example, we might want to put a distribution over the space of strings. Because of this generalization, we will talk mostly in terms of probability densities throughout this thesis, even though variables may often be discrete (for which the density will represent a probability).

We finish the section by considering the relationship between random variables which are deterministic functions of one another. This important case is known as a *change of variables*. Imagine that a random variable $Y = g(X)$ is a deterministic function of another random variable X. Given a probability density function for X, we can define a probability density function on Y using

$$p(y)dy = p(x)dx = p(g^{-1}(y))dg^{-1}(y) \tag{2.7}$$

where $p(y)$ and $p(x)$ are the respective probability densities for Y and X with measures dx and dy, the latter of which is known as a *push-forward* measure. Rearranging we see that, for one-dimensional problems,

$$p(y) = \left|\frac{dg^{-1}(y))}{dy}\right| p(g^{-1}(y)). \tag{2.8}$$

For the multidimensional case, the derivative is replaced by the determinant of the Jacobian for the inverse mapping. Note that by (2.5), changing variables does not change the value of actual probabilities or exceptions (see Section 2.7). However, (2.7) still has the important consequence that the optimum of a probability distribution depends on the parameterization.

[4] An important exception to this is optimization.

[5] Well, almost always. But we are not going to open that can of worms, see e.g. Durrett [2010].

For example, parameterizing the same model with either X or $\log X$ will lead to a different most likely value of the parameter x^*, i.e., in general,

$$x^* = \arg\max_x p(x) \neq g^{-1}\left(\arg\max_{g(x)} p(g(x))\right). \tag{2.9}$$

2.7 Expectations and Variances

The *expected value* $\mathbb{E}[X]$, or mean, of a random variable X is the average value that the variable will take if an infinite number of independent draws are made. Its definition is easiest to convey using probability density notation as follows

$$\mathbb{E}[X] = \int xp(x)dx. \tag{2.10}$$

In the discrete case this leads to $\mathbb{E}[X] = \sum_i x_i P(X = x_i)$. Because expectations are defined by a random variable (rather than a density), they average over all the contained randomness, e.g. $\mathbb{E}[f(X,Y)] = \iint f(x,y)p(x,y)dxdy$. However, if we wish to average only with respect to part of the randomness in a system, we can instead use a conditional expectation for example

$$\mathbb{E}[f(X,Y)|Y = y] = \int f(x,y)p(x|y)dx, \tag{2.11}$$

for which we will sometimes use the shorthand $\mathbb{E}[f(X,Y)|Y]$. It will also sometimes be convenient to implicitly define the random variable and conditioning for an expectation for which we use the slightly loose notation

$$\mathbb{E}_{p(x|y)}\left[f(x,y,z)\right] = \int f(x,y,z)p(x|y)dx, \tag{2.12}$$

where we have implicitly defined the random variables $Y \sim p(y)$ and $X \sim p(x|Y = y)$, we are calculating $\mathbb{E}\left[f(X,Y,z)|Y = y\right]$, and this resulting expectation is a function of z (which is treated as deterministic variable). One can informally think about this as being the expectation of $f(x,y,z)$ with respect to $p(x|y)dx$: our expectation is only over the randomness associated with drawing from $p(x|y)$.

Denoting the mean of a random variable X as $\mu = \mathbb{E}[X]$, the *variance* of X is defined using any one of the following equivalent forms

$$\mathrm{Var}(X) = \mathbb{E}\left[(X-\mu)^2\right] = \int (x-\mu)^2 p(x)dx = \mathbb{E}[X^2] - \mu^2 = \int x^2 p(x)dx - \mu^2. \tag{2.13}$$

In other words, it is the average squared distance of a variable from its mean. Its square root, the *standard deviation*, informally forms an estimate of the average amount of variation of the variable from its mean and has units which are the same as the data. We will use the same notational conventions as for expectations when defining variances.

The variance is a particular case of the more general concept of a covariance between two random variables X and Y. Defining $\mu_X = \mathbb{E}[X]$ and $\mu_Y = \mathbb{E}[Y]$ then the covariance is defined by any one of the following equivalent forms

$$\begin{aligned}\mathrm{Cov}(X,Y) &= \mathbb{E}\left[(X-\mu_X)(Y-\mu_Y)\right] = \iint (x-\mu_X)(y-\mu_Y)p(x,y)dxdy \\ &= \mathbb{E}\left[XY\right] - \mathbb{E}\left[X\right]\mathbb{E}\left[Y\right] = \iint xyp(x,y)dxdy - \left(\int xp(x)dx\right)\left(\int yp(y)dy\right). \quad (2.14)\end{aligned}$$

The covariance between two variables measures the joint variability of two random variables. It is perhaps easiest to interpret through the definition of correlation (or more specifically, Pearson's correlation coefficient) which is the correlation scaled by the standard deviation of each of the variables

$$\mathrm{Corr}(X,Y) = \frac{\mathrm{Cov}(X,Y)}{\sqrt{\mathrm{Var}(X)\mathrm{Var}(Y)}}. \quad (2.15)$$

The correlation between two variables is always in the range $[-1, 1]$. Positive correlations indicate that when one variable is relatively larger, the other variable also tends to be larger. The higher the correlation, the more strongly this relationship holds: if the correlation is 1 then one variable is *linearly* dependent on the other. The same holds for negative correlations except that when one variable increases, the other tends to decrease. Independent variables have a correlation (and thus a covariance) of zero, though the reciprocal is not necessarily true: variables with zero correlation need not be independent. Note that correlation is not causation.

3
Probabilistic Machine Learning

In this chapter, we will provide a high-level introduction to some of the core approaches to machine learning. We will distinguish between discriminative and generative approaches, outlining some of the key features that indicate when problems are more suited to one approach or the other. Our attention then settles on probabilistic generative approaches, which will be the main focus of this thesis. We will explain how the *Bayesian paradigm* provides a powerful framework for generative machine learning that allows us to combine data with existing expertise. We will then go on to show how *graphical models* can be used as a convenient framework to express Bayesian models and extract important features. We continue by introducing the main counterpart to the Bayesian approach – frequentist approaches – and present arguments for why neither alone provides the full story. In particular, we will outline the fundamental underlying assumptions made by each approach and explain why the differing suitability of these assumptions to different tasks means that both are essential tools in the machine learning arsenal, with many problems requiring both Bayesian and frequentist elements in their analysis. Though much of the focus of this thesis will be on Bayesian approaches, understanding their limitations is essential for understanding when the methods we discuss should be used and, critically, when they should not. We finish the chapter by discussing some of the key practical challenges for Bayesian modeling.

3.1 Discriminative vs Generative Machine Learning

In some machine learning applications, huge quantities of data are available that dwarf the information that can be provided from human expertise. In such situations, the main challenge is in processing and extracting all the desired information from the data to form a useful characterization, typically an artifact providing accurate predictions at previous unseen inputs. Such problems are typically suited to *discriminative* machine learning approaches [Breiman et al., 2001; Vapnik, 1998], such as neural networks [Rumelhart et al., 1986; Bishop, 1995], support vector machines [Cortes and Vapnik, 1995; Schölkopf and Smola, 2002], and decision tree ensembles [Breiman, 2001; Rainforth and Wood, 2015]. Discriminative machine learning approaches focus on directly learning a predictive model: given training data $\mathcal{D} = \{x_n, y_n\}_{n=1}^N$ they learn a parametrized mapping f_θ from the inputs $x \in \mathcal{X}$ to the outputs $y \in \mathcal{Y}$ that can

be used directly to make predictions for new inputs $\tilde{x} \notin \{x_n\}_{n=1}^N$. *Training* uses the data $\mathcal{D}$ to estimate optimal values of the parameters θ^*. *Prediction* at a new input $\tilde{x}$ involves applying the mapping with the optimal parameters giving an estimate for the output $\tilde{y} = f_{\theta^*}(\tilde{x})$. Perhaps the simplest example of this is linear regression: one finds the hyperplane that best represents the data and then uses this hyperplane to interpolate or extrapolate to previously unseen points. As a more advanced example, in a neural network one uses training to learn the weights of the network, after which prediction can be done by running the network forwards.

There are many intuitive reasons to take a discriminative machine learning approach. Perhaps the most compelling is the idea that if our objective is prediction, then it is simplest to solve that problem directly, rather than try and solve some more general problem such as learning an underlying generative process [Vapnik, 1998; Breiman et al., 2001]. Furthermore, if sufficient data is provided, discriminant approaches can be spectacularly successful in terms of predictive performance. Discriminant methods are typically highly flexible and can capture intricate structure in the data that would be hard, or even impossible, to establish manually. Many approaches can also be run with little or no input on behalf of the user, delivering state-of-the-art performance when used "out-of-the-box" with default parameters [Rainforth and Wood, 2015].

However, this black-box nature is also often their downfall. Discriminative methods typically make such weak assumptions about the underlying process that is difficult to impart prior knowledge or domain-specific expertise. This can be disastrous if insufficient data is available, as the data alone is unlikely to possess the required information to make adequate predictions. Even when substantial data is available, there may be significant prior information available that needs to be exploited for effective performance. For example, in time series modeling the sequential nature of the data is critically important information [Liu and Chen, 1998], while in vision tasks the knowledge that scenes are generated from objects can be invaluable [Kulkarni et al., 2015]. Many problems also increase in complexity as more data is added – "big data" problems are often actually a collection, or sometimes hierarchy, of many small problems, such that the complexity of the required parametrization increases as more data is added. Consider, for example, modeling interactions in a social network. Adding a new user into the model increases the amount of data, but also requires the model to grow and accommodate the new user [Ravasz and Barabási, 2003]. In this situation, it is essential to use an approach that respects the known structure, while the amount of data available for each individual user is often quite small, such that it will be essential to use prior information by transferring insights gathered from some users

to others. Therefore even for such large-scale problems, the inflexibility of many discriminative approaches to incorporate known characteristics of the target problem can be problematic.

Not only does the black-box nature of many discriminative methods restrict the level of human input that can be imparted on the system, it often restricts the amount of insight and information that can be extracted *from* the system once trained. The parameters in most discriminative algorithms do not have physical meaning that can be queried by a user, making their operation difficult to interpret and hampering the process of improving the system through manual revision of the algorithm. Furthermore, this typically makes them inappropriate for more statistics orientated tasks, where it is the parameters themselves which are of interest, rather than the ability for the system itself to make predictions. For example, the parameters may have real-world physical interpretations which we wish to learn about.

Most discriminative methods are also poor at providing realistic uncertainty estimates. Because they are typically trained in a manner that optimizes the parameters to minimize some loss criterion (e.g. the predictive error), they do not, in general, encode any uncertainty in either their parameters or the subsequent predictions. Though many methods can produce uncertainty estimates either as a by-product or from a post-processing step, these are typically heuristic based, rather than stemming naturally from a statistically principled estimate of the target uncertainty distribution. This lack of reliable uncertainty estimates can lead to overconfidence and can make discriminative methods inappropriate in many scenarios, e.g. for any application where there are safety concerns. It can also reduce the composability of discriminative methods within larger systems, as information is lost when only providing a point estimate. Not representing uncertainty in the parameters can also restrict the power of the resultant models, compared with approaches that can average over different possible parameter values.

These shortfalls mean that many tasks instead call for a *generative* machine learning approach [Ng and Jordan, 2002; Bishop, 2006]. Rather than directly learning a predictor, generative methods look to explain the observed data using a *probabilistic model*. Whereas discriminative approaches aim only to make predictions, generative approaches model how the data is actually generated: they model the joint probability $p(X, Y)$ of the inputs X and outputs Y. By comparison, we can think of discriminative approaches as only modeling the outputs given the inputs $Y|X$.

A key upshot of this difference is that generative approaches generally make stronger modeling assumptions about the problem. Though this can be problematic when the model assumptions are wrong and is often unnecessary in the limit of large data, it is essential for combining prior information with data and therefore for constructing systems that exploit application-specific

expertise. In the eternal words of George Box, "*all models are wrong, but some are useful*" [Box, 1979; Box et al., 1979]. In a way, this is a self-fulfilling statement: a model for any real phenomena is by definition an approximation and so is never exactly correct, no matter how powerful. However, it is still an essential point that is all too often forgotten, particularly by academics trying to convince the world that only their approach is correct. Only in artificial situations can we construct exact models and so we must remember, particularly in generative machine learning, that the first, and often largest, error is in our original mathematical abstraction of the problem. On the other hand, real situations have access to finite and often highly restricted data, so it is equally preposterous to suggest that a method is superior simply due to better asymptotic behavior in the limit of large data, or that if our approach does not work then the solution always just to get more data.[1] As such, the ease of which domain-specific expertise can be included in generative approaches is often essential to achieving effective performance on real-world tasks.

To highlight the difference between discriminative and generative machine learning, we consider the example of the differences between logistic regression (a discriminative classifier) and naïve Bayes (a generative classifier). We will consider the binary classification case for simplicity. Logistic regression is a linear classification method where the class label $y \in \{-1, +1\}$ is predicted from the input features $x \in \mathbb{R}^D$ using

$$p_{a,b}(y|x) = \frac{1}{1 + \exp(-y(a + b^T x))}, \tag{3.1}$$

and where $a \in \mathbb{R}^D$ and $b \in \mathbb{R}^D$ are the parameters of the model. The model is trained by finding the values for a and b that minimize a loss function on the training data. For example, a common approach is to find the *most likely* parameters a^* and b^* by minimizing cross-entropy loss function

$$\{a^*, b^*\} = \underset{a \in \mathbb{R}^D, b \in \mathbb{R}^D}{\arg\min} - \log \left(\prod_{n=1}^{N} p_{a,b}(y_n|x_n) \right). \tag{3.2}$$

Once found, a^* and b^* can be used with (3.1) to make predictions at any possible x. Logistic regression is a discriminative approach as we have directly calculated a characterization for the predictive distribution, rather than constructing a joint distribution on the inputs and outputs.

The naïve Bayes classifier, on the other hand, constructs a generative mode for the data. Namely it presumes that each data point is generated by sampling a class label $y_n \sim p_\psi(y)$ and

[1] It should, of course, be noted that the availability of data is typically the biggest bottleneck in machine learning. At times, it feels like the machine learning community would we well served to remember that the differences in performance between machine learning approaches is often, if not usually, dominated by variations in the inherently difficulty of the problem, which is itself not usually known up front, rather than differences between approaches.

then sampling the features given the class label $x_n \sim p_\phi(x|y_n)$. Here the so-called naïve Bayes assumption is that different data points are generated independently given the class label, namely

$$p_{\psi,\phi}(y_{1:N}|x_{1:N}) \propto p_\psi(y_{1:N}) \prod_{n=1}^{N} p_\phi(x_n|y_n). \tag{3.3}$$

We are free to choose the form for both $p_\psi(x|y)$ and $p_\phi(y)$ and we will use the data to learn their parameters ψ and ϕ. For example, we could take a maximum likelihood approach by calculating

$$\{\psi^*, \phi^*\} = \arg\max_{\psi,\phi} p_{\psi,\phi}(y_{1:N}|x_{1:N}) = \arg\max_{\psi,\phi} p_\psi(y_{1:N}) \prod_{n=1}^{N} p_\phi(x_n|y_n) \tag{3.4}$$

and then using these parameters to make predictions $\tilde{y}$ at a given input $\tilde{x}$ at test time as follows

$$p_{\psi^*,\phi^*}(\tilde{y}|\tilde{x}) \propto p_{\psi^*}(\tilde{y}) p_{\phi^*}(\tilde{x}|\tilde{y}). \tag{3.5}$$

The freedom to choose the form for $p_\psi(x|y)$ and $p_\phi(y)$ is both a blessing and a curse of this generative approach: it allows us to impart our own knowledge about the problem on the model, but we may be forced to make assumptions without proper justification in the interest of tractability, for convenience, in error, or simply because it is challenging to specify a sufficiently general purpose model that can cover all possible cases. Further, even after the forms of $p_\phi(x|y)$ and $p_\psi(y)$ have been defined, there are still decisions to be made: do we take a Bayesian or frequentist approach for making predictions? What is the best way to calculate the information required to make predictions? We will go into these questions in more depth in Section 3.4.

As we have shown, generative approaches are inherently probabilistic. This is highly convenient when it comes to calculating uncertainty estimates or gaining insight from our trained model. They are generally more intuitive than discriminative methods, as, in essence, they constitute an explanation for how the data is generated. As such, the parameters tend to have physical interpretation in the generative process and therefore provide not only prediction but also insight. Generative approaches will not always be preferable, particularly when there is an abundance of data available, but they provide a very powerful framework that is essential in many scenarios. Perhaps their greatest strength is in allowing the use of so-called Bayesian approaches, which we now introduce.

3.2 Learning from Data – the Bayesian Paradigm

At its core, the Bayesian paradigm is simple, intuitive, and compelling: for any task involving learning from data, we start with some prior knowledge and then update that knowledge to incorporate information from the data. This process is known as *Bayesian inference*. To give an intuitive example, consider the problem of identifying objects in a visual scene. Here it is

relatively straightforward to construct a model for generating images by constructing a sampler for which objects appear in the scene and their respective positions. Such graphics generators are used for computer games all the time. Here our parameters are the objects and the data is the image. Bayesian inference can now be thought of as the process of *inverting* our generator: given objects, we can already know how to generate images, but what we want to do is identify objects from images. We will return to these ideas in Chapter 4 where we show how, using probabilistic programming, one can think of all stochastic simulators as defining Bayesian models and the process of inference as inverting these simulators.

To be more precise, imagine we are trying to reason about some variables or parameters θ. We can encode our initial belief as probabilities for different possible instances of θ, this is known as a *prior* $p(\theta)$. Given observed data $\mathcal{D}$, we can characterize how likely different values of θ are to have given rise to that data using a *likelihood function* $p(\mathcal{D}|\theta)$. These can then be combined using Bayes' rule to give a *posterior*, $p(\theta|\mathcal{D})$ that represents our updated belief about θ once the information from the data has been incorporated

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{\int p(\mathcal{D}|\theta)p(\theta)d\theta} = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})}. \tag{3.6}$$

Here the denominator, $p(\mathcal{D})$, is a normalization constant known as the *marginal likelihood* and is necessary to ensure $p(\theta|\mathcal{D})$ is a valid probability distribution (or probability density for continuous problems). One can, therefore, think of Bayes' rule in the even simpler form of the posterior being proportional to the prior times the likelihood. For such a fundamental theorem, Bayes' rule has a remarkably simple derivation, following directly from the product rule of probability as shown in Chapter 2.

A key feature of Bayes' rule is that it can be used in a self-similar fashion where the posterior from one task becomes the prior when the model is updated with more data, i.e.

$$p(\theta|\mathcal{D}_1, \mathcal{D}_2) = \frac{p(\mathcal{D}_2|\theta, \mathcal{D}_1)p(\theta|\mathcal{D}_1)}{p(\mathcal{D}_2|\mathcal{D}_1)} = \frac{p(\mathcal{D}_2|\theta, \mathcal{D}_1)p(\mathcal{D}_1|\theta)p(\theta)}{p(\mathcal{D}_2|\mathcal{D}_1)p(\mathcal{D}_1)}. \tag{3.7}$$

As a consequence, there is something quintessentially human about the Bayesian paradigm: we learn from our experiences by updating our beliefs after making observations. Our model of the world is constantly evolving with time and is the cumulation of experiences over a lifetime. If we make an observation that goes against our prior experience, we do not suddenly make drastic changes to our underlying belief,[2] but if we see multiple corroborating observations our view will change. Furthermore, once we have developed a strong prior belief about something, we can

[2]This is not always quite true – as probabilities are multiplicative then a particularly unexpected observation can still drastically change our distribution.

take substantial convincing to change our mind, even if that prior belief is high illogical. Perhaps this why humans seem to have a tendency to develop deep-rooted prejudices.

There is similarly something distinctively Bayesian to the scientific process itself. In science, we construct models to explain observed phenomena and then run experiments to validate how well our model matches real observations. We then update and improve our model accordingly in a never-ending process of increasing understanding for the world around us. We can never hope to truly understand the workings of the universe – after all, it is, at least for practical purposes, fundamentally random – and so we can hope only to construct increasingly accurate and pertinent models.

To give a more concrete example of Bayesian modeling, consider estimating the probability of getting a heads from a weighted coin. Let's call this weighting $\theta \in [0, 1]$ such that the probability of getting a heads (H) when flipping the coin is $p(y = H|\theta) = \theta$ where y is the outcome of the flip. This will be our likelihood function, corresponding to a *Bernoulli* distribution, noting that the probability of getting a tails (T) is $p(y = T|\theta) = 1 - \theta$. Before seeing the coin being flipped we have some prior belief about its weighting. We can, therefore, define a prior $p(\theta)$, for which we will take the beta distribution

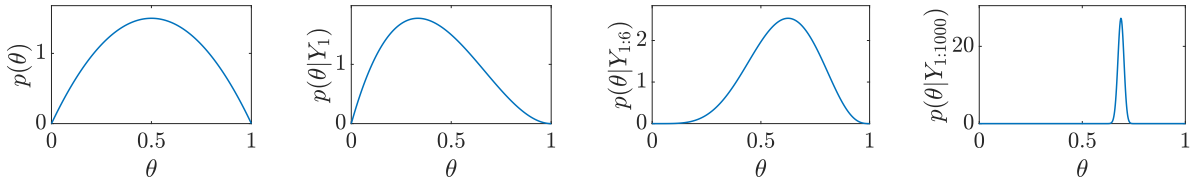
$$p(\theta) = \text{BETA}(\theta; \alpha, \beta) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)}\theta^{\alpha-1}(1 - \theta)^{\beta-1} \tag{3.8}$$

where $\Gamma(\cdot)$ is the gamma function and we will set $\alpha = \beta = 2$. A plot for this prior is shown in Figure 3.1a where we see that under our prior then it is more probable that θ is close to 0.5 than the extremes 0 and 1.

We now flip the coin and get a tails (T). We can calculate the posterior using Bayes' rule

$$p(\theta|y_1 = T) = \frac{p(\theta)p(y_1 = T|\theta)}{\int p(\theta)p(y_1 = T|\theta)d\theta} = \frac{\theta(1 - \theta)^2}{\int \theta(1 - \theta)^2 d\theta} = \text{BETA}(\theta; 2, 3)\,. \tag{3.9}$$

Here we have used the fact that a Beta prior is *conjugate* to a Bernoulli likelihood to give an analytic solution. Conjugacy means that the prior-likelihood combination gives a posterior that is of the same form as the prior distribution. More generally, for a prior of $\text{BETA}(\theta; \alpha, \beta)$ then the posterior will be $\text{BETA}(\theta; \alpha + 1, \beta)$ if we observe a heads and $\text{BETA}(\theta; \alpha, \beta + 1)$ if we observe a tails. Figure 3.1b shows that our posterior incorporates the information from the prior and the observed data. For example, our observation means that it becomes more probable that $\theta < 0.5$. The posterior also reflects the fact that we are still uncertain about the value of θ, it is not simply the empirical average of our observations which would give $\theta = 0$.



(a) Prior **(b)** Posterior 1 flip **(c)** Posterior 6 flips **(d)** Posterior 1000 flips

Figure 3.1: Prior and posteriors for coin flip example after different numbers of observations.

If we now flip the coin again, our previous posterior (3.9) becomes our prior and we can incorporate the new observations in the same way. Through our previous conjugacy result, then if we observe n_H heads and n_T tails and our prior is $\text{BETA}(\theta; \alpha, \beta)$ then our posterior is $\text{BETA}(\theta; \alpha + n_H, \beta + n_T)$. Thus if our sequence of new observations is $HTHHH$ then our new posterior is

$$p(\theta|y_1, \ldots, y_6) = \frac{p(y_2, \ldots, y_6|\theta)p(\theta|y_1)}{\int p(y_2, \ldots, y_6|\theta)p(\theta|y_1)d\theta} = \text{BETA}\left(\theta; 6, 4\right), \tag{3.10}$$

which is shown in Figure 3.1c. We see now that our belief for the probability of heads has shifted higher and that the uncertainty has reduced because of the increased number of observations. After seeing a total of 1000 observations as shown in Figure 3.1d, we find that the posterior has predominantly collapsed down to a small range of θ.

Having calculated our posterior, we can now make predictions using the *posterior predictive distribution* which marginalizes out over the parameters. For the coin flip case, we have

$$\begin{aligned} p(y_{N+1} = H|y_{1:N}) &= \int p(y_{N+1} = H, \theta|y_{1:N})d\theta = \int p(y_{N+1} = H|\theta)p(\theta|y_{1:N})d\theta \\ &= \int \theta \ \text{BETA}(\theta; \alpha + n_H, \beta + n_T)d\theta = \frac{\alpha + n_H}{\alpha + n_H + \beta + n_T} \end{aligned} \tag{3.11}$$

where we have used the known result for the mean of the Beta distribution. The role of the parameters α and β in our prior now become apparent – they take on the role of pseudo-observations. Our prediction is in line with the empirical average from seeing $\alpha + n_H$ heads and $\beta + n_T$ tails. The larger $\alpha + \beta$ is then the strong our prior compared to the observations, while we can skew towards heads or tails being more likely by changing the relative values of α and β.

More generally the posterior predictive distribution will depend on a queried input point. To demonstrate this, consider the example of a Bayesian linear regression from inputs $x \in \mathbb{R}^D$ to outputs $y \in \mathbb{R}^D$. Assume that we have N observations $\mathcal{D} = \{x_n, y_n\}_{n=1}^N$ and let $\mathbf{x} = [x_1, \ldots, x_N]^T$ and $\mathbf{y} = [y_1, \ldots, y_N]^T$ respectively be a $N \times D$ matrix and a column vector whose rows correspond to the different data points. Our regression is of the form $y_n = x_n^T\mathbf{w} + b + \epsilon_n$ where $\mathbf{w} \in \mathbb{R}^D$, $b \in \mathbb{R}$ and each $\epsilon_n \overset{i.i.d.}{\sim} \mathcal{N}(0, \sigma^2)$. This implies a likelihood of

$$p(\mathbf{y}|\mathbf{x}, \mathbf{w}, b, \sigma) = \prod_{n=1}^N p(y_n|x_n, \mathbf{w}, b, \sigma) = \prod_{n=1}^N \mathcal{N}(y_n; x_n^T\mathbf{w} + b, \sigma^2). \tag{3.12}$$

For simplicity, we will assume that σ and b are known fixed parameters, but we will put a prior on $\mathbf{w}$, namely $p(\mathbf{w}) = \mathcal{N}(\mathbf{w}; \mathbf{0}, C)$ where C is a fixed covariance matrix, in order to perform inference. To make predictions, we first calculate the posterior

$$p(\mathbf{w}|\mathcal{D}, b, \sigma) = \mathcal{N}(\mathbf{w}; \mathbf{0}, C) \prod_{n=1}^{N} \mathcal{N}(y_n; x_n^T \mathbf{w} + b, \sigma^2) = \mathcal{N}(\mathbf{w}; m, S) \tag{3.13}$$

$$\text{where} \quad m = S^{-1}\mathbf{x}^T (\mathbf{y} - b) / \sigma^2 \quad \text{and} \quad S = \left(C^{-1} + \frac{\mathbf{x}^T \mathbf{x}}{\sigma^2} \right)^{-1}.$$

We have omitted the necessary linear algebra (see for example Bishop [2006] Sections 2.3.3 and 3.3) but note the conjugacy between the normal distribution and itself. Prediction now uses the posterior predictive, marginalizing over the parameters in the same manner as the coin flip example. Here though, we are interested in predicting the output $\tilde{y}$ at a particular input point $\tilde{x}$ for which we have

$$\begin{aligned} p(\tilde{y}|\tilde{x}, \mathcal{D}, b, \sigma) &= \int p(\tilde{y}|\tilde{x}, \mathbf{w}) p(\mathbf{w}|\mathcal{D}, b, \sigma) d\mathbf{w} = \int \mathcal{N}(\tilde{y}; \tilde{x}^T \mathbf{w} + b, \sigma^2) \mathcal{N}(\mathbf{w}; m, S)\, d\mathbf{w} \\ &= \mathcal{N}\left(\tilde{y}\ ;\ \tilde{x}^T m + b,\ \tilde{x}^T S^{-1} \tilde{x} + \frac{1}{\sigma^2} \right) \end{aligned} \tag{3.14}$$

which again follows from standard results for Gaussian distributions. We, therefore, have an analytic predictive distribution at any possible input point. Though this linear regression example might seem overly simple for practical purposes, we will see in Section 8.2 that substantially more advanced models, such as Gaussian processes, can be viewed as linear regressions between a set of features on the inputs $\phi(x)$ and the output y.

The models we have introduced so far are specific examples of Bayesian modeling and will clearly only be applicable or appropriate in very particular scenarios. Bayesian modeling is at its heart a generative approach and its real power will be when we design a rich and expressive *generative model* that reflects our application-specific knowledge. In other words, we can define our model by carefully constructing a stochastic process that describes how the parameters and data are generated. For our object recognition example at the start of the section, this would correspond to a graphics generator for sampling scenes. More generally, the model corresponds to the joint distribution on parameters and data $p(\theta, \mathcal{D})$. We can then think of observing real data as refining our model through *conditioning*, providing an updated distribution on the parameters $p(\theta|\mathcal{D})$ and subsequent predictions that incorporate the information from the data. The more flexible we make our generative model, the more we can make it represent the our data. The less flexible we make the generative model, the more weighting is given to our prior assumptions.

We finish by making a technical point of note about the behavior of Bayesian methods of the limit of large data. Assume that our likelihood model $p(\mathcal{D}|\theta)$ is correct such that the data $\mathcal{D} = y_{1:N}$ is generated according to $p(y_{1:N}|\theta^*)$ where θ^* are the (finite) set of ground truth parameters and the prior $p(\theta)$ satisfies $p(\theta^*) > 0$. Informally speaking, the Bernstein-Von Mises theorem now states that in the limit of large N, the posterior distribution $p(\theta|y_{1:N})$ converges to a normal distribution we mean θ^* and variance of order $O(1/N)$ (i.e. it decreases at a rate $1/N$) [Doob, 1949; Freedman, 1963]. This is a hugely important result in Bayesian statistics as it demonstrates that, when our model assumptions are correct, we converge to the true parameters and the posterior becomes independent of the prior when we are provided with sufficient data. It further transpires that when no such θ^* exists (i.e. our model is misspecified), the convergence is instead to the parameters $\hat{\theta}$ which minimize the Kullback-Leibler (KL) divergence[3] to the true data generating distribution $p^*(y_{1:N})$, namely

$$\hat{\theta} = \arg\min_{\theta} \mathrm{KL}\left(p^*(y_{1:N}) \| p(y_{1:N}|\theta)\right) = \arg\min_{\theta} \int p^*(y_{1:N}) \log\left(\frac{p^*(y_{1:N})}{p(y_{1:N}|\theta)}\right) dy_{1:N}. \quad (3.15)$$

See for example [Kleijn et al., 2012] and the references therein for further details.

3.3 Graphical Models

Generative models will typically have many variables and a complex *dependency structure*. In other words, many variables will be conditionally independent of one another given values for other variables. Graphical models are a ubiquitously used method for representing and reasoning about generative models, with a particular focus on the dependency structure. At a high-level, they capture how the joint probability distribution can be broken down into a product of different factors, each defined over a subset of the variables. They are formed of a number of connected nodes, where each node represents a random variable (or collection of random variables) in the model. Links between nodes represent dependencies: any two connected nodes have an explicit dependency, though unconnected nodes may still be dependent. Various independence assumptions can be deduced from the graphical model, though the exact nature of these deductions will depend on the type of graphical model – nodes without direct links between them will often still be dependent.

Graphical models can be separated into two distinct classes: directed graphical models and undirected graphical models. Undirected graphical models, also known as Markov random fields, imply no ordering on their factorization and are used only to express conditional independences

[3]The KL divergence can informally be thought of as a measure of discrepancy between two distributions. Though it is not symmetric in its inputs, it is always non-negative and zero if and only if the two distributions are the same.

between variables. They are used in scenarios where it is difficult to specify the target distribution in a generative way, e.g. Boltzmann machines [Ackley et al., 1985]. To give a more concrete example, if modeling whether it will rain at various locations, then there is a clear dependence between nearby locations, but not a natural ordering to the joint probability distribution of where it will rain. Independence in undirected graphical models can be deduced through the *global Markov property* which states that any two non-intersecting subsets of the variables A and B are conditionally independent given a third, separating, subset C if there is no path between A and B that does not pass through C. This means, for example, that each variable is conditionally independent of all the other variables given its neighbors.

Our main focus, though, will instead be on directed graphical models and in particular directed acyclic graphical models (DAGs), i.e. directed graphical models containing no cycles or loops one can follow and arrive back in the starting position. DAGs, also known as Bayesian networks, are particularly useful in the context of Bayesian modeling because they can be used to express *causal relationships*. As such, they can be used as a piecewise explanation for how samples are generated from a distribution. This forms a natural means to describe and design models as we can carefully order the breakdown to factorize the distribution into only terms we know. For example, in the linear regression model, we did not know (at least when the model was first defined) $p(\mathcal{D})$ but we did know $p(\mathbf{w})$ and $p(\mathcal{D}|\mathbf{w})$. Therefore even though we could factorize our joint $p(\mathcal{D}, \mathbf{w})$ as $p(\mathbf{w}|\mathcal{D})p(\mathcal{D})$ and construct a DAG this way, it is much more convenient to factorize and construct the DAG the other way round, namely as $p(\mathcal{D}|\mathbf{w})p(\mathbf{w})$. We will generally not have access to all possible factorizations in an analytic form as otherwise there would be no need to perform inference. As a rule-of-thumb, when we define a model using a DAG, we need to be able to define the probability of each variable given its *parents*, i.e. all the nodes with arrows, representing a link and its direction, pointing to the node the question.

To demonstrate this factorization more explicitly and give a concrete example of a DAG, imagine a medical diagnostic problem where we wish to predict if a patient has lung cancer. Let a denote lifestyle and genetic factors of the patient such as whether they smoke or have (potentially unknown) preexisting conditions. These will generally either be known or can reasonably be estimated by other means, e.g. using tests or by considering prevalence within the population, allowing definition of a prior marginal on a, $p(a)$. Given these factors, we can develop a model for the probability that a patient will develop lung cancer, which we can denote $p(b|a)$ where $b = 1$ indicates cancer is the present. Given the lifestyle and genetic factors and the knowledge of whether lung cancer is present, we can predict what symptoms,

c, might be observed, e.g. a persistent cough, encoding this using $p(c|a, b)$. We thus have the following breakdown of the joint distribution

$$p(a, b, c) = p(a)p(b|a)p(c|a, b). \tag{3.16}$$

which can be expressed using the DAG shown in Figure 3.2. Here we have shaded in c to express the fact that this is observed. The graphical model expresses our dependency structure as we have the probability of each node given its parents. As shown in (3.16), the product of all these factors is equal to our joint distribution. The DAG has thus formed a convenient means of representing our generative process. Our aim for this problem was to find the probability cancer is present given the observed symptoms, i.e. $p(b|c)$, which will require Bayesian inference. In our previous simple examples, the posterior had an analytic form. However, in general this will not be the case and we will need to develop strategies for carrying out the inference as explained in Chapter 5. For these, knowing the dependency structure and, in particular, the independence relationships, of a model will often be very helpful information, for which DAGs can be very useful.

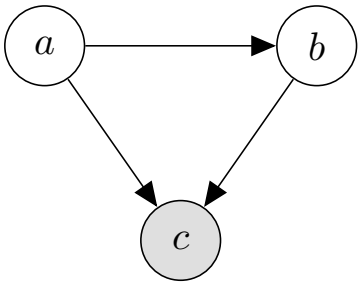


Figure 3.2: Simple example DAG corresponding to (3.16)

A natural question is now how can we deduce the independence relationships from a DAG? This can be done by introducing the notion of *d-separation* [Pearl, 2014]. Consider three arbitrary, non-intersecting, subsets A, B, and C of a DAG. A and B are conditionally independent given C if there are no *unblocked* paths from A to B (or equivalently from B to A), in which case A is said to be d-separated from B by C. Paths do not need to be in the directions defined by the DAG but are blocked if either

1. Consecutive arrows both point towards a node that is not in and has no descendants in C, i.e. we cannot get to any of the nodes in C by following the arrows from this node.
2. Consecutive arrows meet at a node in C and one of them points away from the node.

Note that only the first of these rules is necessary for establishing marginal independence between nodes as this rule can still be used when C is empty. Examples of blocked paths are shown in Figure 3.3 while examples of unblocked paths are shown in Figure 3.4, explanations for which are given in the captions. For a more comprehensive introduction to establishing independence in DAGs, we refer the reader to [Bishop, 2006, Section 8.2].

In the simple example of Figure 3.2 there were no independence relationships and so we gain little from working with the DAG compared to just the joint probability distribution. A more

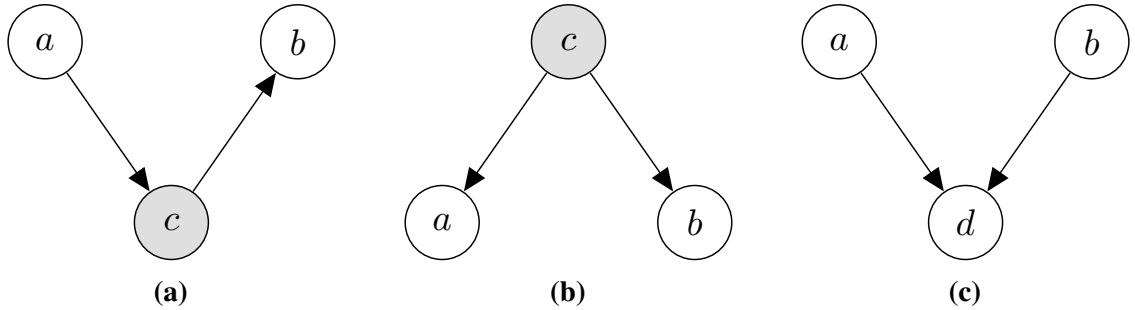


Figure 3.3: Examples of DAGs blocked between a and b. **(a)** is blocked by the second rule of d-separation, while **(b)** is blocked by the first. Consequently, for **(a)** and **(b)** then a and b are conditionally independent given c and thus $p(b|a,c) = p(b|c)$ and $p(a|b,c) = p(a|c)$. **(c)** is an instead example of where a and b are *marginally* independent. Here the path between a and b is blocked because the arrows meet head-to-head at d and neither d nor any of its descendants are observed and so $p(b|a) = p(b)$. Note though that a and b are not conditionally independent given d as the path becomes unblocked if this is observed (see Figure 3.4).

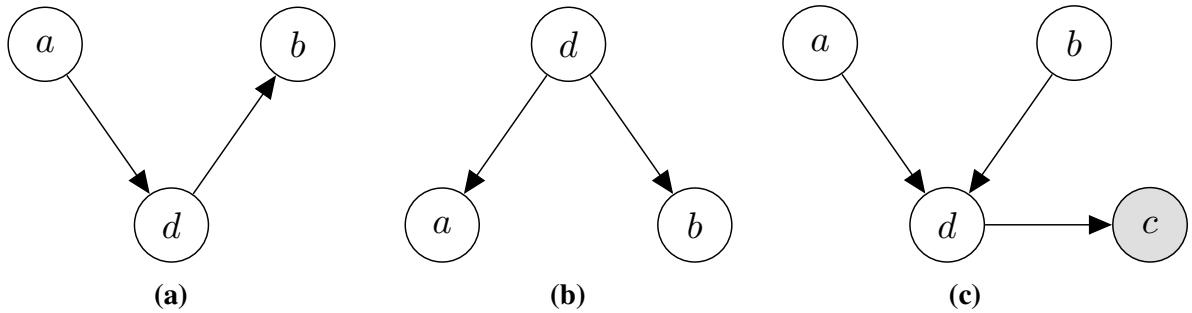


Figure 3.4: Examples of DAGs unblocked between a and b. For **(a)** then the path from a to b is not blocked by the d-separation rules. Perhaps more intuitively, we have that $p(b|a) = \int p(b,d|a)dd = \int p(b|d)p(d|a)dd \neq p(b)$ unless $p(d|a) = p(d)$. Similarly, there is an unblocked path for **(b)** from a to b as the path that does not pass through any observed nodes or nodes with both arrows points towards it. The path in **(c)** is unblocked because of the phenomenon of *explaining away*. The first rule of d-separation does not apply here because c is an observed descendant of d. We thus have that a and b are marginally independent as per Figure 3.3c, but not conditionally independent given c (and or d). The rationale for explaining away can be thought of in terms of events needing an explanation – if two precursor events (here a and b) can give rise to a third event (here c), then the third event occurring but not the first precursor event implies that the second precursor event occurs. Thus the two precursor events are correlated because one *explains away* the other. As described in Section 2.3, one example of this is that if a speed camera is triggered and the camera is not malfunctioning, this implies the vehicle is speeding, even though the vehicle speeding and the camera malfunctioning are marginally independent.

advanced example where there are substantial independence relationships which can be exploited is shown in Figure 3.5. This model is known as a hidden Markov model[4] (HMM) and has T latent variables $x_{1:T}$ and T observations $y_{1:T}$. The joint distribution is as follows

$$p(x_{1:T}, y_{1:T}) = p(x_1)p(y_1|x_1)\prod_{t=2}^{T} p(x_t|x_{t-1})p(y_t|x_t), \tag{3.17}$$

where each x_t is independent of $x_{1:t-2}$ and $y_{1:t-1}$ given x_{t-1} and of $x_{t+2:T}$ and $y_{t+1:T}$ given x_{t+1}. This is known as the Markov property and means that each latent variable only depends

[4]The use of the term HMM in the literature (and later in this paper) often also implies that the latent states are discrete variables, with the equivalent continuous model referred to as a (Markovian) state space model.

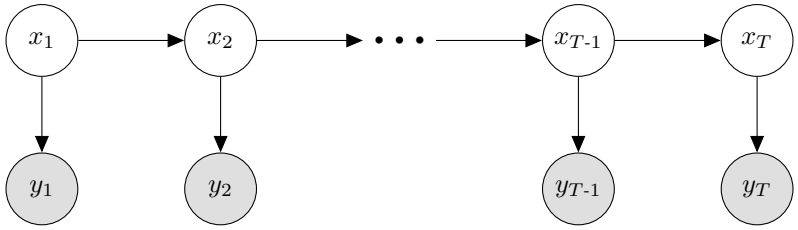


Figure 3.5: DAG for a hidden Markov model.

on the other variables and observations through its immediate neighboring states. In essence, the model has no memory as information is passed forwards (or backwards) only through the value of the previous (or next) latent state. A number of stochastic processes and dynamical systems obey the Markov property and HMMs and their extensions are extensively used for a number of tasks involving sequential data, such as DNA sequencing [Durbin et al., 1998] and tracking animals [Dhir et al., 2016, 2017] to name but a few.

A key part of the appeal of HMMs is that the structure of the DAG can be exploited to give analytic solutions to the resulting Bayesian inference whenever each $p(y_t|x_t)$ and $p(x_t|x_{t-1})$ are either a categorical or Gaussian distribution. Even when this does not hold, there are still a number of features of the dependency structure that can make the inference substantially easier. As we will show in Chapter 5, Bayesian inference is generally a challenging problem, often prohibitively so. Therefore the (fast) analytic inference for HMMs is highly convenient. However, it can mean that HMMs are perhaps overused. More generally, simplifying approximations or unjustified assumptions are often made by Bayesian practitioners for tractability, e.g. by using an off-the-shelf model like an HMM with known analytic solution. Though often necessary, this must be done with extreme care and the implications of the approximations should carefully considered. Unfortunately, quantifying the implications of approximations can be very difficult, given that they are typically made in the interest of tractability in the first place.

3.4 Bayesianism vs Frequentism

We have just introduced the Bayesian approach to generative modeling, but this is far from the only possible approach. In this section, we will briefly introduce and compare the alternative, *frequentist*, approach. As a community, we have a come a long way from the absolutism of Feller [1950], with most researchers appreciating that both Bayesian and frequentist approaches are a necessary component of the general statistical method. Nonetheless, the divide between those within the statistics and machine learning communities who advocate, at least at the philosophical level, the use Bayesian or frequentist methods is at times surprisingly large. Many

researchers have strong views one way or another and it is easy, certainly as a graduate student, to be sufficiently isolated to develop a somewhat warped view of the overall picture. The actual statistical differences between the approaches are somewhat distinct to the well-known philosophical differences we touched on in Section 2.2, even though the latter are often dubiously used for justification for the practical application of a particular approach. These statistical differences are arguably less well-known, in general, by those in the early stages of their research careers without statistics backgrounds. Our aim in this section is not to advocate the use of one approach over the other, but to (hopefully objectively) highlight these statistical differences and demonstrate that *both* approaches have advantages and drawbacks, such that "head in the sand" mentalities either way can be highly detrimental, with effective modeling often requiring us to draw on both. We note that whereas Bayesian methods are always, at least in theory, generative [Gelman et al., 2014, Section 14.1], frequentist methods can be either generative or discriminative. As we have already discussed differences between generative and discriminative modeling in Section 3.1, we will mostly omit this difference from our subsequent discussion.

At their root, the statistical differences between Bayesian and frequentist methods[5] stem from distinct fundamental assumptions: frequentist modeling presumes fixed parameters, Bayesian modeling assumes fixed data [Jordan, 2009]. In many ways, both of these assumptions are somewhat dubious. Why assume fixed parameters when we do not have enough information from the data to be certain of the correct value? Why ignore that fact that other data could have been generated by the same underlying true parameters? However, making such assumptions can sometimes be unavoidable for carrying out particular analyses.

To elucidate the different assumptions further and start looking into why they are made, we will now step into a decision-theoretic framework. Let's presume that the universe gives us some data X and some true parameter θ, the former of which we can access, but the latter of which is unknown. We can alternatively think in terms of X being some information that we actually receive and θ being some underlying truth or oracle from which we could make optimal predictions, noting that there is no need for θ to be some explicit finite parameterization. Any machine learning approach will take the data as input and return some artifact or decision, for example, predictions for previously unseen inputs. Let's call this process the decision rule d, which we presume, for the sake of argument, to be deterministic for a giving dataset, producing

[5] At least in their decision-theoretic frameworks. It is somewhat inevitable that delineation here and later will be a simplification on what is, in truth, not a clear-cut divide [Gelman et al., 2011].

decisions $d(X)$.[6] Presuming that our analysis is not frivolous, there will be some loss function $L(d(X), \theta)$ associated with the action we take and the true parameter θ, even if this loss function is subjective or unknown. At a high level, our aim is always to minimize this loss, but what we mean by minimizing the loss changes between the Bayesian and frequentist settings.

In the frequentist setting, X is a random variable but θ is not. Therefore, one takes an expectation over possible data that could have been generated, giving the frequentist risk [Vapnik, 1998]

$$R(\theta, d) = \mathbb{E}\left[L(d(X), \theta)|\theta\right] \tag{3.18}$$

which is thus a function of theta and our decision rule. The frequentist focus is therefore on *repeatability*, i.e. the generalization of the approach to different datasets that *could* have been generated. Choosing the parameters θ is thus based on optimizing for the best average performance over all possible datasets.

In the Bayesian setting, θ is a random variable but X is fixed: the focus of the Bayesian approach is on generalizing over possible values of the parameters and using all the information at hand. Therefore one takes an expectation over θ to make predictions conditioned on the value of X, giving the *posterior expected loss* [Robert, 2007]

$$\varrho(\pi, d(X)|X) = \mathbb{E}_{\pi(\theta|X)}[L(d(X), \theta)|X], \tag{3.19}$$

where $\pi(\theta|X)$ is our posterior distribution on θ. Although $\varrho(\pi, d(X)|X)$ is a function of the data, the Bayesian approach takes the data as given (after all we have a particular dataset) and so for a given prior and decision rule, the posterior expected loss is a fixed value and, unlike in the frequentist case, further assumptions are not required to calculate the optimal decision rule d^*. To see this, we can consider calculating the *Bayes risk* [Robert, 2007], also known as the *integrated risk*, which averages over both data and parameters

$$r(\pi, d) = \mathbb{E}\left[\varrho(\pi, d(X)|X)\right] = \mathbb{E}_{\pi(\theta|X)}\left[R(\theta, d)\right]. \tag{3.20}$$

Here we have noted that we could have equivalently taken the expectation of the frequentist risk over the posterior, such that, despite the name, the Bayes risk is neither wholly Bayesian nor frequentist. It is now straightforward to show (see e.g. [Robert, 2007]) that the decision function which minimizes $r(\pi, d)$ is obtained by, for each possible dataset $X \in \mathcal{X}$, choosing the decision that minimizes the posterior expected loss, i.e. $d^*(X) = \arg\min_{d(X)} \varrho(\pi, d(X)|X)$. By comparison, because the frequentist risk is still a function of the parameters, further work is

[6] If we allow our predictions to be probability distributions this assumption is effectively equivalent to assuming we can solve any analysis required by our approach exactly.

required to define the optimal decision rule, e.g. by taking a *minimax* approach [Vapnik, 1998]. We now see that the Bayesian approach can be relatively optimistic, as it is constrained to choose decisions that optimize the expected loss, whereas the frequentist approach allows, for example, d to be chosen in a manner that optimizes for the worst case θ.

We now introduce some shortfalls that originate from taking each approach. We emphasize that we are only scratching the surface of one of the most hotly debated issues in statistics and do not even come close to doing the topic justice. Our aim is less to provide a comprehensive explanation of the relative merits of the two approaches, but more to make the reader aware that there are a vast array of complicated, and sometimes controversial, issues associated with whether to use a Bayesian or frequentist approach, most of which have no simple objective conclusion.

3.4.1 Shortcomings of the Frequentist Approach

One of the key criticisms of the frequentist approach is that predictions depend on the experimental procedure and can violate the *likelihood principle*. The likelihood principle states that, for a given model, the only information relevant to the parameters θ conveyed by the data is encoded through the likelihood function [Robert, 2007]. In other words, the same data and the same model should always lead to the same inferences about θ. Though this sounds intuitively obvious, it is actually violated by taking an expectation of X in frequentist methods, as this introduces a dependency from the experimental procedure.

As a classic example, imagine that our data from flipping a coin is 3 heads and 9 tails. In a frequentist setting, we make different inferences about whether the coin is biased depending on whether our data originated from flipping the coin 12 times and counting the number of heads, or if we flipped the coin until we got 3 heads. For example, at the 5% level of a *significance test*, we can reject the *null hypothesis* that the coin is unbiased in the latter case, but not the former. This is obviously somewhat problematic, but it can be used to argue both for and against frequentist methods. Using it to argue against frequentist methods, and in particular significance tests, is quite straightforward: the subjective differences in our experiment should not affect our conclusions about whether the coin is fair or not. We can also take things further and make the results change for absurd reasons. For example, imagine our experimenter had intended to flip until she got 3 heads, but was then attacked and killed by a bear while the twelfth flip was in the air, such that further flips would not have been possible regardless of the outcome. In the frequentist setting, this again changes our conclusions about whether the coin is biased. Clearly,

it is ridiculous that the occurrence or lack of a bear attack during the experiment should change our inferences, but that is need-to-know information for frequentist approaches.

As we previously suggested though, one can also use this example to argue for frequentist methods as one can argue that it actually suggests the likelihood principle is incorrect. Although significance tests are a terribly abused tool whose misuse has had severe detrimental impact on many applied communities [Goodman, 1999; Ioannidis, 2005], they are not incorrect, and extremely useful, if interpreted correctly. If one very carefully considers the definition of a *p-value* as being the probability that a given, or more extreme, event is observed if the *experiment is repeated*, we see that our bear attack does actually affect the outcome. Namely, the chance of getting the same or more extreme data from repeating the experiment of "*flip the coin until you get* 3 *heads*" is different to the chance of getting the same or a more extreme result from repeating the experiment "*flip the coin until you get* 3 *heads or make* 12 *flips (at which point you will be killed by a bear)*". As such, one can argue that the apparently absurd changes in conclusions originate from misinterpreting the results and that, in fact, these changes actually demonstrate that the likelihood principle is flawed because, without a notion of an experimental procedure, we have no concept of repeatability. Imagine instead the more practical scenario where a suspect researcher stops their experiment early as it looks like the results are likely to support their hypothesis and they do not want to take the risk that if they keep it running as long as they intended, then the results might no longer be so good. Here the researcher has clearly biased their results in a way that ostensibly violates the likelihood principle.

Whichever view you take, two things are relatively indisputable. Firstly a number of a frequentist concepts, such as p-values, are not compatible with the likelihood principle. Secondly, frequentist methods are not always *coherent*, such that they can return answers that are not consistent with each other, e.g. probabilities that do not sum to one.

Another major criticism of the frequentist approach is that it takes a point estimate for θ, rather than averaging over different possible parameters. This can be somewhat inefficient in the finite data case, as it limits the information gathered from the learning process to that encoded by the calculated point estimate for θ, which is then wasteful when making predictions. Part of the reason that this is done is to actively avoid placing a prior distribution on the parameters, either because this prior distribution might be "wrong"[7] or because, at a more philosophical level, they are not random variables under the frequentist definition of probability. Some people thus object

[7]Whether a prior can be wrong, or what that even means, is a somewhat open question except in the case where it fails to put any probability mass (or density for continuous problems) on the ground truth value of the parameters.

to placing a distribution over them at a fundamental level (we will see this objection mirrored by Bayesians for the data in the next section). For the Bayesian perspective (and a viewpoint we actively argue for elsewhere in the paper), this is itself also a weakness of the frequentist approach as incorporating prior information is often essential for effective modeling.

3.4.2 Shortcomings of the Bayesian Approach

Unfortunately, the Bayesian approach is also not without its shortcomings. We have already discussed one key criticism in the last section in that the Bayesian approach relies on the likelihood principle which itself may not be sound, or at the very least ill-suited for some statistical modeling problems. More generally, it can be seen as foolhardy to not consider other possible datasets that might have been generated. Taking a very strict stance, then even checking the performance of a Bayesian method on test data is fundamentally frequentist, as we are assessing how well our model generalizes to other data. Pure Bayesianism, which is admittedly not usually carried out in practice, shuns empiricism as empiricism, by definition, is rooted in the concept of repeated trials which is not possible if the data is kept fixed. The rationale typically given for this is that we should use all the information available in the data and by calculating a frequentist risk we are throwing some of this away. For example, cross-validation approaches only ever use a subset of the data when training the model. However, a common key aim of statistics is generalization and repeatability. Pure Bayesian approaches include no consideration for *calibration*, which means that, even if our likelihood model is correct, there is still no reason that any probabilities or confidence intervals must be also. This at odds with frequentist approaches, for which we can often derive absolute guarantees.

A related issue is that Bayesian approaches will often reach spectacularly different conclusions for ostensibly inconsequential changes between datasets.[8] At least when making the standard assumption of i.i.d. data in Bayesian analysis, then likelihood terms are multiplicative and so typically when one adds more data, the relative probabilities of two parameters quickly diverge. This divergence is necessary for Bayesian methods to converge to the correct ground truth parameters for data distributed exactly as per the model, but it also means any slight misspecifications in the likelihood model become heavily emphasized very quickly. As a consequence, Bayesian methods can chronically underestimate uncertainty in the parameters, particularly for large datasets, because they do not account for the *unknown unknowns*. This means that ostensibly inconsequential features of the likelihood model can lead to massively

[8] Though this is arguably more of an issue with generative approaches than Bayesian methods in particular.

different conclusions about the relative probabilities of different parameters. In terms of the posterior expected loss, this is often not much of a problem as the assumptions might be similarly inconsequential for predictions. However, if our aim is actually to learn about the parameters themselves then this is quite worrying. At the very least it shows why we should view posteriors with a serious degree of skepticism (particularly in their uncertainty estimates), rather than taking them as ground truth.

Though techniques such as cross-validation can reduce sensitivity to model misspecification, generative frequentist methods still often do not fare much better than Bayesian methods for misspecified models (after all they produce no uncertainty estimates on θ). Discriminative methods, on the other hand, do not have an explicit model to specify in the same way and so are far less prone to the same issues. Therefore, though much of the criticisms of Bayesian modeling stem from the use of a prior, issues with model (i.e. likelihood) misspecification are often much more severe and predominantly shared with generative frequentist approaches [Gelman and Robert, 2013]. It is, therefore, often the question of discriminative vs generative machine learning that is most critical [Breiman et al., 2001].

Naturally one of the biggest concerns with Bayesian approaches is their use of a prior, with this being one of the biggest differences to generative frequentist approaches. The prior is typically a double-edged sword. On the one hand, it is necessary for combining existing knowledge and information from data in a principled manner, on the other hand, priors are inherently subjective and so all produced posteriors are similarly subjective. Given the same problem, different practitioners will use different priors and reach potentially different conclusions. In the Bayesian framework, there is no such thing as a correct posterior for a given likelihood (presuming finite data). Consequently, "all bets are off" for repeated experimentation with Bayesian methods as there is no quantification for how wrong our posterior predictive might be compared with the true generating distribution. This can mean they are somewhat inappropriate for tasks where the focus is on repeated use, e.g. providing reliable confidence intervals for medical trials, though many Bayesian methods retain good frequentist properties. Such cases both predicate a need for considering that there are many possible datasets that might occur and, ideally, an objective approach that means the conclusions are independent of the whims of the analyst.

In particular, there is no (objective) Bayesian alternative to frequentist *falsification* methods such as the aforementioned significance tests.[9] Both Bayesian and frequentist methods require

[9]This is not to say one cannot perform falsification as part of Bayesian modeling, in fact avoiding doing so would be ridiculous, but that providing objective statistical guarantees to this falsification requires the use of frequentist methods. For example, even the essential research process of informally rejecting and improving models undermines

assumptions and neither can ever truly prove that a particular model or prediction is correct, but frequentist methods do allow one to indisputably *disprove* hypotheses to within some confidence interval. The real power of this is realized when we consider disproving *null hypotheses*, e.g. the hypothesis that an output is independent of a potential driving factor. This is why significance tests are so widely used through the sciences as, although they cannot be used to prove a model is correct (as much as people might try), they can certainly be used to show particular assumptions or models are wrong.

The use of a prior in Bayesian modeling can also be problematic because it is often easy to end up "using the data twice" [Gelman et al., 2008]. The Bayesian paradigm requires that the prior is independent from the data and this means that there should not be any human-induced dependency. In other words, it is necessary that the user sets their prior before observing the data they condition on, otherwise, the data will both influence their choice of prior and be incorporated through the likelihood. This is not a problem with Bayesian methods per se, but it can be a common shortfall in their application.

Another practical issue with Bayesian methods is their computational complexity at both train time and test time. Firstly, using Bayesian methods requires one to carry out inference, which, as we explain in Chapter 5, is typically a challenging and computationally expensive process, often prohibitively so. Some frequentist approaches can also be similarly expensive at train time, but others can be substantially cheaper, particularly discriminative approaches. Further, Bayesian methods tend to also be somewhat expensive for making predictions with the posterior predictive itself being an integral. Frequentist methods tend to instead predict using point estimates for θ, such that prediction is typically much cheaper.

3.4.3 Practical Usage

Although Bayesianism and frequentism are both exact frameworks, their application to real problems is not. Both frameworks have their strengths and weaknesses and so perhaps the key question is not which framework is correct, but when should we use each. In particular, the Bayesian approach is often essential when working with small datasets but where we have substantial prior expertise. On the other hand, a frequentist approach is essential to providing guarantees and ensuring repeatability. Bayesian and frequentist methods are also by no means mutually exclusive and effective modeling often requires elements of both to be used concurrently. For example, one could be Bayesian about the results from a cross-validation test or look to

Bayesian coherence [Gelman et al., 2011]. On the other hand, the process of peer review effectively invalidates frequentist calibration by ensuring some studies never make it to the public domain.

calculate frequentist guarantees for a Bayesian model. In essence, Bayesian and frequentist analysis have different aims – Bayesianism is about updating subjective beliefs and frequentism is about creating long run, or repeated application, guarantees. We often we care about both. It is also worth noting that a number of Bayesian methods exhibit good frequentist properties, see e.g. McAllester [2013] and the references therein.

We finish by noting a critical assumption made by both Bayesian and generative frequentist methods – that there is some true underlying value for the parameters. Because all models are approximations of the real world, this is often a misguided and harmful assumption. That this assumption is made is clear in the frequentist setting, but is somewhat subtle for Bayesian approaches. Bayesian methods allow for multiple hypotheses or parameter values, but this originates from our own uncertainty about which parameter or hypothesis is correct, thereby still implicitly assuming that *one* of them is correct. Namely, as we showed with the Bernstein-Von Mises theorem, in the limit of large data, Bayesian methods with finite numbers of parameters will collapse to a point estimate, corresponding to the "true parameter values" (assuming the model is correct). Consequently, a Bayesian approach does not fundamentally enrich the model space by averaging over parameters – it is still necessary that exactly one set of parameters lead to the data, but we are not exactly sure which one [Minka, 2000].

Consider as an example, Bayesian modeling for decision trees [Chipman et al., 1998; Lakshminarayanan et al., 2013] compared to (discriminative) ensemble-based approaches [Breiman, 2001; Rainforth and Wood, 2015]. The Bayesian approaches explicitly assume that our data is generated by one single decision tree and so in the limit of large data the relative probabilities of different trees in the posterior diverge and will collapse to a single tree.[10] The ensemble approaches, on the other hand, maintain a full ensemble of trees in the limit of large data. They are, in fact, a fundamentally more general model class as they do not require the data to have been generated by a single tree. Here the averaging over trees enriches the model class itself, rather than just representing uncertainty about which of the trees in the ensemble is the correct one [Domingos, 1997], typically leading to better performing algorithms for large datasets than Bayesian approaches.

[10]Technically we only get a single tree if we limit the tree depth to ensure a finite parameterization. Nonetheless, the argument still holds from a practical perspective even when trees are unbounded.

3.5 Challenges of Bayesian Modeling

The challenges for Bayesian modeling are remarkably straightforward to identify. As all information is encoded through the prior and the likelihood, the "only" challenges are in specifying these and then carrying out the required Bayesian inference to estimate the posterior. In other words, we need to be able to specify good models and we need to be able to solve them. Though simple to quantify, actually overcoming both these challenges can be extremely difficult in practice. The problem of Bayesian inference, which we discuss in Chapters 5, 6, and 7, is at least clearly defined and one often has a reasonable idea how effectively we have addressed it when conducting Bayesian modeling, i.e. even if our efforts are futile, we often know when we have failed.

The problem of specifying good models is somewhat more subjective. It is inevitably impossible to design a model that covers all potential eventualities, both from the perspective of actually writing down such a model and in having a realistic chance of performing inference. Such a model would also most likely have limited predictive power. However, for practical problems, one can almost always add further refinements to the model at the expense of additional computational cost. For example, we can elaborate our likelihood model, or even use a combination of different possible models, to provide more potential means for generating the data and therefore a more flexible model that is less likely to succumb to misspecification issues. We can also augment our prior with a *hyperprior* that expresses uncertainty in the prior parameters themselves, e.g. for our coin flip example instead of taking a Beta distribution prior with fixed parameters, we could place a distribution on these parameters as well. However, at some point increasing the complexity of our likelihood and our prior inevitably has to stop, at which point we either have to fix the values for some parameters, conduct *model learning* by optimizing those parameter values as discussed in Chapters 8 and 9, or try to define a non-informative prior [Robert, 2007].

Probabilistic programming, as introduced in the next chapter, is an attempt to address the challenges of both specifying and solving models. Firstly, by providing a highly flexible and expressive framework for specifying models, they provide a platform for writing models that are true to the assumptions the user wishes to make. Secondly, they democratize the Bayesian inference process, by providing automated inference engines that can be run on any model the user might write, thereby streamlining the Bayesian modeling process and reducing barriers to entry for those with non-statistical backgrounds.

4

Probabilistic Programming – the User's Perspective

Probabilistic programming systems (PPSs) allow probabilistic models to be represented in the form of a generative model and statements for conditioning on data [Gordon et al., 2014; Goodman et al., 2008b]. Informally, one can think of the generative model as the definition of a prior, the conditioning statements as the definition of a likelihood, and the output of the program as samples from a posterior distribution. Their core philosophy is to decouple model specification and inference, the former corresponding to the user-specified program code, which implicitly defines a distribution on random variables, and the latter to an inference engine capable of operating on arbitrary programs. Removing the need for users to write inference algorithms significantly reduces the burden of developing new models and makes effective statistical methods accessible to non-experts. The inference/model abstraction barrier further means that some systems allow the definition of models that would be hard, or even impossible, to convey using conventional frameworks like graphical models.

Two key challenges for PPS are providing the syntax and semantics to allow easy definition of models, and in designing the solvers, i.e. inference engines, to provide effective inference for those models. In this chapter, we focus on the former of these, providing an introduction to probabilistic programming from a user's perspective. We will outline how it can be used for, and to extend, conventional Bayesian modeling and how it can also be used to reinterpret many computational simulation techniques not usually associated with Bayesian modeling in a more Bayesian mindset. We will mostly ignore the rather major issue of how to construct inference engines for PPS, returning to address this in Chapter 7. Instead, we will focus on how general purpose PPSs aim to provide the *flexibility* to define wide-ranging and potentially obscure models, the *expressivity* of a framework for model definition that is more in line with conventional scientific simulation than mainstream statistical approaches, and the *automation* to run any problem the user might write by decoupling model specification and inference. Together these characteristics produce a framework that allows researchers whose expertise lies elsewhere, to carry out powerful statistical analyses for their application specific tasks. This framework also aids in the development of both

inference algorithms and models for those within the machine learning and statistics communities, by removing many of the complications of one while developing the other.

We note that it will be necessary at times during this chapter to refer briefly to some Bayesian inference algorithms that will not be properly introduced until Chapters 5 and 6. We have situated this chapter before those partly in order to emphasize the point that one should not need an intricate knowledge of inference methods to *use* PPSs. Though it is difficult to introduce PPSs while completely omitting reference to inference methods, readers who are not familiar with them should be able to safely ignore which methods are referred to at a first pass, noting only that different inference algorithms have different requirements and sets of problems they perform well on, and thus that the design of a PPS is often intricately linked to the inference method(s) used.

4.1 Inverting Simulators

Though the use of Bayesian modeling through the sciences and engineering is widespread, it is still dwarfed by the use of simulators more generally. Some simulations are inherently probabilistic, such as many of those used in statistical physics [Landau and Binder, 2014], financial modeling [Jäckel, 2002], and weather prediction [Evensen, 1994]. Others are deterministic approximations of a truly stochastic world, such as lap time simulation for formula one cars [Perantoni and Limebeer, 2014] and finite element simulations for fluid dynamics [Versteeg and Malalasekera, 2007]. In many of these scenarios, real data is also available, but not in sufficient quantities that the carefully constructed simulations can be done away with entirely and replaced by a purely data-driven approach. Imagine the potential utility of general-purpose methods for incorporating real data into these simulators to improve them, without having to throw away the existing carefully constructed models. What about if we could even find methods for automatically inverting these simulators? Given a target lap time, we could return the best car setup; given observations of, and a simulator for, human behavior, we could learn about the underlying cognitive processes; given a climate change model and measurements, we might infer what the driving factors are.

An ambitious long-term aim of probabilistic programming is to solve such problems and to do so in an automated fashion so that it requires little or no statistical expertise on the behalf of the user, allowing for simple, widespread usage across many fields. The key realization is that stochastic simulators implicitly define probability distributions. They, therefore, represent generative models and using probabilistic programming we can reason about, and work with, these generative models explicitly. One typically thinks of Bayesian modeling in terms of the

prior and the posterior, but one can also think about it in terms of defining a joint distribution over both parameters and data, then fixing the latter to the actual observations to get a conditional distribution from this joint. Simulators implicitly define such joint distributions, with the outputs of the simulator corresponding to the data, and the inputs and internal variables the parameters. Probabilistic programming allows us to turn this on its head, using the same code as the original simulator, but instead providing the observed data as the input and then inverting the simulator through inference to learn about possible input parameters and other variables sampled during the program's forward execution. As well as the clear direct utility of allowing such inversion, this process also allows us to improve our simulator using real data, by calculating the posterior predictive distribution that incorporates both the original model and the information from the data.

To explain what we mean by inverting simulators more precisely, we will now consider the example of inferring Captchas [Mansinghka et al., 2013]. Even if the name is not familiar, everyone should hopefully have come across Captchas before when a website shows us an image, such as those in Figure 4.1a, and asks us to type the characters in that image to demonstrate we are not a robot. We now ask the question: how might we write an algorithm that breaks this Captcha by automatically predicting these characters directly from the image? In other words, can we build a robot that mimics a human on a task specifically designed to distinguish between the two? If we had access to large supply of training examples, i.e. character-image pairs, we could, of course, use an off-the-shelf discriminative algorithm: neural networks have been used to try and solve the problem in exactly this way with reasonable success [Von Ahn et al., 2008]. However, without access to an abundance of data, this is a rather challenging task and we will need to exploit our prior knowledge about the problem.

Doing this process in reverse, i.e. simulating a Captcha, on the other hand, is a substantially less daunting process. The true Captchas are actually generated by a simulator and so we can attempt to mimic this original simulation process. For example, as shown in Figure 4.1b we might first sample a number of characters, then a symbol for each character, apply manipulations such as rotations and warpings, simulate some obscuring lines or other noise, and finally render our simulated components into an image. Though admittedly it might take some time and effort to construct a high fidelity simulator that well matches the produced images, the technical requirements for doing this (other than possibly the final rendering) are minimal and so it could be carried out by most people with a reasonable degree of experience in scientific programming. The number of people able to write such a simulator should certainly be substantially larger than the number of people with sufficient experience in Bayesian modeling to construct an

(a) Examples of real Captchas **(b)** Example simulation of a Captcha

Figure 4.1: Solving Captchas by simulator inversion. **(a)** gives examples of real Facebook Captchas taken from Le et al. [2017b]. Here the corresponding generating strings going to clockwise from the top left are YPL9ceu, mJGD7zP, 9xPBS5k, 8VNARw, z7T7Jda, and GePHEz4. A user is asked to type in these strings when shown the corresponding image to show they are not a robot. **(b)** gives an example of the process of simulating Captchas taken by Le et al. [2017a]. Here we see that we can generate a Captcha by first simulating a series of characters, then simulating appropriate manipulations to those characters such as warping, rotating, and adding noise. Inverting this simulation process corresponds to an inference problem, where we want to find out the characters that lead to a particular image.

equivalent graphical model or direct mathematical formulation. The number of people with the expertise to then write an appropriate inference scheme for the problem is even smaller. In a PPS, writing this simulator and providing the data is all that is required. Given these, the PPS will carry out the inference required to invert the simulator automatically, inferring the characters from the image. More generally, we are estimating the inputs and internal variables of our simulator, given target values for the outputs.

There are two key factors to realizing our aim of inverting simulators. Firstly we need to provide a language which easily allows users to write down simulators and which has semantics that allows the compiler to extract an appropriate representation of the joint distribution. In other words, we need our language to be sufficiently general purpose and easy to use to not burden the user, while at the same time having syntax and semantics that ensure the corresponding joint distribution is well defined and can be converted into a form where we can run inference. Doing this will require the introduction of means for *conditioning* on data and of specially defined *random primitives* whose behavior can be controlled at run time, rather than just always sampling from the same predefined distribution as they would in an ordinary programming language. The latter can be thought of as defining terms in the prior and the former as defining terms in the likelihood as we will discuss in Section 4.3. For certain cases, one can alternatively think of conditioning as applying *constraints* to the program. For example, we can think of a probabilistic

program as defining a simulator and a set of constraints that must be satisfied; this is exactly how the PPS Church [Goodman et al., 2008b] is designed. An important distinction here though is between *hard* and *soft* conditioning. Hard conditioning is as per the conventional interpretation of a constraint – we condition on the fact that an event occurs exactly. Soft conditioning instead assigns a weight to the program based on the probability (or probability density) of a given event occurring. Though hard conditioning is a particular case of soft conditioning (for which the weight is either 1 or 0), one can, at least semantically, use it to specify a soft conditioning, say $p(Y = y|X = x)$, by sampling $Y \sim p(y|X = x)$ and then imposing the constraint $Y = y$. However, only supporting hard conditioning in a PPS is somewhat restrictive for practical use, as one cannot effectively condition on continuous data because there is zero probability of satisfying the resulting constraint.

The second key factor is that our language needs a general purpose inference(s) engine capable of working on any program the user writes. Bayesian models are fully defined by their joint distribution and the data. Therefore, once a user has written their simulator and provided the data, this uniquely defines a posterior and the only problem is in solving the resulting Bayesian inference. If we can now construct inference engines capable of working on arbitrary code, we can automate inference on any simulator or model the user defines, creating an abstraction barrier between model definition and drawing inferences from that model. We will discuss how this can be done at length in Chapter 7.

If we can construct a system that can successfully carry out these tasks, the huge potential applications this could provide should be clear. We would have a system where the user requires no expertise in inference or conventional Bayesian modeling in order to write application-specific models and have them solved automatically. Instead, they need only have the ability to write stochastic simulators for the process they wish to model, a skill possessed by most of the scientific community and many of those outside it as well. In a hypothetical future where scientists code all their simulators in extremely powerful PPSs, tasks such as inverting those simulators and improving the simulator by incorporating real data would be automated in the same way current compilers convert high-level coding languages to machine code. However, this ability is not completely hypothetical – many such problems can already be handled by existing systems. The challenge is improving and scaling such systems to deal effectively with more difficult and more wide-ranging models in a tractable manner. The need for such systems to work in an automated manner for a wide array of possible problems makes this a very difficult problem; after all, we are somewhat flaunting the no-free-lunch theorem. However, there is a key component

that provides hope that this may be possible: we have access to the target source code of the simulator itself, rather than needing to treat it as a black-box as is the case for, say, approximate Bayesian computation (ABC) methods [Csilléry et al., 2010]. Therefore maybe we can have our cake and eat it by using the source code itself to guide our algorithms, such that they behave in problem-specific ways. We will discuss how this can be done at length in Chapters 7 and 9.

4.2 Differing Approaches

Rather than being a clearly defined method, probabilistic programming is more of an umbrella term that covers a spectrum of different approaches, varying from inference toolboxes through to universal probabilistic programming languages (PPLs) that allow arbitrary probabilistic code to be written. Often there is a trade-off between efficiency and expressivity: the more restricted one makes the language, the more those restrictions can be exploited to improve the efficiency of the inference. This leads itself two distinct philosophies when developing a system. Firstly one can start with a particular inference algorithm and then design a system around making it as easy as possible to write models for which that inference algorithm is suitable. Secondly one can start with a general purpose language that allows as many models as possible to be written and then try to construct inference engines that are capable of working in such a general framework. Both approaches have their merits and drawbacks, with the distinction typically coming down to the intended use. We will now elucidate each approach more precisely.

4.2.1 Inference Driven Systems

Though there is a plethora of bespoke inference algorithms designed for particular models, the vast majority of these are based around a relatively small number of foundational methods such as importance sampling, sequential Monte Carlo, Metropolis-Hastings, Gibbs sampling, message passing, and variational inference (see Chapter 5). The extensive use of these core inference approaches throughout Bayesian statistics and machine learning means that it makes clear sense to write packages for automating them and which make it easy for the user to define appropriate graphical models for which the inference can be automated. This both improves the efficiency of constructing models and reduces barriers to entry by reducing the required expertise for users. This inference-first philosophy is taken by a number of successful PPSs and inference toolboxes (the distinguishing line between which can be a little blurry), a small number of which we now briefly outline.

BUGS (Bayesian inference Using Gibbs Sampling) [Spiegelhalter et al., 1996] and its extensions [Lunn et al., 2000; Plummer et al., 2003; Todeschini et al., 2014] allow finite DAGs

to be specified using declarative code or pictorially using a graphical user interface. These are converted to a form that is suitable for inference, the exact nature of which depends on the implementation, with the original work being based on Gibbs sampling.

Infer.Net [Minka et al., 2010] is modeling language for defining, and automating approximate inference in both DAGs and Markov random fields, using predominantly message-passing algorithms. Distributions are generally, though not exclusively, restricted to be exponential families. Branching (i.e. `if`) is allowed but requires enumeration of all possible paths at run time.

LibBi [Murray, 2013] is a package for doing Bayesian inference for state-space models, using particle-based inference methods (see Chapter 6). It has a strong focus on scalable computation, providing support for multi-core architectures and graphics processing units.

PyMC3 [Salvatier et al., 2016] is a python framework for carrying out MCMC and variational inference, using Theano [Bergstra et al., 2010] to calculate the gradients required by some inference methods.

Stan [Carpenter et al., 2015] is a PPS with interfaces to many difference languages and a focus on performing Hamiltonian Monte Carlo inference [Duane et al., 1987; Hoffman and Gelman, 2014], though other inference methods such as variational inference are also provided [Kucukelbir et al., 2015]. As with PyMC3, automatic differentiation [Baydin et al., 2015] is used to calculate the required gradients. The need to take derivatives means that there is limited support for discrete variables or branching.

Edward [Tran et al., 2016] is a PPS based around Tensorflow [Abadi et al., 2016] that supports directed graphical models, neural networks, and combinations of both. It supports both Monte Carlo and variational inference methods (again using automatic differentiation) and has a strong emphasis on model criticism.

These systems do not allow users to write models that would be difficult (at least for an expert) to code without a PPS – in general, they can all be thought of as defining a graphical model or sometimes a factor graph – but they offer substantial utility through ease of model exposition and automating inference.

4.2.2 Universal Probabilistic Programming

As useful as these inference-driven systems are, they do not fit very well with the notion of inverting simulators we introduced in Section 4.1. They are still closely tied to graphical models and are more toolboxes for streamlining the Bayesian modeling process than a means of writing models that would be problematic to define by conventional means. Achieving our long-term

ambitious aim of making general purpose systems for conducting inference on arbitrary simulators will require us to take a somewhat different approach that instead starts with a general-purpose language and then attempts to design inference algorithms capable of working on arbitrary models and code. It will be necessary for such systems to support models where the set of random variables is dynamically typed, such that it is possible to write programs in which this set, and thus potentially the number of random variables, differs from execution to execution. To avoid hindering the user or restricting the models which can be defined, it will important to allow things such as branching, recursion, higher-order functions, conditional existence of variables, and arbitrary deterministic functions. Ideally, we would like to provide no restrictions on the code that the user can write, except for eliminating programs that do not define valid probability distributions, such as those that have a non-zero probability of never terminating. In practice catching such invalid cases can be hard or even impossible and so many systems actually adopt a philosophy of applying no restrictions, such that it is perfectly possible to define invalid models. General purpose PPSs actually bring up new theoretical questions about what constitutes a valid probability model [Heunen et al., 2017] and the set of valid definable models is a strict superset of those definable by graphical models for many systems [Goodman, 2013].

In the rest of this thesis, we will predominantly focus on these *universal* PPLs [Goodman et al., 2008a; Staton et al., 2016], so-called because they are based on *Turing complete* languages that can specify any computable distribution [Goodman, 2013]. For our purposes, we further refine this definition to systems that also allow specification for any computable conditioning. We will regularly use the universal PPL Anglican [Wood et al., 2014] as a reference, an introduction to which is provided in Section 4.4. Here we will briefly discuss some other prominent higher order PPLs.

Church is a PPL based on Scheme [Goodman et al., 2008a]. The original seminal paper and accompanying system form a foundation on which many of the prominent existing systems are built, through its demonstration that higher-order probabilistic programs define valid probability models, even in the presence of infinite recursion. However, Church predominantly only allows hard conditioning,[1] namely a model in Church comprises of a generative sampler and a separate predicate procedure which returns true if the desired conditions are satisfied. In addition to the aforementioned issues of hard conditioning, this complete separation of the generative process and the conditioning can also be wasteful in not allowing the structure of a model to be exploited

[1] Some very limited support for soft-conditioning is provided in current implementations through a "noisy equals" that equates to a Gaussian likelihood.

(see Chapters 6 and 7). Later systems therefore mostly allow soft conditioning statements to be interleaved through the generative progress (in an analogous manner to likelihood terms), increasing the range of (solvable) models that can be encoded and the potential efficiency of inference algorithms. Inference in Church (and its direct derivatives) is typically carried out using either rejection sampling or MCMC. Church places a particularly strong emphasis on the ability to carry out *nested inference*, something we will look into in depth in Chapter 10.

Venture [Mansinghka et al., 2014] is a probabilistic programming platform providing a flexible system for both specification of models and inference methods. It has a strong emphasis on being extensible and for allowing the hosting of external applications. For example, it allows the user to provide proposals for the inference engine or reprogram the inference strategy entirely. Venture is predominantly used via the VentureScript PPL [Mansinghka et al., 2014].

WebPPL [Goodman and Stuhlmüller, 2014] is a PPL built using a purely functional subset of Javascript, conveniently allowing for embedding in web pages. It combines the ability to write a generative process using sampling statements and to add in likelihood terms through a `factor` primitive that is analogous to the **`observe`** primitive that we will introduce in Section 4.3.1. At its back end, WebPPL provides a number of different inference algorithms, such as SMC and MCMC methods.

Pyro [Bingham et al., 2017] and ProbTorch [Siddharth et al., 2017] are two early-stage PPLs based on PyTorch Paszke et al. [2017] that share many design characteristics. Similarly to Edward, they allow modeling of neural networks, but differ in that they construct gradients dynamically, allowing things such as stochastic control and recursion.

The price for the expressivity of these general purpose systems is a substantial extra burden on the inference engine, as we will discuss in Chapter 7. In general, inference methods for such systems must be formulated in such a manner that they are applicable to models where the density function is intractable and can only be evaluated during forwards simulation of the program. For example, it may not be possible to know if a variable is continuous or discrete except by running the program, while some variables will only exist conditioned on the values of others. This required generality of the inference engine will naturally lead to a drop in performance compared to custom written inference code, but this is often a price worth paying for generality and automation, particularly when considering models that would be challenging to express, let alone do inference in, using more conventional frameworks.

4.3 Bayesian Models as Program Code

In Section 4.1 we showed how one can think of PPS as inverting simulators, predicting internal variables given the outputs. In this section, we will take a different perspective and show how we can translate Bayesian modeling into the framework of program code. In Chapter 3 we showed how a Bayesian model is defined by a prior over parameters and a likelihood function for those parameters given the data. This viewpoint will mostly translate into the probabilistic programming setting by equating between the prior and sampling statements and between the likelihood and conditioning statements. However, in Section 4.3.2 we will why explain why this is not always true and suggest a formulation, distinct to that commonly used in the literature, for how the density implied by a query can be correctly represented.

A key point to note throughout this section is that probabilistic programs define models rather than procedures. We refer to these models as *queries* [Goodman et al., 2008b] which are analogous to functions in a ordinary language. In a standard programming language, functions take in inputs and then run through a series of commands in order until termination is reached.[2] Likewise, random sampling statements like `rand`, `randn` etc, make a single independent draw from the same distribution each time they appear in the execution trace. Neither is the case for a probabilistic program query. Instead a query defines a model which is compiled to a form that can be interpreted by an inference engine which then outputs some characterization of the posterior such as a series of samples. Perhaps the easiest way to think about how a probabilistic program language works (though not necessarily what happens for all systems) is that the query is, or sometimes parts of the query are, run many times and the exact behavior of this running is control by the inference engine.

4.3.1 A Simplified Probabilistic Programming Setup

We first consider the case of constructing a restricted example PPL. We emphasize that this is by no means the only setup one can use, with design choices made in the interest of exposition. We will presume that our PPL has no branching (i.e. there are no `if` statements or equivalent), recursion, or memoization (i.e. functions are always re-evaluated from scratch when called); is first order (i.e. variables cannot be functions); and that it does not allow any conditioning on internally sampled variables. We will give our language two special constructs, **`sample`** and **`observe`**, between which the distribution of the query is defined. Informally, **`sample`** will be

[2] In functional programming languages operations are not necessary carried out in the order they are defined, but it still holds that the function takes inputs carries out a series of actions until the desired output is calculated.

used to specify terms in the prior and **observe** terms in the likelihood. More precisely, **sample** will be used to make random draws $x_t \sim f_t(x_t|\Xi_t)$, where Ξ_t is a subset of the variables in scope at the point of sampling, and **observe** will use to condition on data $g_s(y_s|\Lambda_s)$ with Λ_s defined in the same way as Ξ_t. For our inference, it will be necessary to control the sampling and so we define the syntax of **sample** to take a *distribution object* as its only input and for **observe** to take a distribution object and an observation as input. We further define each distribution object as containing a sampling procedure and a density function that can be evaluated exactly.[3] Our language will be provided with a number of *elementary random procedures* in the form of distribution classes for common sampling distributions such as the normal and Poisson distributions, but will also provide the ability for users to define their own distribution classes. These classes allow a distribution object to be constructed when provided with the required parameters, such that the distribution is fully defined before being passed to a **sample** or **observe**. We complete our syntactic definition of **sample** and **observe** by defining them to return a sample and `nil` respectively. We will presume here and throughout that, other than the effects of **sample** and **observe**, functions in our PPL are *pure*, such that they always provide the same outputs when called with the same inputs. This restriction naturally means that queries should not have any random components other than dictated by **sample** and **observe**, but also suggests that they should be free from side effects such as modifications of global variables.

For our simplified setup, we distinguish between two types of inputs to our queries: external parameters ϕ and data $y_{1:S}$. The external parameters are defined as the inputs that are not "observed" at any point but can affect the conditioning through Ξ_t and Λ_s. We presume that the data terms, defined as the inputs we observe, appear in neither Ξ_t or Λ_s. We now define both **sample** and **observe** from the probability model perspective as adding a factor to the joint distribution which is therefore given by

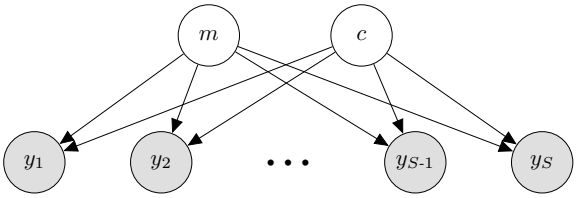
$$p(x_{1:T}, y_{1:S}|\phi) = \prod_{t=1}^{T} f_t(x_t|\Xi_t) \prod_{s=1}^{S} g_s(y_s|\Lambda_s). \tag{4.1}$$

The two vary in whether they define a new random variable (**sample**) or affect the probability of the execution given particular instances of the other random variables (**observe**). Our presumptions for this simplified setup that no y_s terms are present in the Ξ_t or Λ_s and that we do not condition on internally sampled variables, means that the product of the **sample**

[3] It will actually often be possible to use incomplete distribution objects for both **sample** and **observe** that only contain the sampler and the density function respectively. The former can be permitted by using inference algorithms that use the prior as the proposal, such that only the sampler will be required. The latter is sufficient for the vast majority of scenarios as, unless one is carrying out amortization [Paige and Wood, 2016; Le et al., 2017a], the data is fixed and does not need to be sampled.

Inputs: Student-t degrees of freedom ν, error scale σ, data $y_{1:S} = \{u_s, v_s\}_{s=1}^S$

1: $m \leftarrow$ **sample** (**normal** (0,1))
2: $c \leftarrow$ **sample** (**normal** (0,1))
3: `obs-dist` $\leftarrow$ **student-t** (ν)
4: **for** $s = 1, \ldots, S$ **do**
5: $\quad d \leftarrow (v_s - mu_s - c)/\sigma$
6: $\quad$ **observe** (`obs-dist`, d)
7: **end for**
8: **return** $m,\ c$

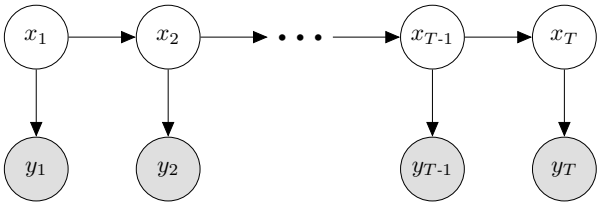


$$p(m, c, y_{1:S}|\nu, \sigma) = \mathcal{N}(m; 0, 1)\ \mathcal{N}(c; 0, 1) \prod_{s=1}^{S} \text{STUDENT-T}\left(\frac{v_s - mu_s - c}{\sigma}; \nu\right)$$

(a) Bayesian linear regression model with student-t likelihood, namely $v_s = mu_s + c + \sigma\epsilon_s$ where $\epsilon_s \sim$ STUDENT-T(ν). We presume that the scaling of the error σ and the number of degrees of freedom ν are fixed input parameters (i.e. $\phi = \{\nu, \sigma\}$), that our fixed data is $y_{1:S} = \{u_s, v_s\}_{t=1}^S$, and that we are trying to infer the slope m and intercept c (we thus have $x_1 = m$, $x_2 = c$ in our general notation), both of which are assigned a unit Gaussian as a prior. Our query first samples m and c (note that **normal** (0,1) generates a unit Gaussian distribution object) and constructs a student-t distribution object `obs-dist`. It then cycles over each datapoint and observes $(v_s - mu_s - c)/\sigma$ using `obs-dist`, before finally returning m and c as outputs. Note that if we instead wished to directly predict the outputs at some untested inputs points $u_{S+1:S+n}$ then we could predict these anywhere in the query (after m and c have been defined) and return them as outputs along with, or instead of, m and c.

Inputs: Transition std-dev σ, output shape α, output rate β, data $y_{1:T}$

1: $x_0 \leftarrow 0$
2: `tr-dist` $\leftarrow$ **normal** $(0, \sigma)$
3: `obs-dist` $\leftarrow$ **gamma** (α, β)
4: **for** $t = 1, \ldots, T$ **do**
5: $\quad x_t \leftarrow x_{t-1}+$**sample** (`tr-dist`)
6: $\quad$ **observe** (`obs-dist`, $y_t - x_t$)
7: $\quad z_t \leftarrow \mathbb{I}(x_t > 4)$
8: **end for**
9: **return** $z_{1:T}$



$$p(x_{1:T}, y_{1:T}|\sigma, \alpha, \beta) = \mathcal{N}(x_1; 0, \sigma^2)\ \text{GAMMA}(y_1 - x_1; \alpha, \beta) \prod_{t=2}^{T} \mathcal{N}(x_t - x_{t-1}; 0, \sigma^2)\ \text{GAMMA}(y_t - x_t; \alpha, \beta)$$

(b) State-space model with Gaussian transition and Gamma emission distributions. It is a form of the HMM model given in (3.17) with $p(x_1) = \mathcal{N}(x_1; 0, \sigma^2)$, $p(x_t|x_{t-1}) = \mathcal{N}(x_t - x_{t-1}; 0, \sigma^2)$, and $p(y_t|x_t) = \text{GAMMA}(y_t - x_t; \alpha, \beta)$ with shape parameter α and scale parameter β. We assume that the input parameters $\phi = \{\sigma, \alpha, \beta\}$ are fixed and we want to sample from $p(z_{1:T}|y_{1:T}, \phi)$ given data $y_{1:T}$, where each z_t is an indicator for if x_t exceeds a threshold of 4. Our query, exploiting the equivalence between $p(x_1)$ and $p(x_t|x_{t-1} = 0)$, first initializes $x_0 = 0$ and creates distribution objects for the transitions `tr-dist` and emissions `obs-dist`. It then loops over time steps, sampling each x_t given x_{t-1}, observing y_t given x_t, and deterministically calculating z_t. Finally the $z_{1:T}$ are returned as the desired output.

Figure 4.2: Example pseudo-queries for our simplified probabilistic programming setup with corresponding graphical models and joint distributions.

terms correspond exactly to our prior $\prod_{t=1}^{T} f_t(x_t|\Xi_t) =: p(x_{1:T}|\phi)$ and that the product of the **observe** terms corresponds exactly to our likelihood $\prod_{s=1}^{S} g_s(y_s|\Lambda_s) =: p(y_{1:S}|x_{1:T}, \phi)$. Consequently, for our simplified setup, each query defines a finite DAG (see Section 3.3) where the conditional relationships are defined through the definitions of f_t and g_s. This breakdown into a prior and likelihood and the equivalence to graphical models will not hold in the more general cases we consider later. Our aim will be to perform inference to provide a characterization of $p(x_{1:T}|y_{1:S}, \phi)$ (or the posterior for some deterministic mapping of $x_{1:T}$), typically in the form of (approximate) samples. Figure 4.2 shows two example queries along with the corresponding graphical models and joint distributions they define.

Other than **sample** and **observe** statements, the rest of our query is, by construction, totally deterministic. Therefore, though it may contain random variables other than $x_{1:T}$, these random variables are deterministic functions of the "raw" random draws $x_{1:T}$ and inputs ϕ and $y_{1:S}$. We can therefore define the outputs of our query as $\Omega := h(x_{1:T}, y_{1:S}, \phi)$ for some deterministic function h. As we explained in Section 2.6, this change of variables means that the density function on Ω, $p(\Omega|y_{1:S}, \phi)$ can have a different form to the posterior implied by our query, namely $p(x_{1:T}|y_{1:S}, \phi)$. Though this is a serious complication in the context of optimization (we may not in general be able to find $\arg\max_\Omega p(\Omega|y_{1:S}, \phi)$ or even evaluate $p(\Omega|y_{1:S}, \phi)$ exactly), it is perfectly acceptable in the context of calculating expectations as

$$\int f(\Omega)p(\Omega|y_{1:S}, \phi)d\Omega = \int f(h(x_{1:T}, y_{1:S}, \phi))p(x_{1:T}|y_{1:S}, \phi)dx_{1:T} \tag{4.2}$$

for implicitly defined reference measures $d\Omega$ and $dx_{1:T}$. One consequence of this is that we can express any expectation calculated by our query as an expectation over $p(x_{1:T}|y_{1:S}, \phi)$ which is fully defined by the joint (4.1). Another is that, provided we are not worried about carrying out optimization, we do not need to explicitly worry about the implicit measures defined by the definition of our query, other than any potential effects on the inference scheme and the assumption that suitable measures exist [Staton et al., 2016]. In particular, if our aim is to generate samples from $p(\Omega|y_{1:S}, \phi)$ then we can simply generate samples from $p(x_{1:T}|y_{1:S}, \phi)$ and deterministically convert each sample to the space of Ω. In other words, our inference engine does not need to worry about the consequences of changes of variables if our intended output is just a sequence of samples.

An important point to note is that (4.1) shows that all of our **sample** and **observe** statements are exchangeable, in the sense that their order can be moved around and still define the same joint distribution, up to restrictions about all the required variables existing and being in scope.

For example, if all variables remain in scope and are not redefined, we can generally move all our `observe` statements to the end of the query without changing the joint distribution. Therefore the query given in Figure 4.2b would define the same model if all the x_t were sampled upfront before making any observations. Nonetheless, the position of the `observe` statements can often be important from the perspective of the performance of the inference engine. This exchangeability result will carry over to the non-simplified case.

4.3.2 A General Probabilistic Programming Setup

Perhaps surprisingly, we do not need to do anything to our language to extend it to a universal PPL other than to relax a number of the restrictions made for our simplified case. Namely, we will allow branching, higher order functions, (potentially infinite) recursion, stochastic memoization [Goodman et al., 2008b], conditioning on internally sampled variables, and the use of the "data" inputs $y_{1:T}$ in the definition of our generate model (instead of just allowing them to be observed). One assumption we will make, here and throughout the rest of this thesis, is that our program terminates with probability 1, asserting that any program that does not satisfy this assumption does not define a valid model. We will maintain the syntax of our simplified setup, along with the assumption that functions are pure other than the effect of `sample` and `observe`. An important point to note though for why such a language is universal, is that arbitrary distributions with countable parameters can be defined through a series of uniform $[0, 1]$ draws followed by an arbitrary deterministic mapping – after all, this is effectively how all probability distributions are defined from a measure theoretic point of view.[4]

Despite their ostensibly modest nature, these generalizations will have a substantial effect on the intuitions relating our universal PPL to the Bayesian framework, the range of models we can define, and the difficulty of performing general purpose inference. For example, as y_s terms can appear in the Ξ_t terms, it can be the case that $p(x_{1:T}|\phi) \neq \prod_{t=1}^{T} f(x_t|\Xi_t)$, such that the latter no longer explicitly corresponds to a conventional definition of a prior. Some variables may change type (e.g. between continuous and discrete) or even not exist depending on the value of other variables. The number of variables may change from one execution to the next and it could even be the case that the number of latent variables is unbounded provided that any possible execution has a finite number of variables is with probability 1, such as is the case for certain Bayesian non-parametric models such as the Dirichlet process [Ferguson, 1973;

[4]Interestingly, this viewpoint breaks down for distributions over functions for which measure theory itself somewhat breaks down Heunen et al. [2017]. As such, universal PPSs actually go beyond the standard measure-theoretic view of probability.

Teh, 2011; Bloem-Reddy et al., 2017]. Note that the number **sample** and **observe** statements lexically defined within our query must, for obvious reasons, be finite, but recursion or looping may mean that they are evaluated an infinite number of times. It is also possible for a variable within a query to itself encode an infinite number of parameters, for example, one might include a Gaussian process within the query (see Section 8.2). These complications make it difficult to mathematically reason about the joint distribution defined by a query in a universal PPS, let alone reason about its breakdown into a prior and a likelihood, or represent the query using a graphical model (which might actually not even be possible).

One way to conceptually overcome these difficulties is to reason in terms of *execution paths*. Even though we might not know upfront which **sample** and **observe** statements are invoked by a particular query output, if we execute the query in order, the draws of the **sample** statements made thus far in an *execution trace* uniquely identify which **sample** statement will be invoked next and the value of all the variables up to that **sample** statement. In other words, given the outcome of the first t evaluated **sample** statements, which **sample** statement corresponds to the $(t+1)$-th to be evaluated is uniquely defined and the query up to that **sample** statement is completely deterministic, including which path to take at each `if` statement, which variables are defined and their values, and the probability arising from the **observe** conditioning statements.[5] Thus although the distribution of the query is not necessarily statically determinable, it can still be *evaluated through execution* as each **sample** statement provides the information we require, in addition to the information from the previous **sample** statements, to deterministically evaluate up to the next **sample** statement. Consequently, we can evaluate the probability of any particular trace as we run it, even though it might be challenging to evaluate the probability of a predefined configuration of the variables.

Using this idea of execution paths, we can define an expression for the conditional probability a query defines, albeit in a complex and somewhat abstract manner that only really retains meaning in our calculation through evaluation mindset, by defining the probability of a particular trace. First let λ denote the value of all the inputs to the query (including both parameters and data as we will now have no explicit distinction between the two). Further let n_x and n_y be the number of **sample** and **observe** statements respectively invoked by the trace, noting that n_x and n_y may themselves be random variables and could potentially be unbounded (e.g. n_x could follow a Poisson distribution). We can now define, for any given execution, $x_{1:n_x} =$

[5] Note that our inference algorithm might induce probabilistic behavior at **observe** statements, but we can safely ignore this in terms of defining the distribution represented by the query.

$x_1, \ldots, x_{n_x}$ and $y_{1:n_y} = y_1, \ldots, y_{n_y}$ respectively as the outputs of our n_x **sample** statements and observation inputs for our n_y **observe** statements. To avoid complications regarding variable reassignment, we will not consider the x_j as being variables in our program, such that calling $a \leftarrow$**sample**`(dist)` effectively creates an internal protected variable x_j (whose value can never be reassigned) and then immediately assigns the value of x_j to a. All variables generated before x_j are a deterministic function of $x_{1:j-1}$ and λ. Therefore, even though all variables in our program might be random, including the observations y_k, all are deterministically calculable given $x_{1:n_x}$ and λ. Even n_x itself is a deterministic function of the output of the **sample** statements, as $\{x_{1:j}, \lambda\}$ deterministically dictates whether there will be any further **sample** statements encountered. Consequently, in the same way we only needed to worry about $x_{1:n_x}$ to reason about the distribution over the program outputs for our simplified case, we can reason about the distribution on all variables in program (for a given λ) through considering $x_{1:n_x}$ in our general case. We can thus think of our trace as being defined by $x_{1:n_x}$.

We continue by defining $f_1, \ldots, f_{n_s}$ and $g_1, \ldots, g_{n_o}$ respectively as the densities (with respect to an implicitly defined reference measure) associated with the n_s **sample** and n_o **observe** statements defined lexically within the program (i.e. the distinct **sample** and **observe** statements appearing anywhere in the raw program code), noting that n_s and n_o are *not* random variables. To express these densities, we use the notation $f_i(x_j|\eta_j)$ to indicate the density of lexical **sample** statement i returning outputs x_j when provided with distribution object η_j, which will typically be a random variable itself. Note here that parameters of the density are encoded through the distribution object, so, for example, we can think of **sample** (**normal** $(\mu, 1)$) for a random variable μ as first creating the random distribution object $\eta_j =$**normal** $(\mu, 1)$ before passing this to the **sample** call. We similarly use the notation $g_i(y_k|\psi_k)$ to express the density of using lexical **observe** g_i to observe output y_k with the (random) distribution object ψ_k. We define $a_j \in \{1, \ldots, n_s\},\ \forall j \in \{1, \ldots, n_x\}$ and $b_k \in \{1, \ldots, n_o\},\ \forall k \in \{1, \ldots, n_y\}$ as the random variables respectively used to index which of the lexical **sample** and **observe** statements correspond to the j^{th} and k^{th} execution trace **sample** and **observe** statements.

To encapsulates the notion of a valid trace, i.e. a $x_{1:n_x}$ that can be generated by the program and for which all the density terms are well-defined, we introduce the deterministic boolean function $\mathcal{B}(x_{1:n_x}, \lambda)$ which returns 1 if the trace is valid and 0 otherwise. This is something we do not need to worry about if we take an evaluation based inference approach, whereby we rely on methods that only propose from the generative model, but it is necessary to ensure that the density is well defined if we try to evaluate it at a predetermined point, chosen externally to

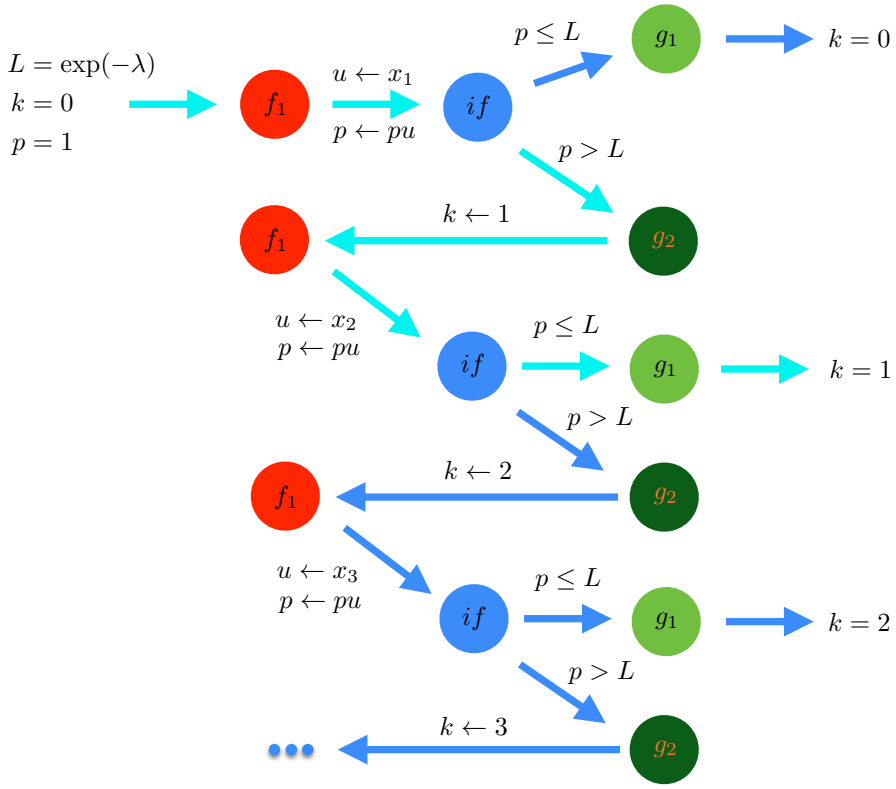
the program. For example, $\mathcal{B}(x_{1:n_x}, \lambda)$ is necessary to ensure that $x_{1:n_x}$ is a "complete" trace: it may be that a certain $x_{1:n_x}$ implies that further **sample** statements are still to be invoked after the n_x^{th}, in which case the trace is not valid as these additional outputs are undefined. For example, if the program defines the distribution $\mathcal{N}(x_1; 0, 1)\mathcal{N}(x_2; 0, 1)\mathcal{N}(4; x_2, x_1)$, then the trace $x_1 = 3.2$ is incomplete. Similarly, we can use $\mathcal{B}(x_{1:n_x}, \lambda)$ to encode that in our example $P(n_x > 2) = 0$. Meanwhile, $\mathcal{B}(x_{1:n_x}, \lambda)$ also ensures that all terms within the trace probability are well defined. For example, an invalid trace might provide parameters of the wrong type to one of the distribution objects, meaning that density of that distribution object is undefined.

We are now finally ready to define the conditional distribution on the trace $\mathcal{T}$ implied by our query as $p(\mathcal{T} = x_{1:n_x}|\lambda) \propto \gamma(x_{1:n_x}, \lambda)$ where

$$\gamma(x_{1:n_x}, \lambda) = \begin{cases} \prod_{j=1}^{n_x} f_{a_j}(x_j|\eta_j) \prod_{k=1}^{n_y} g_{b_k}(y_k|\psi_k) & \text{if} \quad \mathcal{B}(x_{1:n_x}, \lambda) = 1 \\ 0 \quad \text{otherwise} \end{cases}, \tag{4.3}$$

remembering that although the a_j, η_j, etc are random variables, they are all deterministically calculable from $x_{1:n_x}$ for a given query and λ. As a consequence, our program implies a well defined, normalized, conditional distribution, or "posterior", on the produced traces (presuming the normalization constant is finite and non-zero). Note that this definition is explicitly on outcomes of the trace and so, in general, $p(\mathcal{T} = x_{1:j}|\lambda) \neq \int p(\mathcal{T} = x_{1:n_x}|\lambda) dx_{j+1:n_x}$. In fact, because $x_{1:j}$ deterministically dictates whether x_{j+1} exists, it is not possible to have $\mathcal{B}(x_{1:n_x}, \lambda) = \mathcal{B}(x_{1:j}, \lambda) = 1$, i.e. we cannot have that both $x_{1:j}$ and $\{x_{1:j}, x_{j+1:n_x}\}$ are complete traces. Consequently, it is always necessary that at least one of $p(\mathcal{T} = x_{1:j}|\lambda)$ and $p(\mathcal{T} = x_{1:n_x}|\lambda)$ are equal to zero. Note though that we can still define a program that places non-zero weight on, for example, both the *outputs* $[2.3, 3.5]$ and $[2.3, 3.5, 1.2]$, remembering that $x_{1:n_x}$ relates to the *raw draws* from the **sample** statements.

An illustrative example for calculating the probability of a program through execution traces is shown in Figure 4.3. This corresponds to a problem that cannot not be expressed using our simplified PPL or a graphical model, but which is simple to write in our universal framework. Imagine we want to calculate the unnormalized trace probability $\gamma(x_{1:n_x} = [0.2, 0.07], \lambda = 4)$ which we can do by stepping through the program and accumulating terms (for reference we will be following the cyan path in Figure 4.3a). On our path we first hit $f_1(x_1|\eta_1) = \text{UNIFORM}(0.2; 0, 1) = 1$, we fix $u \leftarrow x_1$ and $p \leftarrow 1 \cdot u$, and test if $p = 0.2 \leq \exp(-4) \approx 0.0183$. This gives false and so we do not break the while loop, instead encountering $g_2(y_1|\psi_1) = \text{BERNOULLI}(1; 0.2) = 0.2$ and so our running value for $\gamma(x_{1:n_x} = [0.2, 0.07], \lambda = 4)$ is $1 \times 0.2 = 0.2$. Updating $k \leftarrow 1$ and going back to the start of the while loop we encounter

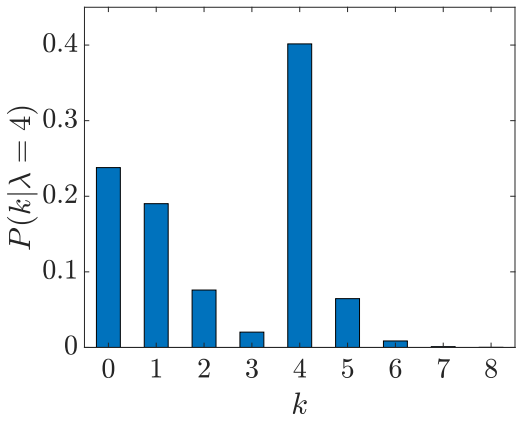


(a) Possible traces for our query. **sample** statements are shown in red, **observe** statements are shown in shades of green, and deterministic computations (given the outputs of the **sample** statements) are shown in shades of blue. Subscripts for the sample and observe statements show the lexical index (i.e. $a_j \in \{1\}$ and $b_k \in \{1, 2\}$), as do the different shades. The cyan arrows correspond to a particular execution path which gives output $k = 1$ and has $n_x = 2$, $n_y = 2$, $b_1 = 2$, and $b_2 = 1$. Valid traces for this particular path must have $x_1 > \exp(-\lambda)$ and $x_2 \leq \exp(-\lambda)/x_1$.

Inputs: Event rate λ

1: $L \leftarrow \exp(-\lambda),\ k \leftarrow 0,\ p \leftarrow 1$
2: **while** $p > L$ **do**
3: $\quad u \leftarrow$ **sample** (**uniform** (0,1))
4: $\quad p \leftarrow pu$
5: $\quad$ **if** $p \leq L$ **then**; **break while**; **end if**
6: $\quad$ **observe** (**bernoulli** (0.2), 1)
7: $\quad k \leftarrow k + 1$
8: **end while**
9: **observe** (**bernoulli** (0.99), $\mathbb{I}(k{>}3)$)
10: **return** k

(b) Query code for warped Poisson sampler



(c) Conditional distribution on k for $\lambda = 4$.

Figure 4.3: Demonstration of stochastic execution traces using a warper Poisson sampler, adapted from [Paige, 2016, Figure 3.3]. The query defined in **(b)** would output k as per a Poisson distribution with event rate λ if it were not for the **observe** statements. As shown in **(c)** though, these **observe** statements warp the distribution to give a lighter tail while discouraging $k \leq 3$. Here both $n_x \in \mathbb{N}^+$ and $n_y \in \mathbb{N}^+$ depend on the trace and are unbounded. Which **observe** we see first is also probabilistic, while it some traces will be invalid, e.g. $x_{1:n_x} = [1, 0.5]$ as $x_1 = 1$ indicates there will be only a single **sample** encountered. Nonetheless, we can calculate the probability of a trace by following its path to completion while accumulating **sample** and **observe** factors.

$f_1(x_2|\eta_2) = \text{UNIFORM}(0.07; 0, 1) = 1$, so our running score is $1 \times 0.2 \times 1 = 0.2$. Reassigning u and p we see that $p = 0.014 \leq \exp(-4)$ is now true and so we take the opposite branch to before with our if statement, thus breaking the while loop. To finish the program we pass through $g_1(y_2|\psi_2) = \text{BERNOULLI}(k > 3; 0.99) = 0.01$ giving the final unnormalized density of $\gamma(x_{1:n_x} = [0.2, 0.07], \lambda = 4) = 0.01 \times 0.2 = 0.002$ and an output of $k = 1$. We finish by checking that our trace was valid, which we can easily see is the case because no terms were undefined and we generated the correct number of sample outputs (i.e. $n_x = 2$ as required).

An important point of note is that, in general, $\gamma(x_{1:n_x}, \lambda)$ is not a normalized joint distribution. This is firstly, and most obviously, because λ might contain terms, referred to as ϕ before, that are not observed and so have no implied density. Secondly, and more critically, even if $\phi = \emptyset$, $\gamma(x_{1:n_x}, \lambda)$ is not necessarily correctly normalized, because of the ability to observe sampled variables and condition the **sample** statements on the observations. As a simple example, our model might consist of a $x_1 \leftarrow$**sample** (**normal** (0,1)) term followed by an **observe** (**normal** $(-1, 2, x_1)$) term. This does not directly define any properly normalized joint distribution on any particular variables (noting that $\mathcal{N}(x_1; 0, 1)\mathcal{N}(x_1; 0, 2)$ is an unnormalized distribution with only the variable x_1). Consequently, there is no means of writing down a normalized joint distribution for a general query in our universal PPL setup in closed form – we might actually need to empirically estimate the normalization constant to evaluate the joint. This actually steps outside the conventional Bayesian modeling framework and raises a number of interesting theoretical questions. However, from a practical perspective, we can note that provided the implicitly defined normalization constant is finite, the query also implicitly defines a correctly normalized conditional distribution (noting that $\gamma(x_{1:n_x}, \lambda) \geq 0$). This is analogous to knowing the joint but not the posterior in Bayesian inference, though it is not exactly equivalent because the normalization constant is no longer the marginal likelihood.

For a simple example of why it is important to be able to define models up to an unnormalized joint distribution, consider a model where a and b are each sampled from discrete distributions and we want to condition on the value of a and b being equal (see for example Figure 4.7 in the next section). Here the combination of the sample statements and the observation that the two are equal clearly does not lead to a correctly normalized joint (we do not even really have a conventional notion of a likelihood), but it clearly defines an unnormalized conditional distribution as

$$P(a, b|\mathbb{I}(a = b)) = \frac{P(a)P(b|a)\mathbb{I}(a = b)}{\sum_a \sum_b P(a)P(b|a)\mathbb{I}(a = b)}.$$

In such cases where our observation is a hard constraint, we can view the normalizing constant for the conditional defined by the query as the probability of our constraint being satisfied. For more general queries of discrete variables, we might have, for example,

$$P(x|\lambda) \propto f(x|\lambda)g(y = \kappa(x, \lambda)|x, \lambda)$$

for some deterministic function κ. Here we can view the marginalization as being the probability of the event $y = \kappa(x, \lambda)$ under the joint $P(x, y) = f(x|\lambda)g(y|x, \lambda)$. For our previous example we have $x := a$, $y := b$, $\kappa(a, \lambda) := a$, $f(a) := p(a)$, and $g(b = a) := p(b = a|a)$. The same intuition applies to continuous cases where we now have the density of the event $y = \kappa(x, \lambda)$. We can also think of $f(x|\lambda)g(\kappa(x, \lambda)|x, \lambda)$ as defining an unnormalized distribution on x whose normalization constant is $\int f(x|\lambda)g(\kappa(x, \lambda)|x, \lambda)dx$. In general, our normalization constant will be a combination of conventional marginal likelihood terms and contributions from these "doubly defined" terms. We refer to this normalization constant as the *partition function*[6] and note that it is given by

$$\begin{aligned} Z(\lambda) &= \mathbb{E}\left[\prod_{k=1}^{n_y} g_{b_k}(y_k|\psi_k)\middle|\lambda\right] = \int \gamma(x_{1:n_x}, \lambda)dx_{1:n_x} \\ &= \int_{x_{1:n_x} \in \{X:\mathcal{B}(X,\lambda)=1\}} \prod_{j=1}^{n_x} f_{a_j}(x_j|\eta_j) \prod_{k=1}^{n_y} g_{b_k}(y_k|\psi_k)dx_{1:n_x} \end{aligned} \tag{4.4}$$

where the expectation is under running the query forwards (i.e. the distribution implied by simulating from an equivalent query with all the `observe` statements removed) and all terms in the integral are deterministically calculable for the query given λ and $x_{1:n_x}$. For our query to represent a well-defined conditional distribution, it is necessary to have $0 < Z(\lambda) < \infty$. Although $Z(\lambda)$ does not correspond exactly to a marginal likelihood, we can still think of it in terms of representing a *model evidence* in the same way, it just might not be correctly normalized density in the same way a marginal likelihood is.

We now see that we can draw a direct analogy to the Bayesian framework whereby the product of the `sample` terms is analogous to the prior, the product of the `observe` terms is analogous to the likelihood, and the partition function is analogous to the marginal likelihood. If observed variables are not sampled within or used elsewhere in the query (e.g. being used as parameters in distribution objects later used for sampling), then this analogy becomes exact as

[6]Note this is not a term that is usually used in the probabilistic programming literature, where it is usually just referred to as a marginal likelihood. Similarly, it is common in the literature to refer to a query as defining a "normalized joint" density of $p_\phi(x_{1:n_x}, y_{1:n_y}) = \prod_{j=1}^{n_x} f_{a_j}(x_j|\eta_j) \prod_{k=1}^{n_y} g_{b_k}(y_k|\psi_k)$. We have deviated from both of these because, as our examples demonstrate, this viewpoint is actually an approximation. For an even more rigorous treatment of the distributions defined by PPSs, we refer the reader to Staton et al. [2016].

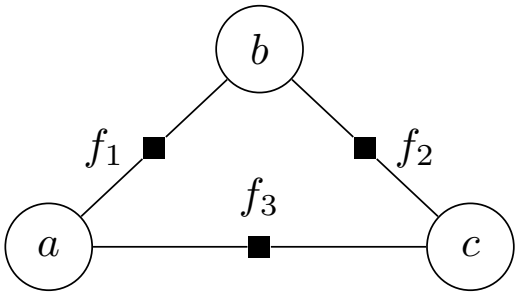
per our simplified setup.[7] However, as we have explained, it will often be desirable to go beyond this framework to specify some models. When we do, we still have an implicit Bayesian model, in the same way that Bayes' rule means that a prior and a likelihood implicitly define a posterior, but we may not actually have access to our implicitly defined prior and likelihood.

Given our query is now defined to specify an unnormalized conditional distribution rather than a normalized joint distribution, it is natural to ask whether it is necessary for each **observe** term to correspond to a correctly normalized density for its output, instead of just restricting it to be a positive semi-definite function representing an arbitrary soft constraint. The answer is that it is not. In fact, from a theoretical perspective, it perhaps easier to not think of the density defined by the query as being a combination of sampling and conditioning terms, but instead the product of a correctly normalized generative model density and a positive semi-definite *score function* that applies an unnormalized weighting to any possible sample the generative model can produce. This is exactly the approach taken by [Staton et al., 2016] and [Goodman and Stuhlmüller, 2014] who explicitly use such score functions, calling them `score` and `factor` respectively. We can also allow such functionality in our setup by defining a special distribution class, **factor**, which can only be used with **observe**, takes no parameters, and adds a factor to the trace probability equal to the exponentiation of the observed value. Thus, for example, we could use **observe** (**factor** , $\log(g(\psi))$) to factor the trace probability by the output of the function $g(\psi)$. The rationale for factoring by the exponentiation of the observed value rather than the observed value itself is firstly to ensure that the factor is positive and secondly because it will be beneficial for our system to work with log probabilities at its back-end to avoid numerical underflow issues. Similarly, we will in practice specify distribution objects using their log densities.

4.3.3 Representing Undirected Models

One important distinction of our general setup is an ability to represent undirected and cyclic models. This stems from the ability to include multiple probability factors for the same variable. For example, any arbitrary cycle in a graph can be represented by adding an observation of an already sampled variable. To instead encode an undirected model, it is perhaps easiest to think in terms of writing an importance sampler (see Section 5.3.2), an example of which is shown in Figure 4.4. Because the set of **sample** statements in a program must form a directed and acyclic model, we here use a proposal that generates all the required variables before using **observe** statements to correct the trace density to that which is desired.

[7] Note that the observed variables should not be used in deterministic elements of the code for this to hold so that they cannot implicitly affect the sampling.



(a) Factor graph

Inputs: Proposal parameters μ, Σ
1: $q \leftarrow$ **mvn** (μ, Σ)
2: $[a, b, c] \leftarrow$ **sample** (q)
3: **observe** (**factor**, $-\log(q(a, b, c))$)
4: **observe** (**factor**, $\log(f_1(a, b))$)
5: **observe** (**factor**, $\log(f_2(b, c))$)
6: **observe** (**factor**, $\log(f_3(a, c))$)
7: **return** $[a, b, c]$

(b) Query code

Figure 4.4: Example of encoding an undirected model. Here we have an undirected factor graph model (see e.g. Bishop [2006]) which we can encode by writing down an importance sampler which first samples the variables from some proposal q, before correcting the unwanted effect this has on the probability trace by adding in a factor equal to the reciprocal of the density. Provided that q is a valid proposal (see Section 5.3.2), the query defines the same distribution as the factor graph.

Although it does not theoretically restrict the models which can be encoded, this can sometimes be an somewhat cumbersome approach for encoding undirected models. For example, it forces the user to choose an appropriate proposal which somewhat goes against the philosophy of automating inference. Furthermore, though one is not restricted to using importance sampling for the inference, performance may still suffer compared to methods specifically designed for working on factor graphs. Therefore, using PPLs such as Infer.Net [Minka et al., 2010] and Factorie [McCallum et al., 2009] will often tend to be preferable for models that are well represented by conventional factor graphs as they explicitly built around providing effective support for them.

4.4 The Anglican Probabilistic Programming Language

To allow for a more precise consideration of issues associated with designing and using a universal PPS, we now introduce the particular language *Anglican* [Wood et al., 2014; Tolpin et al., 2016], which we will use for reference throughout the rest of the thesis. Anglican is a universal PPL integrated into *Clojure* [Hickey, 2008], a dynamically-typed, general-purpose, functional programming language (specifically a dialect of Lisp) that runs on the Java virtual machine and uses just-in-time compilation. Anglican inherits most of the syntax of Clojure, but extends it with the key special forms **sample** and **observe** [Tolpin et al., 2015b, 2016], defined in the same way as our example language setup in the previous section, along with a couple of others which aid in defining queries such as **mem**, **store**, and **retrieve**. Despite using predominantly the same syntax, Anglican has different *probabilistic* semantics to Clojure, i.e. code is written in the same way, but is interpreted differently.

Anglican was the first PPL to introduce particle based inference schemes such as SMC and particle MCMC (see Chapters 6 and 7), which was a key advancement for universal PPSs because it allows the structure of the query to be exploited, often providing substantially more efficient inference then previous approaches. Though these methods still form the core of the inference in Anglican, there have been a number of advancements and alternative inference approaches introduced since the original work [Paige et al., 2014; Tolpin and Wood, 2015; Tolpin et al., 2015a; van de Meent et al., 2015; Rainforth et al., 2016b,c; Le et al., 2017a]. Another notable feature of Anglican is its support for Bayesian non-parametric modeling, providing constructs for general processes along with primitives for particular models such as Dirichlet processes. Inevitably we can only provide a limited introduction here and so we also refer the interested reader to Tolpin et al. [2016] and to the Anglican website `http://www.robots.ox.ac.uk/~fwood/anglican/` for more information.

4.4.1 Clojure

Before getting into the details of Anglican, we first give a very brief introduction to Clojure [Hickey, 2008], because its syntax may be quite unfamiliar to readers without experience in Lisp-based notation or functional programming more generally. There are two key things to get your head around for reading and using Clojure (and Lisp languages more generally): almost everything is a function and parentheses evaluate functions. For example, to code $a + b$ in Clojure one would write `(+ a b)` where `+` is a function taking two arguments (here `a` and `b`) and the parentheses cause the function to evaluate. More generally, Clojure uses prefix notation such that the first term in a parenthesis is always a function, with the other terms the arguments. Thus `((+ a b))` would be invalid syntax as the result of `(+ a b)` is not a function so cannot be evaluated. Expressions are nested in a tree structure so to code $2(a + b)/c$ one would write `(/ (* 2 (+ a b)) c)`. One can thus think about going outwards in terms of execution order – a function first evaluates its arguments before itself. Functions in Clojure are first class (i.e. they can be variables) and can be declared either anonymously using `(fn [args] body)`, for example `(fn [a b] (/ a b))`, or using the macro `defn` to declare a named function in the namespace, for example `(defn divide [a b] (/ a b))`. Local bindings in Clojure are defined using `let` blocks which denote a number of name-value pairs for binding variables, followed by a series of evaluations to carry out. Their return value is the output of the last of these evaluations. Thus for example `(let [a 2 b 3] (print a) (+ a b))` says let

a be equal to 2, b be equal to 3, print a, and then evaluate $a + b$, thus returning 5. Note that a and b remain undefined outside of this let block.

Clojure allows various types of compound literals such as vectors, lists, hash maps, and sets. In general, elements in these compound literals are not restricted in their type so it valid to write for example `[1 (fn [x] (inc x)) "2"]` to construct a vector whose component terms are different types.

Clojure does not generally use for loops, instead relying upon the constructs **`map`** and **`reduce`**, or the **`loop-recur`** pairing. Here **`map`** applies a function to every value in a list or vector so for example `(map (fn [a] (* a 2)) [2.1 3])` doubles each value in the vector of inputs returning a list `(4.2 6)`, where the parentheses in the output now just represent a list rather than a function evaluation, just to be confusing. Meanwhile, **`reduce`** recursively applies a function and so for example `(reduce + 0 [1.2 3.4 2])` sums the vector of elements, returning `6.6`. The **`loop-recur`** pairing forms a syntactic sugar for doing looping through tail recursion. In essence, **`loop`** defines a function and provides initial bindings for the input arguments. Inside the **`loop`** function block, **`recur`** recursively calls the function defined by **`loop`** with new bindings to the inputs. In general, the **`recur`** function should only be called depending on a condition holding true to prevent infinite recursion. To give a concrete example,

```
(loop [x 1]
 (if (> x 10)
  nil
  (do (print x) (recur (+ x 1)))))
```

will print out 1 to 10 before returning **`nil`**. Here the syntax of **`if`** is `(if test then else)` so when `x`$>$ 10 it returns **`nil`**. Otherwise, it calls the **`do`** statement which evaluates each of its arguments in turn, in this case printing out `x` before recalling the body of the **`loop`** block with `x` reset to `(+ x 1)`.

An important feature of Clojure, particularly with regards to Anglican, is its support for (infinite) lazy sequences. Lazy sequences are sequences of terms that are only evaluated as and when they are required. As such, it is valid for them to be infinitely long (typically through recursive definition), but with only a finite number of values from which are ever evaluated in practice. For example one can define the function

```
(defn ints [n] (lazy-seq (cons n (ints (inc n)))))
```

and then call `(ints 1)` to produce a lazy sequence comprising of all the positive integers. We can evaluate the sequence by explicitly requesting terms with the sequence. For example, we can use `(take 5 (ints 1))` to return the first 5 elements of our lazy sequence, thus

returning `(1 2 3 4 5)`. We could also ask for a particular term in the sequence using, for example, `(nth 4 (ints 1))` to return 5. Lazy sequences are conceptually useful, amongst other things, for specifying things that are required to be infinitely long to ensure theoretical guarantees (e.g. an MCMC sampler needs to be run infinitely long to converge), but which are restricted by computational budgets.

4.4.2 Writing Models in Anglican

Anglican queries are written using the macro **defquery**. This allows users to define a model using a mixture of **sample** and **observe** statements and deterministic code in the manner synonymous to that explained in Section 4.3, and bind that model to a variable. As before, Anglican makes use of distribution objects for providing inputs to **sample** and **observe**, for which it provides a number of common constructors corresponding to common elementary random procedures such as **gamma**, **normal**, and **beta**. A distribution object is generated by calling the class constructor with the required parameters, e.g. `(normal 0 1)` for the unit Gaussian. Anglican also allows custom distribution classes to be defined using the **defdist** macro, which requires the user to provide parameterized code for sampling from that distribution class and evaluating the log density at given output.

A simple example of an Anglican query is shown in Figure 4.5, corresponding to a model where we are trying to infer the mean and standard deviation of a Gaussian given some data. The syntax of **defquery** is `(defquery name [args] body)` so in Figure 4.5 our query is named `my-query` and takes in arguments `data`. The query starts by sampling `mu`$\sim \mathcal{N}(0,1)$ and `sig`$\sim$ GAMMA$(2,2)$ and constructing a distribution object `lik` to use for the observations. It then maps over each datapoint and observes it under the distribution `lik`, remembering that **fn** defines a function and **map** applies a function to every element of a list or vector. Note that **observe** simply returns **nil** and so we are not interested in these outputs – they affect the probability of a program execution trace, but not the calculation of a trace itself. After the observations are made, `mu` and `sig` are returned from the **let** block and then by proxy the query itself. Because the data terms only get used as observations and we do not observe any internal random variables, this simple query corresponds exactly to a Bayesian model where the **sample** terms form the prior and the

```
(defquery my-query [data]
  (let [mu (sample (normal 0 1))
        sig (sample (gamma 2 2))
        lik (normal mu sig)]
    (map (fn [obs]
           (observe lik obs))
      data)
    [mu sig]))
```

Figure 4.5: A simple Anglican query.

```
(defquery lin-reg [xs ys]
 (let [m (sample (normal 0 10))
       c (sample (normal 0 3))
       s (sample (gamma 1 1))
       f (fn [x] (+ (* m x) c))]
  (map (fn [x y]
    (observe
      (normal (f x) s) y))
   xs ys)
  [m c s]))
```

(a) Linear regression model with unknown slope, intercept, and observation variance. Here `xs` and `ys` are the data comprising of the inputs and corresponding outputs of the regression. The query returns estimates for the slope `m`, the intercept `c`, and the observation standard deviation (i.e. noise) `s`.

```
(defquery discrete-hmm
 [ys x0 trans obs]
 (loop [xs x0 t 0]
  (let
   [x (sample (nth trans (last xs)))]
   (observe (nth obs x)
     (nth ys t))
   (if (= (inc t) (count ys))
    (conj xs x)
    (recur (conj xs x) (inc t))))))
```

(b) Hidden Markov model with discrete states. Here `ys` is the data, `x0` is a starting state (which has no associated observation), `trans` is the collection of transition distribution objects for each of the K possible states (indexed by the value of x_t), and `obs` are the matching emission distributions. Query returns estimates for the latent states `xs`.

Figure 4.6: Example Anglican queries for simple models. Here **`defquery`** binds a query to a variable, for which inference can be run using `(doquery inf-alg model inputs & options)`.

`observe` terms form the likelihood as we explained in Section 4.3.2. The query thus defines a correctly normalized joint distribution given by

$$p(\mu, \sigma, y_{1:S}) = \mathcal{N}(\mu; 0, 1)\ \text{GAMMA}(\sigma; 2, 2) \prod_{s=1}^{S} \mathcal{N}(y_s; \mu, \sigma) \tag{4.5}$$

where we have defined μ :=`mu`, σ :=`sig`, and $y_{1:S}$:=`data`. Other more complicated example Anglican models are given in Figures 4.6 and 4.7.

Compilation of an Anglican program is performed by the macro **`query`**. Calling **`query`** invokes a source-to-source compilation from an Anglican query to a continuation-passing style (CPS) Clojure function that can be executed using a particular inference algorithm. We leave discussing in depth what this means to Chapter 7, noting only that the inference algorithms associated with Anglican are written in pure Clojure (except for the odd bit of Java code) and the aim of the compilation is to produce a model artifact which can be interpreted by that inference code. When **`defquery`** is called, Anglican internally calls **`query`** and assigns the output to the symbol provided by the first argument of **`defquery`**. Therefore `my-query` in our example is Clojure function that can be read by the Anglican inference engines to produce approximate samples from the conditional distribution. Note that the call structure of this Clojure function is not as per the original Anglican query, but is instead a compilation artifact with call syntax as per what is required by the Anglican inference engine.

Inference on the model is performed using the macro **`doquery`**, which produces a lazy infinite sequence of approximate samples from the conditional distribution and, for appropri-

```
(defdist strike [volatility] []
 (sample* [this]
   (if (sample* (flip volatility))
       :war :peace))
 (observe* [this value]
  (observe* (flip volatility)
    (= value :war))))

(declare A B)

(with-primitive-procedures [strike]
 (defm A [depth]
  (let [B-strike (B (dec depth))
        A-strike (strike 0.1)]
   (observe A-strike B-strike)
   B-strike))

(defm B [depth]
 (let [B-strike (strike 0.2)]
  (if (> depth 0)
   (let [A-strike (A depth)]
     (observe B-strike A-strike)
     A-strike)
   (sample B-strike))))

(defquery war [depth]
  (or (= (A depth) :war)
      (= (B depth) :war))))
```

Figure 4.7: Anglican code for an example Schelling co-ordination game [Schelling, 1980; Stuhlmüller and Goodman, 2014] modeling if a peace will hold between two volatile countries which we refer to as A and B. This example is adapted from an online Anglican example developed by Brooks Paige and Frank Wood which can be found at `http://www.robots.ox.ac.uk/~fwood/anglican/examples/`, providing a more detailed consideration along with other examples. Our code first defines a custom distribution called `strike` for modeling whether one country would attack the other if it did not reason about the other's actions. It then constructs an Anglican function for each country using **`defm`**. Each country has their own volatility and the ability to reason about whether they think the other country will initiate a strike against them. We presume they have access to the exact model of the other country. A's aim is to match the action it thinks B will do – it would prefer there to be no war, but if there is, it wants to strike first. Its model first samples a draw for B's action by simulating from B's model and then observes this action under its own distribution on whether to strike; this is equivalent to sampling both in isolation and constraining them to be the same. However, B can also reason about A and A knows it. It, therefore, provides the simulation of B's model with a *meta-reasoning depth*. If this depth is zero, B makes its decision in isolation, simply sampling from `B-strike`. If the depth is greater than zero though, it uses the same approach as A, simulating from A's model (with a depreciated meta-reasoning depth) and then observing this outcome under `B-strike`. This creates a mutually recursive cycle of reasoning. As an observer, we are interested in establishing the probability that one of the two will strike and so we define a query which outputs whether the two are the same. The expectation of the output from this query corresponds to the probability of war. Invoking inference using `(doquery :lmh war [depth])` we see that for a depth of 0, there is roughly a 0.222 chance of war, while depths of 1, 2, and 10 give rough respective chances of 0.052, 0.0021, and 3×10^{-6}. Thus the chance of war goes down the more each country thinks about the reasoning of the other. It is interesting to note that if the volatilities are set to the same value then the critical tipping point, above which things escalate towards war with increased depth, is around 0.38. The mutual recursion involved in the model and the fact that variables are both sampled and observed means that it would be difficult to encode it using a graphical model. It is not even clear how one would go about writing down a well-defined joint distribution for the model, without having to empirically calculate a normalization constant.

ate inference algorithms, an estimate of the partition function. The syntax of **`doquery`** is `(doquery inf-alg model inputs & options)` where `inf-alg` specifies an inference algorithm, `model` is our query, `inputs` is a vector of inputs to the query, the & represents that there might be multiple additional inputs, and `options` is a series of option-value pairs for the inference engine. For example, to produce 100 samples from our `my-query` model with data

`[2.1 5.2 1.1]` using the LMH inference algorithm with no additional options, we can write `(take 100 (doquery :lmh my-query [2.1 5.2 1.1]))`.

Although Anglican mostly inherits Clojure syntax, some complications arise from the compilation to CPS-style Clojure functions. It is thus not possible to naïvely use all Clojure functions inside Anglican without providing appropriate information to the compiler to perform the required transformation. The transformation for many core Clojure functions has been coded manually, such that they form part of the Anglican syntax and can be used directly. Anglican further allows users to write functions externally to **defquery** using the macro **defm**, which is analogous to **defn** in Clojure and has the same call syntax,[8] which can then be called from inside the query without restrictions. Other deterministic Clojure functions can also be called from inside queries but must be provided with appropriate wrapping to assist the compiler. Thankfully this is done automatically (except when the function itself is higher order) using the macro **with-primitive-procedures** that takes as inputs Clojure functions and creates an environment where these functions have been converted to their appropriate CPS forms.

An important feature of Anglican for our purposes is its ability to *nest* queries within one another. This is somewhat experimental and comes with a number of health warnings because, as we will explain in Chapter 10, it allows users to write so-called *nested estimation* problems that fall beyond the scope of the standard proofs of convergence for stand Monte Carlo estimation schemes. Nonetheless, there are problems that cannot be encoded without this ability, as we will discuss in Section 10.7. Though **doquery** is not allowed directly within a **defquery** block, one can still nest queries by either using a **doquery** in a Clojure function that is then passed to another query using **with-primitive-procedures**, using a **doquery** within a custom distribution defined by **defdist**, or using the special form **conditional** which takes a query and returns a distribution object constructor corresponding to that query, for which the inputs to the query become the parameters. We will return to consider the statistical implications of this at length in Section 10.7.

4.4.3 Formal Interpretation of observe

We finish our introduction to Anglican by discussing a more formal interpretation of the **observe** special form than that taken in the existing documentation. In short, **observe** statements are formally factors applied to the unnormalized trace density (akin to the use of "score" functions in [Staton et al., 2016]), rather than likelihood terms. This is because distribution objects do not

[8] A small exception to this is that it is restricted to have a single input. As this itself can be a vector of inputs, one can still write arbitrary functions.

have explicit associated measures, with the trace density reference measure implicitly defined only through the **sample** statements. Consequently, **observe** internally evaluates the density of the observation and directly factors the trace density with the resulting numeric value, without any consideration of whether this numeric value corresponds to a probability or a probability density (i.e. whether the reference measure for the distribution object was counting or Lebesgue). The impact of this perhaps best understood through the following example

```
(defquery q1 [y]
 (let [x (sample (normal 0 1))]
  (if (< x 0)
   (observe (normal 0 1) y)
   (observe (poisson 1) y))
  x))
```

```
(defquery q2 [y]
 (let [x (sample (normal 0 1))
       z (if (< x 0)
           (sample (normal 0 1))
           (sample (poisson 1)))]
  (observe (flip 1) (= y z))
  x))
```

One would intuitively expect these queries to induce the same distribution on outputs: instead of directly observing `y`, `q2` is sampling a new variable `z` (with ostensibly the same distribution as the conditioning on `y` in `q1`) and then conditioning on the event `y=z` occurring with probability 1. We can see that if, for example, $y = 2$, then `q2` will only output positive values for `x`, as is it infinitely more probable to generate y as exactly 2 if we take the second branch than the first. However, Anglican actually interprets `q1` very differently to this: it effectively treats the **observe** in each branch as being defined with respect to the same measure, weighting the density for traces which take the first branch by $\frac{1}{\sqrt{2\pi}} \exp\left(-y^2/2\right)$ and traces which take the second branch by $e^{-1}/y!$. Consequently, `q1` will still generate negative values of `x`.

Though there is nothing mathematically wrong with this approach and it does not restrict the class of models which can be encoded, we argue that it is misleading and should be changed. The natural way to do this would be to associate reference measures with distribution objects, such that it would become possible to assert that some traces are dominated by others because of differences in the reference measures of the invoked **observe** statements, e.g. traces taking the former branch are dominated by those taking the latter in `q1`. If a trace is dominated, its density is simply set to zero. Though establishing dominated traces could potentially be achieved using static code analysis (at least for a restricted class of programs), a much simpler approach would be to establish domination at runtime by associating output samples with reference measures and pruning samples (e.g. by setting the sample weight to zero) as and when it becomes apparent that they are dominated. We leave the specific implementation of such an approach to future work.

5

An Introduction to Bayesian Inference

In Chapter 3 we introduced the concept of Bayesian modeling and showed how we can combine prior information $p(\theta)$ and a likelihood model $p(\mathcal{D}|\theta)$ using Bayes' rule (i.e. (3.6)), to produce a posterior $p(\theta|\mathcal{D})$ on variables θ that characterizes both our prior information and information from the data $\mathcal{D}$. We now consider the problem of how to calculate (or more typically approximate) this posterior, a process known as Bayesian *inference*. At first, this may seem like a straightforward problem: by Bayes' rule we have that $p(\theta|\mathcal{D}) \propto p(\mathcal{D}|\theta)p(\theta)$ and so we already know the relative probability of any one value of θ compared to another. In practice, this could hardly be further from the truth. Bayesian inference for the general class of graphical models is, in fact, an NP-hard problem [Cooper, 1990; Dagum and Luby, 1993]. In this chapter, we will outline these challenges and introduce methods for overcoming them in the form of *inference algorithms*. We will focus, in particular, on Monte Carlo inference methods, for which we will introduce some key underlying results, before introducing a number of foundational methods that form the building blocks for more advanced strategies such as those discussed in Chapter 6. We will also later use these methods to develop automated inference engines for PPS in Chapter 7.

5.1 The Challenge of Bayesian Inference

We can break down Bayesian inference into two key challenges: calculating the normalization constant $p(\mathcal{D}) = \int p(\mathcal{D}|\theta)p(\theta)d\theta$ and providing a useful characterization of the posterior, for example, an object we can draw samples from. Many inference schemes, for example, Markov chain Monte Carlo (MCMC) methods [Hastings, 1970], will not try to tackle these challenges directly and instead look to generate samples directly from the posterior. However, this breakdown will still prove useful in illustrating the intuitions about the difficulties presented by Bayesian inference.

5.1.1 The Normalization Constant

Calculating the normalization constant in Bayesian inference is essentially a problem of integration. Our target, $p(\mathcal{D})$, is the expectation of the likelihood under the prior, hence the name *marginal likelihood*. When $p(\mathcal{D})$ is known, the posterior can be evaluated exactly at any possible

input point using (3.6) directly. When it is unknown, we lack a scaling in the evaluation of any point and so we have no concept of how relatively significant that point is relative to the distribution as a whole. For example, for a discrete problem then if we know the normalization constant, we can evaluate the exact probability of any particular θ by evaluating that point alone. If we do not know the normalization constant, we do not know if there are other substantially more probable events that we have thus-far missed, which would, in turn, imply that the queried point has a negligible chance of occurring.

To give a more explicit example, consider a model where $\theta \in \{1, 2, 3\}$ with a corresponding uniform prior $P(\theta) = 1/3$ for each θ. Now presume that for some reason we are only able to evaluate the likelihood at $\theta = 1$ and $\theta = 2$, giving $p(\mathcal{D}|\theta = 1) = 1$ and $p(\mathcal{D}|\theta = 2) = 10$ respectively. Depending on the marginal likelihood $p(\mathcal{D})$, the posterior probability of $P(\theta = 2|\mathcal{D})$ will vary wildly. For example, $p(\mathcal{D}) = 4$ gives $P(\theta = 2|\mathcal{D}) = 5/6$, while $p(\mathcal{D}) = 1000$ gives $P(\theta = 2|\mathcal{D}) = 1/100$. Though this example may seem far-fetched, this lack of knowledge of the marginal likelihood is almost always seen in practice for realistic models, at least those with non-trivial solutions. Typically it is not possible to enumerate all the possible values of θ in a reasonable time and we are left wondering – how much probability mass is left that we have not seen? The problem is even worse in the setting where θ is continuous, for which it is naturally impossible to evaluate all possible values for θ. Knowing the posterior only up to a normalization constant is deceptively unhelpful – we never know how much of the probability mass we have missed and therefore whether the probability (or probability density) where we have looked so far is tiny compared to some other dominant region we are yet to explore. At its heart, the problem of Bayesian inference is a problem of where to concentrate our finite computational resources so that we can effectively characterize the posterior. If $p(\mathcal{D})$ is known, then we immediately know whether we are looking in the right place or whether there are places left to look that we are yet to find. This brings us onto our second challenge – knowing the posterior in closed form is often not enough.

5.1.2 Characterizing the Posterior

Once we have the normalization constant, it might seem that we are done; after all, we now have the exact form of the posterior using Bayes' rule. Unfortunately, it tends to be the case, at least when θ is continuous, that this is insufficient to carry out most tasks that we might want to use our posterior for. There are a number of different, often overlapping, reasons for wanting to calculate a posterior including

- To calculate the posterior probability or probability density for one or more particular instances of the variables.
- To calculate the expected value of some function, $\mu_f = \mathbb{E}_{p(\theta|\mathcal{D})}[f(\theta)]$. For example, we might want to calculate the expected values of the variables themselves $\mu_\theta = E_{p(\theta|\mathcal{D})}[\theta]$.
- To make predictions as per the posterior predictive distribution introduced in Section 3.2.
- To find the most probable variable values $\theta^* = \arg\max_\theta p(\theta|\mathcal{D})$. This is known as maximum a posteriori estimation and will be discussed in Chapters 8 and 9.
- To produce a useful representation of the posterior, e.g. a set of samples, for passing on to another part of a computational pipeline or to be directly observed by a user.
- To estimate, or sample from, a marginal distribution over some variables of particular interest. For example, if $\theta = \{u, v\}$ then we might be interested in the marginal $p(u|\mathcal{D})$.

If θ is continuous, or some elements of θ are continuous, then only the first of these can be carried out directly using the form of the posterior provided by Bayes' rule with known normalization constant. We see, therefore, that knowing the normalization alone will not be enough to fully solve the Bayesian inference problem in a useful manner. In particular, it will generally not be sufficient in order to be able to *sample* from the posterior. As we will see later, the ability to sample will be at the core of most practical uses for the posterior as it allows use of Monte Carlo methods [Metropolis and Ulam, 1949; Robert, 2004; Rubinstein and Kroese, 2016], which can, in turn, be used to carry out many of the outlined tasks.

To further demonstrate why knowing the normalization constant is insufficient for most Bayesian inference tasks, we consider the following simple example

$$p(\theta) = \textsc{Gamma}(\theta; 3, 1) = \frac{\theta^2 \exp(-\theta)}{2}, \quad \theta \in (0, \infty), \tag{5.1a}$$

$$p(y = 5|\theta) = \textsc{Student-t}(5 - \theta; 2) = \frac{\Gamma(1.5)}{\sqrt{2\pi}} \left(1 + \frac{(y-\theta)^2}{2}\right)^{-3/2}, \tag{5.1b}$$

$$p(\theta|y = 5) \approx 5.348556\, \theta^2 \exp(-\theta) \left(2 + (5-\theta)^2\right)^{-3/2}. \tag{5.1c}$$

Here we have that the prior on θ is distributed according to a gamma distribution with shape parameter 3 and rate parameter 1. The likelihood function is a student-t distribution on the difference between θ and the output $y = 5$. Using a numerical integration over θ, the normalization constant can be calculated to a high accuracy, giving the provided closed-form equation for the posterior shown in (5.1c). This posterior, along with the prior and likelihood are shown Figure 5.1.

Here knowing the marginal likelihood means that we have a closed-form equation for the posterior. Imagine though that we wish to sample from it. As it does not correspond to a standard distribution, there is, in fact, no way to directly sample from it without doing further calculations. For example, if we also know the inverse of the cumulative density function of the posterior

$$P(\Theta \leq \theta | y = 5) = \int_{\Theta=0}^{\Theta=\theta} p(\theta = \Theta | y = 5) d\Theta, \quad (5.2)$$

then we can sample from the posterior by first sampling $\hat{u} \sim \text{UNIFORM}(0, 1)$ and then taking as our exact sample $\hat{\theta} = P^{-1}(\hat{u})$ such that $\hat{u} = P(\Theta \leq \hat{\theta} | y = 5)$. However, the cumulative distribution function and its inverse cannot, in general, be calculated analytically. Though in this simple one-dimensional problem they can be easily estimated numerically, this will prove prohibitively difficult for most problems where θ has more than a few dimensions. Similarly, if we wish to estimate an expectation with respect to this posterior we could do this relatively easily numerically for this simple problem, for example using Simpson's rule, but in higher dimensions, this will be impractical.

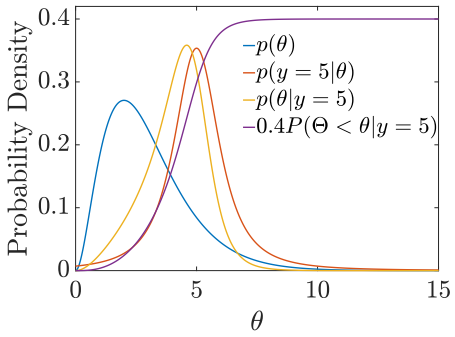


Figure 5.1: Example inference for problem given in (5.1). Also shown is the cumulative distribution for the posterior as per (5.2). This is scaled by a factor of 0.4 for visualization.

There are a number of indirect methods we could use instead to sample from the posterior such as rejection sampling, importance sampling, and MCMC. However, as we will show in Section 5.3, these all only require that we can evaluate an unnormalized version of target distribution, such that they side-step the need to calculate the marginal likelihood. Nonetheless, knowledge of the marginal likelihood can still be helpful in a number of scenarios (e.g. in adapting our inference algorithm) for the reasons outlined in Section 5.1.1.

5.2 Monte Carlo

Monte Carlo [Metropolis and Ulam, 1949] is the characterization of a probability distribution through random sampling. It is the foundation for a huge array of methods for numerical integration, optimization, and scientific simulation; forming the underlying principle for all stochastic computation. Monte Carlo provides us with a means of dealing with complex models and problems in a statistically principled manner. Without it, one would have to resort to deterministic approximation methods whenever the target problem is too complex to permit an analytic solution. As we will show, it is a highly composable framework that will allow the output of one system to be input directly to another. For example, the Monte Carlo samples

from a joint distribution will also have the correct marginal distribution over any of its individual components, while sampling from the marginal distribution then sampling from the conditional distribution given these samples, will give samples distributed according to the joint. As Monte Carlo will be key to most methods for Bayesian inference that we will consider, we take the time in this section to introduce Monte Carlo at a foundational level.

The most common usage of Monte Carlo in this work will be the Monte Carlo estimation of expectations, sometimes known as Monte Carlo integration. The critical importance of Monte Carlo estimation stems from the fact that most of the example target tasks laid out in 5.1.2 can be formulated as expectations. Even when our intention is simply to generate samples from a target distribution, we can usually think of this as being an implicit expectation of an, as yet unknown, target function. Here our implicit aim is to minimize the bias and variance of whatever process the samples are eventually used for, even if that process is simply visual inspection.

Consider the problem of calculating the expectation of some function $f(\theta)$ under the distribution $\theta \sim \pi(\theta)$ ($= p(\theta|\mathcal{D})$ for the Bayesian inference case), which we will denote as

$$I := \mathbb{E}_{\pi(\theta)}\left[f(\theta)\right] = \int f(\theta)\pi(\theta)d\theta. \tag{5.3}$$

This can be approximated using the Monte Carlo estimator I_N where

$$I \approx I_N := \frac{1}{N}\sum_{n=1}^{N} f(\hat{\theta}_n) \quad \text{and} \quad \hat{\theta}_n \sim \pi(\theta). \tag{5.4}$$

The first result we note is that (5.4) is an *unbiased* estimator for I, i.e. we have

$$\mathbb{E}\left[I_N\right] = \mathbb{E}\left[\frac{1}{N}\sum_{n=1}^{N} f(\hat{\theta}_n)\right] = \frac{1}{N}\sum_{n=1}^{N}\mathbb{E}\left[f(\hat{\theta}_n)\right] = \frac{1}{N}\sum_{n=1}^{N}\mathbb{E}\left[f(\hat{\theta}_1)\right] = I \tag{5.5}$$

where we have first moved the sum outside of expectation using linearity,[1] then the fact that each $\hat{\theta}_n$ is identically distributed to note that each $\mathbb{E}\left[f(\hat{\theta}_n)\right] = \mathbb{E}\left[f(\hat{\theta}_1)\right]$, and finally that $\mathbb{E}\left[f(\hat{\theta}_1)\right] = I$ by the definition of I and the distribution on $\hat{\theta}_1$. This is an important result as it means that Monte Carlo does not introduce any systematic error, i.e. bias, into the approximation: in expectation, it does not pathologically overestimate or underestimate the target. This is not to say though that it is equally likely to overestimate or underestimate as it may, for example, typically underestimate by a small amount and then rarely overestimate by a large amount. Instead, it means that if we were to repeat the estimation an infinite number of times and average the results, we would get the true value of I. This now hints at another important question – do we

[1] Note that this presumes that N is independent of the samples. This is usually the case, but care is necessary in some situations, namely when the number of samples taken is adaptively chosen based on the sample values, for example in adaptive stratified sampling [Etoré and Jourdain, 2010].

also recover the true value of I when we conduct one infinitely large estimation, namely if we take $N \to \infty$? This is known as *consistency* of a statistical estimator, which we will now consider next.

Before moving on, we make the important note that many common Monte Carlo inference methods, for example MCMC, are in fact biased. This is because it is often not possible to independently sample $\hat{\theta}_n \sim \pi(\theta)$ exactly as we have assumed in (5.4), with the bias resulting from the approximation. The convergence of such methods relies on the bias diminishing to 0 as $N \to \infty$, such that they remain unbiased in the limit.

5.2.1 The Law of Large Numbers

A key mathematical idea underpinning the convergence of many Monte Carlo methods is the law of large numbers (LLN). Informally, the LLN states that the empirical average of independent and identically distributed (i.i.d.) random variables converges to the true expected value of the underlying process as the number of samples in the average increases. We can, therefore, use it to prove the consistency of Monte Carlo estimators where the samples are drawn independently from the same distribution, as is the case in for example rejection sampling and importance sampling. The high-level idea for the LLN can be shown by considering the *mean squared error* of a Monte Carlo estimator as follows

$$\begin{aligned}
\mathbb{E}\left[(I_N - I)^2\right] &= \mathbb{E}\left[\left(\frac{1}{N}\sum_{n=1}^{N} f(\hat{\theta}_n) - I\right)^2\right] = \frac{1}{N^2}\mathbb{E}\left[\left(\sum_{n=1}^{N}\left(f(\hat{\theta}_n) - I\right)\right)^2\right] \\
&= \frac{1}{N^2}\sum_{n=1}^{N}\mathbb{E}\left[\left(f(\hat{\theta}_n) - I\right)^2\right] + \frac{1}{N^2}\sum_{n=1}^{N}\sum_{m=1, m\neq n}^{N}\mathbb{E}\left[(f(\hat{\theta}_n) - I)(f(\hat{\theta}_m) - I)\right] \\
&= \frac{1}{N^2}\sum_{n=1}^{N}\mathbb{E}\left[\left(f(\hat{\theta}_1) - I\right)^2\right] + \frac{1}{N^2}\sum_{n=1}^{N}\sum_{m=1, m\neq n}^{N}\underbrace{\left(\mathbb{E}\left[(f(\hat{\theta}_1) - I)\right]\right)^2}_{\to 0} \\
&= \frac{\sigma_\theta^2}{N} \quad \text{where} \quad \sigma_\theta^2 := \mathbb{E}\left[\left(f(\hat{\theta}_1) - I\right)^2\right] = \mathrm{Var}\left[f(\theta)\right]. \qquad (5.6)
\end{aligned}$$

Here the second line follows from the first simply by expanding the square and using linearity to move the sum outside of the expectation as in the unbiasedness derivation. The first term in the third line follows from the equivalent term in the second line by again noting that each $\hat{\theta}_n$ has the same distribution. The second term in the third line follows from the assumption that the samples are drawn independently such that

$$\mathbb{E}\left[(f(\hat{\theta}_n) - I)(f(\hat{\theta}_m) - I)\right] = \mathbb{E}\left[(f(\hat{\theta}_n) - I)\right]\mathbb{E}\left[(f(\hat{\theta}_m) - I)\right] = 0.$$

by unbiasedness of the estimator. The last line simply notes that $\mathbb{E}\left[\left(f(\hat{\theta}_1) - I\right)^2\right]$ is a constant, namely the variance of $f(\theta)$. Our final result has a simple and intuitive form – the mean squared

error for our estimator using N samples is $1/N$ times the mean squared error of an estimator that only uses a single sample, which is itself equal to the variance of $f(\theta)$. As $N \to \infty$, we thus have that our expected error goes to 0.

A key upshot of this result is that the difference between our empirical estimate and the true value (i.e. $I_N - I$) should be of order $O(1/\sqrt{N})$. In some way this is rather slow: deterministic numerical integration schemes often have much faster theoretical convergence rates. For example, Simpson's rule has a convergence rate of $O(1/N^4)$ for one-dimensional functions [Owen, 2013, Chapter 7]. As such, Monte Carlo is often an inferior way of estimating integrals for smooth functions in low dimensions. However, these deterministic numerical integration schemes require smoothness assumptions on f and, more critically, their convergence rates diminish rapidly (typically exponentially quickly) with the dimensionality. By comparison, the dimensionality only effects the Monte Carlo convergence rate through changes in the constant factor σ_θ and though this will typically increase with the dimensionality, this scaling will usually be substantially more graceful than deterministic numerical methods.

5.2.2 Convergence of Random Variables

To introduce the concept the LLN more precisely, we now consider some more formal notations of convergence of random variables. There will be times (e.g. Chapter 10) when mathematical rigor will require us to distinguish between alternative notations of convergence. However, those less interested in theoretical details may wish to skip Sections 5.2.2.2, 5.2.2.3, and 5.2.2.4 on first reading, as the notion of L^p convergence will be sufficient, for most practical purposes, to guarantee that an estimator will return the correct answer if provided with sufficient samples.

5.2.2.1 L^p-Convergence

We start by introducing the notion of L^p-convergence, also known as convergence in expectation, as this is the type of convergence we have just alluded to in our informal proof of the LLN. At a high level, L^p-convergence means that the expected value of the related error metric tends to zero as $N \to \infty$. More precisely, we first define the L^p-norm for a random variable X as

$$\|X\|_p = \left(\mathbb{E}\left[|X|^p\right]\right)^{\frac{1}{p}} \tag{5.7}$$

where $|\cdot|$ denotes the absolute value. For example, we can write the mean squared error used in (5.6) as the squared L^2-norm: $\mathbb{E}\left[(I_N - I)^2\right] = \|I_N - I\|_2^2$. We further define the notion of L^p-space as being the space of random variables for which $\|X\|_p < \infty$. We can now formally define L^p-convergence as follow.

Definition 5.1 (L^p-convergence)**.** *A sequence of random variables X_N converges in its L^p-norm to X (where $p \geq 1$) if $X \in L^p$, each $X_N \in L^p$, and*

$$\lim_{N\to\infty} \|X_N - X\|_p = 0. \tag{5.8}$$

A key point to note is that $\|X_N - X\|_p \geq 0 \; \forall X_N, X$ by definition of the L^p-norm and so rather than this simply being a statement of asymptotic unbiasedness, L^p-convergence says that the expected *magnitude* of the error tends to zero as $N \to \infty$. Different values of p correspond to different metrics for the error, with larger values of p constituting stronger converge guarantees, such that L^{p_2}-convergence implies L^{p_1}-convergence whenever $p_2 > p_1$. Similarly, if a random variable satisfies $X \in L^{p_2}$, then it follows that $X \in L^{p_1}$.

5.2.2.2 Convergence in Probability

At a high level, convergence in probability between two random variables (or between a random variable and a constant) means that they become arbitrarily close to one another. More formally we have the following definition.

Definition 5.2 (Convergence in probability)**.** *A sequence of random variables X_N converges in probability to X if, for every $\varepsilon > 0$,*

$$\lim_{N\to\infty} P(|X_N - X| \geq \varepsilon) = 0. \tag{5.9}$$

As ε can be made arbitrarily small, this ensures that X_N becomes arbitrarily close to X in the limit of large N. Estimators are *consistent* if they converge in probability.

In (5.6) we demonstrated the L^2 convergence of the Monte Carlo estimator as we have that

$$\lim_{N\to\infty} \|I_N - I\|_2 = \lim_{N\to\infty} \frac{\sigma_\theta}{\sqrt{N}} = 0.$$

Convergence in probability is, in general, a weaker form of convergence than L^p convergence as L^p convergence implies convergence in probability [Williams, 1991]. We therefore also have the Monte Carlo estimator convergences in probability to its expectation. This is known as the *weak law of large numbers*, which we can also prove more explicitly as follows

Theorem 5.1 (Weak law of large numbers)**.** *If I and I_N are defined as per* (5.3) *and* (5.4) *respectively, $I \in L^1$, each $I_N \in L^1$, and each $\hat{\theta}_n$ in* (5.4) *is drawn independently, then I_N converges to I in probability: $\lim_{N\to\infty} P(|I_N - I| \geq \varepsilon) = 0 \quad \forall \varepsilon > 0$.*

Proof. In the interest of exposition, we prove the result in the case where the stronger assumptions that $I \in L^2$ and each $I_N \in L^2$ hold. In practice this is not needed as the theorem can be proved by other means, see for example [Durrett, 2010, Theorem 2.2.7].

By (5.5) we have that $\mathbb{E}[I_N] = I$ and by (5.6) we have that $\|I_N - I\|_2^2 = \frac{\sigma_\theta^2}{N}$ where σ_θ^2 is an unknown, but finite, constant by the assumption that $I \in L^2$. Chebyshev's inequality states that if $\mathbb{E}[I_N] = I$, then for any $k > 0$

$$P(|I_N - I| \geq \varepsilon) \leq \frac{\mathrm{Var}(I_N)}{\varepsilon^2}.$$

By further noting that as I_N is unbiased, we have that $\mathrm{Var}(I_N) = \|I_N - I\|_2^2$ and therefore

$$\lim_{N\to\infty} P(|I_N - I| \geq \varepsilon) \leq \lim_{N\to\infty} \frac{\sigma_\theta^2}{\varepsilon^2 N} = 0 \quad \forall \varepsilon > 0.$$

□

5.2.2.3 Almost Sure Convergence

Almost sure convergence, also known as strong convergence, is a similar, but stronger, form of convergence than convergence in probability. At a high level, the difference between convergence in probability and almost sure converge is a difference in the tail behavior: convergence in probability suggests the rate at which an event happens tends to zero; almost sure convergence means that there is some point in time after which the event never happens again. More formally we have the following definition

Definition 5.3 (Almost sure convergence)**.** *A sequence of random variables X_N converges almost surely to X if*

$$P\left(\lim_{N\to\infty} X_N = X\right) = 1. \tag{5.10}$$

Almost sure convergence implies convergence in probability, but not vice-versa – the rate at which events occur might tend to zero without there ever being a point at which the event never occurs again. It does not imply, nor is it implied by, L^p convergence. The *strong law of large numbers* is the dual for the weak law of large numbers as follows.

Theorem 5.2 (Strong law of large numbers)**.** *Assuming the setup of Theorem 5.1 then I_N converges to I almost surely*

$$P\left(\lim_{N\to\infty} I_N = I\right) = 1. \tag{5.11}$$

The proof is somewhat more complicated than the weak law and so is not provided here, but can be found in, for example, [Durrett, 2010, Theorem 2.4.1].

5.2.2.4 Convergence in Distribution

At a high level, convergence in distribution, also known as weak convergence, states that a sequence of variables become increasingly closely distributed to a target distribution. Whereas our previous notions of convergence ensure that our sequence of random variables converge to a particular value, convergence in distribution only implies that our sequence of random variables tends towards having a particular distribution. For example, a variable might converge in distribution to having a unit normal distribution, whereas a different variable might converge in probability to zero. More formally, we can define convergence in distribution as follows.

Definition 5.4 (Convergence in distribution)**.** *A sequence of random variables X_N converges in distribution X if*

$$\lim_{N\to\infty} P(X_N \leq x) = P(X \leq x) \tag{5.12}$$

for every x at which $P(X \leq x)$ is continuous, where $P(X_N \leq x)$ and $P(X \leq x)$ are the cumulative distribution functions for X_N and X respectively.

Because it only requires that a variable has a particular distribution, rather than an exact value, convergence in distribution is a weaker form of convergence than those previously discussed and is implied by any of the other forms of convergence.

5.2.3 The Central Limit Theorem

The central limit theorem (CLT) is a core result in the study of Monte Carlo methods. In its simplest form, it states that the empirical mean of N i.i.d. random variables tends towards a Gaussian in the limit $N \to \infty$. While the LLN demonstrated the convergence of the average of i.i.d. random variables towards the true mean, the CLT provides a means of constructing consistent confidence intervals of our estimate I_N by exploiting its asymptotic normality. Furthermore, it has variants that do not require that the variables are i.i.d., meaning we can use it do demonstrate convergence in scenarios where samples are correlated, such as occurs when doing MCMC inference as shown in Section 5.3.4. In the i.i.d. case, the CLT is as follows.

Theorem 5.3 (Central Limit Theorem)**.** *Assume $X_1, \ldots, X_N$ is a sequence of i.i.d. random variables with $\mathbb{E}[X_i] = I$ and $\mathbb{E}[X_i^2] = \sigma < \infty$ for all $i \in \{1, \ldots, N\}$. Let $I_N := \frac{1}{N}\sum_{n=1}^{N} X_n$ be the sample average of this sequence. Then $\sqrt{N}\,(I_N - I)\,/\sigma$ converges in distribution to the unit normal:*

$$\lim_{N\to\infty} P\left(\frac{\sqrt{N}\,(I_N - I)}{\sigma} \leq z\right) = \Phi(z) = \int_{-\infty}^{z} \frac{1}{2\pi} \exp\left(-\frac{\zeta^2}{2}\right) d\zeta \quad \forall z \in \mathbb{R} \tag{5.13}$$

where $\Phi(z)$ is the cumulative distribution function for the unit normal. Furthermore, if $\sigma_N^2 = \frac{1}{N-1}\sum_{n=1}^{N}(X_N - I_N)^2$ is the empirical estimate of the variance of X_N (including Bessel's correction), then $\sqrt{N}(I_N - I)/\sigma_N$ also converges in distribution to the unit normal.

Proof. See, for example, [Durrett, 2010, Chapter 3]. □

The second result here is especially key as it will allow us to calculate consistent confidence intervals on our estimates without needing to know σ as we can instead use our empirical estimate

$$\sigma_N^2 = \frac{1}{N-1}\sum_{n=1}^{N}(X_N - I_N)^2 \quad \text{where} \quad I_N = \frac{1}{N}\sum_{n=1}^{N} X_N. \tag{5.14}$$

The form of (5.14) might at first look at bit strange - why is the normalizing term $1/(N-1)$ instead of $1/N$? Although replacing the $1/(N-1)$ term with $1/N$ would still lead to a consistent estimator, this estimator turns out to be biased whereas (5.14) is in fact unbiased. The bias correction $N/(N-1)$ is known as Bessel's correction, while the proof that (5.14) is unbiased follows from straightforward algebraic manipulations. Note though that the estimator of the standard deviation, σ_N, is still biased by Jensen's inequality. As a consequence of the unbiasedness, $\sigma_N^2 \overset{a.s.}{\to} \sigma^2$ by the strong LLN, from which the second result in Theorem 5.3 follows from the first by Slutsky's theorem.

Imagine we want to construct a confidence interval on our estimate I_N such that there is a probability $0 < \alpha < 1$ that I is in the range $[I_N - \varepsilon, I_N + \varepsilon]$. Using the CLT we can do this as follows

$$\begin{aligned}
\alpha :&= P(I_N - \varepsilon \leq I \leq I_N + \varepsilon) = P(-\varepsilon \leq I_N - I \leq \varepsilon) \\
&= P\left(-\frac{\sqrt{N}\varepsilon}{\sigma_N} \leq \frac{\sqrt{N}(I_N - I)}{\sigma_N} \leq \frac{\sqrt{N}\varepsilon}{\sigma_N}\right) \\
&\approx \Phi\left(\frac{\sqrt{N}\varepsilon}{\sigma_N}\right) - \Phi\left(-\frac{\sqrt{N}\varepsilon}{\sigma_N}\right) = 2\Phi\left(\frac{\sqrt{N}\varepsilon}{\sigma_N}\right) - 1.
\end{aligned} \tag{5.15}$$

Rearranging for ϵ we have that $\varepsilon = \left(\sigma_N/\sqrt{N}\right)\Phi^{-1}\left(\frac{\alpha+1}{2}\right)$ and therefore the confidence interval

$$P\left(I_N - \frac{\sigma_N \Phi^{-1}\left(\frac{\alpha+1}{2}\right)}{\sqrt{N}} \leq I \leq I_N + \frac{\sigma_N \Phi^{-1}\left(\frac{\alpha+1}{2}\right)}{\sqrt{N}}\right) = \alpha. \tag{5.16}$$

Thus for example, if we set $\alpha = 0.99$ then $\Phi^{-1}\left(\frac{\alpha+1}{2}\right) \approx 2.576$ and our confidence interval 99% confidence interval for I is $I_N \pm \frac{2.576\sigma_N}{\sqrt{N}}$. We reiterate that these confidence intervals are exact in the limit $N \to \infty$, but approximate for finite N. As one is often far from the asymptotic regime, care is often required in interpreting them. Nonetheless, the ability to estimate realistic uncertainties for sufficiently large N is an extremely powerful result.

Through most of this section, we have assumed that our random variables are i.i.d.. In fact, neither the assumption of being identically distributed nor that of independence is actually necessary for the CLT to hold. One can instead use the concept of *strong mixing*, namely that variables sufficiently far apart in the sequence are independent, to generalize beyond the i.i.d. setting [Jones et al., 2004]. This is critical for the numerous Monte Carlo inference schemes, such as MCMC methods, that do not produce independent samples but instead rely on the samples converging in distribution to a target distribution. In most MCMC settings, one can also prove a CLT using reversibility of the Markov chain [Kipnis and Varadhan, 1986]. It is beyond the scope of this thesis to go into these more general forms of the CLT in depth, so we simply note their critical importance and refer the reader to Durrett [2010] for a comprehensive introduction.

5.3 Foundational Monte Carlo Inference Methods

In this section, we introduce the key methods that form the basis upon which most Monte Carlo inference schemes are based. The key idea at the heart of all Monte Carlo inference methods is to use some form of proposal distribution that we can easily sample from and then make appropriate adjustments to achieve (typically approximate) samples from the posterior. Most methods will require only an unnormalized distribution

$$\gamma(\theta) = \pi(\theta)Z \tag{5.17}$$

as a target where $Z = \int \gamma(\theta)d\theta$. As such they will apply to any situation where we desire to sample from an unnormalized (or in some cases normalized) distribution, for which the Bayesian inference setting is a particular case where $\gamma(\theta) = p(\mathcal{D}|\theta)p(\theta)$ and $Z = p(\mathcal{D})$. Nonetheless, knowing Z can be useful in a number of scenarios, e.g. allowing for unbiased importance sampling estimators. The methods will, in general, vary only on how samples are proposed and the subsequent adjustments that are made. However, this will lead to a plethora of different approaches, varying substantially in their motivation, theoretical justification, algorithmic details, and the scenarios for which they are effective.

5.3.1 Rejection Sampling

Rejection sampling is one of the simplest Monte Carlo inference methods and one of the only ones to produce exact samples from the target. Before going into the method itself, we first consider an example to demonstrate the underlying intuition. Imagine we want to generate samples distributed uniformly over some arbitrary two-dimensional shape. One simple way of doing this would be to sample uniformly from a box enclosing the shape and then only taking the samples which fall

within the shape. An example of such sampling by rejection is shown in Figure 5.2. As all the samples within the box are distributed uniformly, they are also uniformly distributed on any subset of the space. Therefore if we sample from the box and then only take the samples that fall within the desired shape, we will generate samples uniformly over that shape. We can also use this method to estimate the area of the shape by using the fact that the probability of any one sample falling within the shape is equal to the ratio of the areas of the shape and the bounding box, namely

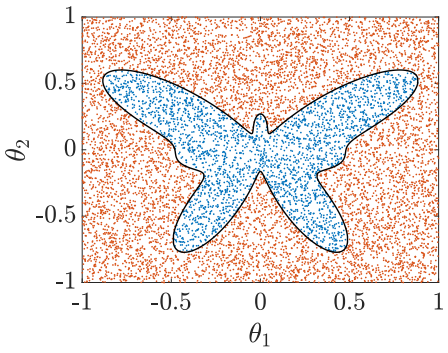


Figure 5.2: Sampling uniformly from an arbitrary shape by rejection. Samples are proposed uniformly from the $[-1, 1]$ square. Any sample falling within the black outline is accepted (blue), otherwise it is rejected (red).

$$\begin{aligned} A_{\text{shape}} &= A_{\text{box}} P(\theta \in \text{shape}) \\ &\approx \frac{A_{\text{box}}}{N} \sum_{n=1}^{N} \mathbb{I}(\hat{\theta}_n \in \text{shape}) \quad \text{where} \quad \hat{\theta}_n \sim \textsc{Uniform}(\text{box}) \end{aligned}$$

and we have used a Monte Carlo estimator for $P(\theta \in \text{shape})$. Note that the value of $P(\theta \in \text{shape})$ will dictate the efficiency of our estimation as it represents the *acceptance rate* of our samples. In other words, we need to generate on average $1/P(\theta \in \text{shape})$ samples from our proposal for each sample created in the target area. As we will show later in Section 5.3.3, $P(\theta \in \text{shape})$ typically becomes very small as θ becomes high-dimensional, so this approach will typically only be effective in low dimensions.

The underlying idea to extend this approach to rejection sampling more generally, is that we can sample from any distribution by sampling uniformly from the hyper-volume under its unnormalized probability density function. Though this is effectively axiomatic by the definition of a probability density function with respect to the Lebesgue measure, we can get a non measure-theoretic intuition for this by considering augmenting a target distribution with a new variable u such that $p(u|\theta) = \textsc{Uniform}(0, \gamma(\theta))$. Sampling $\hat{\theta} \sim \pi(\theta)$ and then $\hat{u} \sim p(u|\theta)$ corresponds to sampling uniformly from hyper-volume under the probability density function, while we clearly have that the marginal distribution on θ is $\pi(\theta)$.

Using this idea, we can sample from any unnormalized distribution by sampling from an appropriate bounding as per Figure 5.2 and then accepting only samples that fall within the hyper-volume of the probability density function. More specifically, we define a proposal distribution $q(\theta)$ which completely envelopes a scaled version of the unnormalized target distribution $C\gamma(\theta)$,

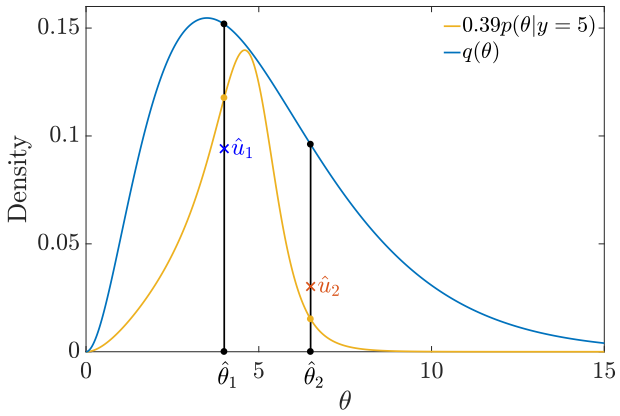


Figure 5.3: Demonstration of rejection sampling for problem shown in (5.1). We first sample $\hat{\theta} \sim q(\theta)$, correspond to sampling from the distribution shown in blue, and then sample $\hat{u} \sim \text{UNIFORM}(0, q(\theta))$, corresponding to sampling a point uniformly along the black lines for the two shown example values of $\hat{\theta}$. The point is accepted if $\hat{u} \leq Cp(\theta|y=5)$ (i.e. if it below the yellow curve), where we have taken $C = 0.39$ to ensure $Cp(\theta|y=5) \leq q(\theta)$ for all theta. Here the example sample pair $\{\hat{\theta}_1, \hat{u}_1\}$ is accepted, while $\{\hat{\theta}_2, \hat{u}_2\}$ is rejected. The resulting accepted sample pairs will be uniformly sampled from the region under the unnormalized target distribution given by the yellow curve and therefore the accepted $\hat{\theta}$ will correspond to exact samples from the target.

for some fixed C, such that $q(\theta) \geq C\gamma(\theta)$ for all values of θ. We then sample a pair $\{\hat{\theta}, \hat{u}\}$ by first sampling $\hat{\theta} \sim q(\theta)$ and then $\hat{u} \sim \text{UNIFORM}(0, q(\theta))$. The sample is accepted if

$$\hat{u} \leq C\gamma(\hat{\theta}) \tag{5.18}$$

which occurs with an acceptance rate CZ (note that $q(\theta) \geq C\gamma(\theta)\ \forall\theta$ ensures that $C \leq 1/Z$). This can be used to estimate the normalization constant Z, corresponding to the marginal likelihood for Bayesian models, by calculating the empirical estimate of the acceptance rate and dividing this by C. A graphical demonstration of the rejection sampling process is shown in Figure 5.3.

Rejection sampling can be a highly effective sampling or inference method in low dimensions. In particular, the fact that it generates exact samples from the target distribution can be very useful. This very rare characteristic is used to construct efficient samplers for many common distributions such as in the ziggurat algorithm [Marsaglia et al., 2000] often used for generating Gaussian random variables. More generally, whenever one wishes to construct a sampler for a non-standard low dimensional distribution, e.g. when declaring an Anglican `defdist`, rejection sampling is the clear go-to approach because it produces exact samples and because one can usually engineer a very efficient sampler. However, the efficiency of rejection sampling is critically dependent on the value of C, because C is directly proportional to the acceptance rate. By proxy, it is also critically dependent on the proposal $q(\theta)$ as this dictates the minimum possible value of C, namely $C_{\text{min}} = \min_\theta q(\theta)Z/\pi(\theta)$. Consequently, it is very prone to

the *curse of dimensionality* as we discuss in Section 5.3.3, meaning performance cannot be maintained for higher dimensional problems.

5.3.2 Importance Sampling

Importance sampling is another common sampling method that forms the key building block for many more advanced inference schemes. It is closely related to rejection sampling in that it uses a proposal, i.e. $\hat{\theta} \sim q(\theta)$, but instead of going through an accept/reject step, it assigns an *importance weight* to each sample. These importance weights act like correction factors to account for the fact that we sampled from $q(\theta)$ rather than our target $\pi(\theta)$.

To demonstrate the key idea, consider the problem of calculating an expectation as per (5.3). If we cannot sample exactly from $\pi(\theta)$ then we cannot apply (5.4) directly. However, we can rearrange the form of our expectation to generate a different Monte Carlo estimator which we can evaluate directly as follows

$$\begin{aligned} I := \mathbb{E}_{\pi(\theta)}\left[f(\theta)\right] &= \int f(\theta)\pi(\theta)d\theta = \int f(\theta)\frac{\pi(\theta)}{q(\theta)}q(\theta)d\theta \\ &\approx \frac{1}{N}\sum_{n=1}^{N}\frac{\pi(\hat{\theta}_n)}{q(\hat{\theta}_n)}f(\hat{\theta}_n) \quad \text{where} \quad \hat{\theta}_n \sim q(\theta) \end{aligned} \tag{5.19}$$

where $\frac{\pi(\hat{\theta}_n)}{q(\hat{\theta}_n)} =: w_n$ is known as an importance weight. The key trick we have applied is to multiply the integrand by $\frac{q(\theta)}{q(\theta)}$, which equals 1 for all points where $q(\theta) \neq 0$. Thus if $q(\theta) \neq 0$ for all θ for which $\pi(\theta) \neq 0$ (to avoid infinite importance weights), this has no effect on the expectation. However, we can informally view the new formulation as being the expectation of $f(\theta)\frac{\pi(\theta)}{q(\theta)}$ under the distribution $q(\theta)$. We can now construct a Monte Carlo estimator for this new formulation, by choosing $q(\theta)$ to be a distribution we sample from. A graphical demonstration of importance sampling is given in Figure 5.4 in the more general setting where we do not have access to the $\pi(\theta)$ exactly, but only an unnormalized version $\gamma(\theta) = \pi(\theta)Z$. As we will show in detail in Section 5.3.2.1, we can still use importance sampling in this case by *self-normalizing* the weights.

Importance sampling has a number of desirable properties as an inference method. In particular, it is both unbiased and consistent. The former can be shown trivially as follows

$$\begin{aligned} \mathbb{E}\left[\frac{1}{N}\sum_{n=1}^{N}\frac{\pi(\hat{\theta}_n)}{q(\hat{\theta}_n)}f(\hat{\theta}_n)\right] &= \frac{1}{N}\sum_{n=1}^{N}\mathbb{E}_{q(\theta_n)}\left[\frac{\pi(\hat{\theta}_n)}{q(\hat{\theta}_n)}f(\hat{\theta}_n)\right] \\ &= \mathbb{E}_{q(\theta_1)}\left[\frac{\pi(\hat{\theta}_1)}{q(\hat{\theta}_1)}f(\hat{\theta}_n)\right] = \mathbb{E}_{\pi(\theta)}\left[f(\theta)\right] \end{aligned} \tag{5.20}$$

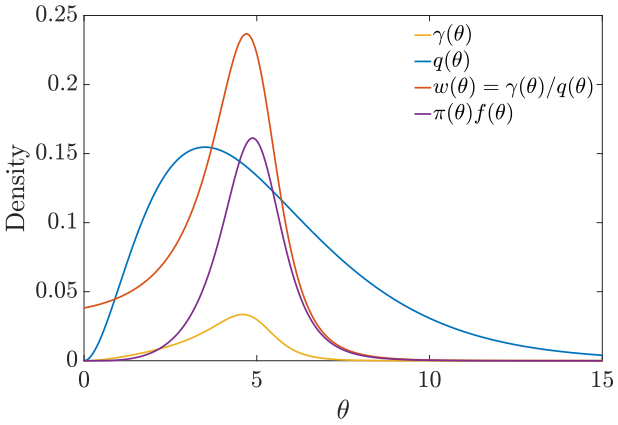


Figure 5.4: Demonstration of importance sampling for problem shown in (5.1). We are trying to estimate $\mathbb{E}_{\pi(\theta)}[f(\theta)]$: the expectation of the function $f(\theta) := \theta^2/50$ under the posterior $\pi(\theta) := p(\theta|y = 5)$ defined as per (5.1c). We assume the setting where $\pi(\theta)$ is only known up to a normalization constant (see Section 5.3.2.1), namely we only have access to $\gamma(\theta) := p(\theta)p(y = 5|\theta)$ as shown in yellow. Our procedure is to draw samples independently $\hat{\theta}_n \sim q(\theta)$ and then evaluate their weight $w_n = w(\hat{\theta}_n) = \gamma(\hat{\theta}_n)/q(\hat{\theta}_n)$. This produces a set of weighted samples which can then be used to estimate to estimate the expectation using (5.26) (one can also use (5.19) if the normalized $\pi(\theta)$ is used instead of $\gamma(\theta)$).

where we have effectively stepped backward through (5.19).[2] Given this unbiasedness result, L^2 convergence, and thus convergence in probability, can be shown in the same manner as (5.6) by replacing each $f(\hat{\theta}_n)$ with $\frac{\pi(\hat{\theta}_n)}{q(\hat{\theta}_n)} f(\hat{\theta}_n)$, leading to the same result except that σ_θ is now

$$\sigma_\theta^2 = \mathbb{E}_{q(\theta)}\left[\left(\frac{\pi(\hat{\theta}_1)}{q(\hat{\theta}_1)} f(\hat{\theta}_1) - I\right)^2\right] = \mathrm{Var}_{q(\theta)}\left[\frac{\pi(\theta)}{q(\theta)} f(\theta)\right]. \tag{5.21}$$

Almost sure convergence of importance sampling can be similarly shown using the strong LLN.

The form of (5.21) provides substantial insight into how best to set the proposal: we have shown that the mean squared error of an importance sampler is directly proportional to $\mathrm{Var}_{q(\theta)}[f(\theta)\pi(\theta)/q(\theta)]$, thus the lower this term is, the better the expected performance of our estimator. One obvious question is what is the optimal proposal $q^*(\theta)$? It turns out that $q^*(\theta) = \frac{\pi(\theta)|f(\theta)|}{\int \pi(\theta)|f(\theta)|d\theta}$ [Kahn and Marshall, 1953; Owen, 2013], which can be shown as follows where we will make use of Jensen's inequality and comparing to an arbitrary proposal $q(\theta)$

$$\begin{aligned}
\mathrm{Var}_{q^*(\theta)}\left[\frac{\pi(\theta)}{q^*(\theta)} f(\theta)\right] &= \mathbb{E}_{q^*(\theta)}\left[\left(\frac{\pi(\theta)}{q^*(\theta)} f(\theta)\right)^2\right] - \left(\mathbb{E}_{q^*(\theta)}\left[\frac{\pi(\theta)}{q^*(\theta)} f(\theta)\right]\right)^2 \\
&= \int \frac{\pi(\theta)^2 f(\theta)^2}{q^*(\theta)} d\theta - I^2 = \left(\int \pi(\theta)\,|f(\theta)|\,d\theta\right)^2 - I^2 \qquad (5.22) \\
&\leq \int \left(\frac{\pi(\theta) f(\theta)}{q(\theta)}\right)^2 q(\theta) d\theta - I^2 = \mathrm{Var}_{q(\theta)}\left[\frac{\pi(\theta)}{q(\theta)} f(\theta)\right]. \qquad (5.23)
\end{aligned}$$

[2] Note that this unbiasedness result does not pass over to the self-normalized variant given in Section 5.3.2.1.

Here we have shown that the variance for $q^*(\theta)$ is less than or equal to the variance using an arbitrary $q(\theta)$. It must, therefore, be the optimal proposal. A further point of note, is that if $f(\theta) \geq 0 \ \forall\theta$ (or $f(\theta) \leq 0 \ \forall\theta$), then (5.22) will equal zero giving a zero variance estimator: each importance weight will be equal to the $I/f(\theta)$ and thus I can be calculated by evaluating a single point.

Though it will typically be impossible to find $q^*(\theta)$ in practice, it still provides a guide as to what constitutes a good proposal – we want $\pi(\theta)\left|f(\theta)\right|/q(\theta)$ to be as close to constant as possible. In particular, we need to be careful to avoid scenarios where $\frac{\pi(\theta)|f(\theta)|}{\int \pi(\theta)|f(\theta)|d\theta} \gg q(\theta)$ as this will cause the ratio to explode, leading to high variances. A consequence of this is that we want $q(\theta)$ to have *light tails* compared to $\pi(\theta)\left|f(\theta)\right|$ to ensure that the ratio does not systematically increase as θ moves away from the modes of $q(\theta)$. Aside from the clear practical issues, if this requirement does not hold, then it can easily be the case that $\sigma_\theta = \infty$ and thus that the estimator has infinite variance. Consider, for example, the case where $\pi(\theta) = \mathcal{N}(\theta; 0, 1)$, $f(\theta) = \theta$, and $q(\theta) = \mathcal{N}(\theta; 0, s^2)$ (Example 9.1 from [Owen, 2013, example 9.1]). Noting that the mean, I, is zero by symmetry and defining $\nu = \frac{1}{2s^2} - 1$, we have that

$$\sigma_\theta^2 = \int_{-\infty}^{\infty} \theta^2 \frac{\left(\exp\left(-\theta^2/2\right)/\sqrt{2\pi}\right)^2}{\exp\left(-\theta^2/\left(2s^2\right)\right)/\sqrt{2\pi s^2}} d\theta - I^2 = \frac{s}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \theta^2 \exp\left(\theta^2\nu\right) d\theta.$$

Now this integral is clearly only finite for $\nu < 0$ (as otherwise the integrand is $+\infty$ and $\theta = \pm\infty$ and finite elsewhere). Therefore, σ_θ is only finite when $s^2 > 1/2$. In other words, we only get a finite estimator in this case if the proposal variance is at least half that of the target distribution $\pi(\theta)$! This highlights the pitfalls of having insufficiently heavy tails on our proposal. Overcoming these will typically require careful setup of the proposal on a case-by-case basis, for example, choosing a distribution type for the proposal that is known to have heavier tails than $\pi(\theta)$.

5.3.2.1 Self-Normalized Importance Sampling

In the previous section, we presumed that we have access to a normalized version of the target $\pi(\theta)$. Typically this will not be the case and we will only have access to an unnormalized target $\gamma(\theta) = \pi(\theta)Z$ as per (5.17), for example only having access to the joint rather than the posterior in the Bayesian inference setting. In a less common, but still plausible situation, it may also only be possible to evaluate the proposal up to a normalization constant. We now show how one can still use importance sampling in these scenarios, by *self-normalizing* the importance weights.

The key idea for self-normalized importance sampling (SNIS) is that the weights provide an unbiased and consistent estimator of the marginal likelihood

$$Z_N = \frac{1}{N}\sum_{n=1}^{N} w_n, \tag{5.24}$$

$$\mathbb{E}[Z_N] = \frac{1}{N}\sum_{n=1}^{N} \mathbb{E}[w_n] = \mathbb{E}_{q(\hat{\theta}_1)}[\gamma(\hat{\theta}_1)/q(\hat{\theta}_1)] = Z. \tag{5.25}$$

Now as $\mathbb{E}_{q(\theta)}\left[\frac{\gamma(\theta)}{q(\theta)}f(\theta)\right] = E_{q(\theta)}\left[\frac{\pi(\theta)}{q(\theta)}Zf(\theta)\right] = Z\ \mathbb{E}_{\pi(\theta)}\left[f(\theta)\right]$, we can use our samples to construct Monte Carlo estimators for both Z and $Z\ \mathbb{E}_{\pi(\theta)}\left[f(\theta)\right]$ and use the ratio of our estimates to get an estimate for $E_{\pi(\theta)}\left[f(\theta)\right]$

$$\mathbb{E}_{\pi(\theta)}\left[f(\theta)\right] \approx \frac{\sum_{n=1}^{N} w_n f(\hat{\theta}_n)}{\sum_{n=1}^{N} w_n} \quad \text{where} \quad \hat{\theta}_n \sim q(\theta), \quad w_n = \frac{\gamma(\hat{\theta}_n)}{q(\hat{\theta}_n)}. \tag{5.26}$$

This can alternatively be expressed as $\mathbb{E}_{\pi(\theta)}\left[f(\theta)\right] \approx \sum_{n=1}^{N} \bar{w}_n f(\hat{\theta}_n)$ where $\bar{w}_n = \frac{w_n}{\sum_n w_n}$ are the normalized importance weights such that $\sum_{n=1}^{N} \bar{w}_n = 1$.

The consistency of (5.26) follows by Slutsky's from the individual consistency of both the numerator and the denominator to $Z\ \mathbb{E}_{\pi(\theta)}\left[f(\theta)\right]$ and Z respectively. However, unlike (5.19), (5.26) is a *biased* estimator for finite N. This is because the numerator and denominator are correlated and because even though Z_N is an unbiased estimator of Z, $1/Z_N$ is not an unbiased estimator of $1/Z$. The latter follows directly from Jensen's inequality noting that inversion is a convex function for strictly positive inputs,

$$\mathbb{E}\left[\frac{1}{\frac{1}{N}\sum_{n=1}^{N} w_n}\right] \geq \frac{1}{\mathbb{E}\left[\frac{1}{N}\sum_{n=1}^{N} w_n\right]} = \frac{1}{Z}, \tag{5.27}$$

where equality holds if and only if Z_N is a zero variance estimator for Z (which typically happens only in the limit $N \to \infty$). However, it can be shown that the bias decreases at a rate $O(1/N)$ (see e.g. Doucet and Johansen [2009]), whereas the standard deviation of the estimate decreases at a rate $O(1/\sqrt{N})$. Thus the bias becomes dominated as $N \to \infty$.

Strangely, even when the normalization constant is known, the self-normalized importance sampler can still be lower variance [Owen, 2013]. Therefore, with the bias becoming dominated, it can actually be preferable to use SNIS even when the normalized target is known. Note also that the optimal proposal in the SNIS case varies slightly from the $q^*(\theta)$ derived earlier in the Section and is instead [Hesterberg, 1988]

$$q^*_{\text{SNIS}}(\theta) = \frac{\pi(\theta)\left|f(\theta) - I\right|}{\int \pi(\theta)\left|f(\theta) - I\right| d\theta}. \tag{5.28}$$

As a consequence, there is a minimum variance for the SNIS estimator, unlike in the pre-normalized case where $q^*(\theta)$ was a zero variance estimator if $f(\theta) \geq 0\ \forall\theta$.

5.3.2.2 Unknown f

So far we have assumed that we are using importance sampling to calculate an expectation of a known function. In practice, there will be many scenarios, particularly in the Bayesian inference setting, where $f(\theta)$ is not known ahead of time and we instead desire to generate samples for some future unknown use. For example, in Section 6.1 we will introduce the concept of *sequential Monte Carlo*, where we will typically have multiple importance sampling steps before any target function is itself evaluated. When no $f(\theta)$ is specified, we can carry out importance sampling in the same fashion, sampling from $q(\theta)$ and returning a set of *weighted* samples $\{\hat{\theta}_n, w_n\}_{n=1:N}$ where the weights are equal to $\gamma(\hat{\theta}_n)/q(\hat{\theta}_n)$ as before. Here we can think of importance sampling as approximating the posterior with a series of deltas functions, namely

$$\pi(\theta) \approx \hat{\pi}(\theta) := \sum_{n=1}^{N} \bar{w}_n \delta_{\hat{\theta}_n}(\theta) \tag{5.29}$$

where $\delta_{\hat{\theta}_n}(\theta)$ are delta functions centered at $\hat{\theta}_n$.

Importance weights are multiplicative when doing conditional sampling: if we sample $\hat{\theta}_n \sim q_1(\theta)$ then $\hat{\phi}_n|\hat{\theta}_n \sim q_2(\phi|\hat{\theta}_n)$ when targeting $\gamma_1(\theta)\gamma_2(\phi|\theta)$ then the importance weight is

$$\frac{\gamma_1(\hat{\theta}_n)\gamma_2(\hat{\phi}_n|\hat{\theta}_n)}{q_1(\hat{\theta}_n)q_2(\hat{\phi}_n|\hat{\theta}_n)} = \frac{\gamma_1(\hat{\theta}_n)}{q_1(\hat{\theta}_n)} \times \frac{\gamma_2(\hat{\phi}_n|\hat{\theta}_n)}{q_2(\hat{\phi}_n|\hat{\theta}_n)} = w_{n,1} \times w_{n,2}. \tag{5.30}$$

This is known as sequential importance sampling and means that we can propagate importance weighted samples through a computational system and retain a valid importance sampler with the standard properties such as unbiasedness (presuming the weights are not self-normalized) and consistency.

Again a natural question in this "unknown f" setting is what is the optimal proposal $q^*(\theta)$? This is a somewhat more challenging and subjective question than when f is known as the optimality will depend on what the samples are eventually used for. In particular, even if we do not know f precisely, it may be the case that we believe some f are more likely than others, or we may know that f lives within some space of possible functions, for example, functions that have a countable number of discontinuities.

One simple, but insightful approach, is to consider the *minimax* optimal proposal, i.e. the proposal that has the minimum error if f is the most adversarial possible function for that proposal. Imagine that we have N weighted samples $\{\hat{\theta}_n, w_n\}_{n=1:N}$, with N corresponding evaluations $f_n := f(\hat{\theta}_n)$. We use these to calculate an estimate I_N for the target I. Now assume that each $w_n \geq 0$, $\sum_{n=1}^{N} w_n = 1$, and $\sum_{n=1}^{N} |f_n - I| = N$ to preclude the ability to provide an improved

solution simply by scaling the problem. The error for the estimator is given by

$$|I_N - I| = \left| \sum_{n=1}^{N} w_n f_n - I \right| = \left| \sum_{n=1}^{N} w_n (f_n - I) \right|. \tag{5.31}$$

For any given set of weights, the most adversarial set of function evaluations is to have all our error at the largest weight, i.e. $|f_{n^*} - I| = N$ and $|f_n - I| = 0,\ n \neq n^*$ where n^* is the index of the largest weight. For this set of adversarial f_n, the lowest error is achieved when each of the weights are equal. Consequently, we see that the minimax proposal under our assumptions is $q(\theta) = \pi(\theta)$. This result is perhaps not surprising as it effectively states that the best sample representation of $\pi(\theta)$ for an unknown future use is to sample from that distribution directly. As shown in the next section, this proposal will also minimize the variance of the estimator under the assumption that the f_n are independent.

Nonetheless, this point of view is not without criticism in the literature [O'Hagan, 1987; Ghahramani and Rasmussen, 2003; Briol et al., 2015b]. Most of this criticism revolves around the valid point that in practice the value of $f(\theta)$ does convey information about $f(\theta + \varepsilon)$ and so the correlation between samples should be taken into account when making an estimation. Though the thrust of this argument is mostly directed towards changing the Monte Carlo method, and in particular importance sampling, the same arguments would also suggest that it can be beneficial to use a proposal that is more diffuse than the target, thereby reducing the correlation between the produced samples.

5.3.2.3 Effective Sample Size

In this section, we consider an important diagnostic for the performance of importance sampling based schemes, the *effective sample size* (ESS). In Section 5.2.3 we showed how an uncertainty estimate for a Monte Carlo estimator can be derived. However, this cannot be used when f is unknown and even if f is known, it does necessarily convey much information about how effective our proposal is compared to how effective it *could* be. The ESS instead informally provides an estimated measure of the amount of information stored in our weighted sample set. The more information stored in the samples, the better our approximation of the posterior, and, at a high level, the more evenly balanced our weights, the more information they encode. Therefore, the ESS is a measure of how many unweighted samples would be required to convey the same information about the posterior as the weighted sample set.

The weighted average of N_e independent evaluations $\{f_n\}_{n=1}^{N}$, each with individual variance σ^2, has variance σ^2/N_e as we showed in (5.6). Therefore, we can calculate the ESS of a set of weighted samples by comparing the variance of our weighted estimate to the variance of an

estimate using a set of unweighted evaluations. More specifically, the ESS will be the number of unweighted evaluations N_e that gives an equivalent variance to our weighted estimate as follows where we will make use of the assumption that the f_n are independent

$$\frac{\sigma^2}{N_e} = \mathrm{Var}\left[\frac{\sum_{n=1}^N w_n f_n}{\sum_{n=1}^N w_n}\middle|\{w_n\}_{n=1}^N\right] = \sum_{n=1}^N \left(\frac{w_n}{\sum_{n=1}^N w_n}\right)^2 \mathrm{Var}[f_n] = \frac{\sigma^2 \sum_{n=1}^N w_n^2}{\left(\sum_{n=1}^N w_n\right)^2}. \tag{5.32}$$

Now rearranging for N_e we get

$$N_e = \frac{\left(\sum_{n=1}^N w_n\right)^2}{\sum_{n=1}^N w_n^2} = \frac{1}{\sum_{n=1}^N \bar{w}_n^2} \tag{5.33}$$

which completes our definition for the effective sample size.[3] It transpires that N_e is independent of f, so we can still use the ESS as a diagnostic when f is unknown. It is straightforward to show using Jensen's inequality we have that $N_e \leq N$ with equality holding if and only if all the weights are equal. On the other hand, if all but one of the weights is zero, then $N_e = 1$. These two extremes respectively occur when the proposal is equal to the target, $q(\theta) = \pi(\theta)$, and when the proposal provides a very poor representation of the target. The ESS is often therefore used for *proposal adaptation*, as a larger value of the ESS generally indicates a better proposal.

However, the ESS is far from a perfect measure of sample quality. For example, if the proposal perfectly matches one of the modes of the target but completely misses another larger mode, the ESS will usually be very high, even though the samples provide a very poor representation of the target. It is not uncommon in practice to see the ESS drop drastically as more samples are added, due to the addition of a new dominating sample, typically indicating a region of significant target probability mass that had previously been missed. Nonetheless, the ESS is still a very useful performance metric and is usually a reliable indicator for whether our importance sampling is struggling. In particular, though the possibility of missing modes means that it is possible for the ESS to be high even when the approximation of the posterior is poor, if the ESS is low then the approximation of the posterior will always be poor and any subsequent estimates will usually be high variance.

5.3.2.4 Resampling

A useful feature of SNIS is that it can be used to produce unweighted samples by *sampling with replacement* from the set of produced samples in proportion to the sample weights. This procedure is typically known as resampling, because we are resampling samples from the empirical distribution of our original samples. Resampling allows us to generate unweighted

[3] Note that it is sometimes necessary to collapse identical weighted samples to a single sample when calculating (5.33), such as when doing resampling (see Section 5.3.2.4)

samples with importance sampling which are approximately distributed according to $\pi(\theta)$, with this approximation becoming exact in the limit $N \to \infty$. Resampling on its own always leads to a higher variance estimator than using (5.26) directly. However, in addition to being necessary when unweighted samples are required, it will be a key component in the so-called particle-based inference methods discussed in Chapter 6.

Mathematically, we can express resampling as producing a set unweighted resampled samples $\left\{\tilde{\theta}_n\right\}_{n=1}^N$ using

$$\tilde{\theta}_n = \hat{\theta}_{a_n} \quad \text{where} \quad a_n \sim \textsc{Discrete}\left(\{\bar{w}_n\}_{n=1}^N\right). \tag{5.34}$$

Here $\{a_n\}_{n=1}^N$ are known as ancestor indices as they indicate which ancestor in the original sample set each unweighted sample originated from. Note that the a_n need not be drawn independently and typically are not; (5.34) instead conveys the required marginal distribution for each a_n.

Considering the approximation of the posterior provided by importance sampling given in (5.29), we can view resampling as producing the approximation

$$\tilde{\pi}(\theta) := \sum_{n=1}^N \frac{k_n}{N} \delta_{\hat{\theta}_n}(\theta) \tag{5.35}$$

where k_n is the number times the sample $\hat{\theta}_n$ appears in the resampled sample set $\left\{\tilde{\theta}_n\right\}_{n=1}^N$. Provided that $\mathbb{E}[k_n | \{w_n\}_{n=1}^N] = N k_n$, then it directly follows that $\tilde{\pi}(\theta)$ is an unbiased estimator for $\hat{\pi}(\theta)$. Consequently, the L^2 convergence, and thus consistency, of SNIS with resampling follows directly from the L^2 convergence of SNIS.

There are a number of different methods one can use for resampling [Douc and Cappé, 2005]. They all share in common the requirements above but vary in correlations between the a_n. The simplest method, *multinomial resampling*, simply involves sampling each a_n independently, such that the k_n have a multinomial distribution. Though simple, this method is generally inadvisable as it adds unnecessary variation to the resampling compared to methods using randomized quasi-Monte Carlo [L'Ecuyer and Lemieux, 2005], such as systematic resampling [Carpenter et al., 1999; Whitley, 1994],[4] or other variance reduction techniques, such as stratified resampling [Kitagawa, 1996] and residual resampling [Whitley, 1994]. Though residual resampling is a little more complicated (see Douc and Cappé [2005]), stratified and systematic resampling can be viewed as small changes on the underlying random number draws made in multinomial resampling. The typical way to draw from a multinomial distribution with N trials

[4] Note that systematic resampling, as it is now known, is somewhat confusingly referred to as stratified resampling in the former of these papers and universal sampling in the latter.

Algorithm 5.1 Resampling

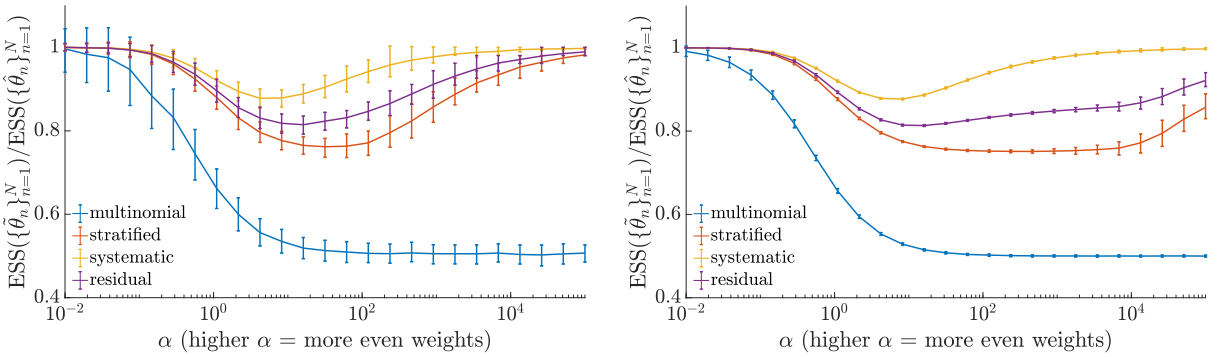
Inputs: weighted samples $\{w_n, \hat{\theta}_n\}_{n=1}^N$, method $\mathcal{M}$
Outputs: unweighted samples $\{\tilde{\theta}_n\}_{n=1}^N$

switch $\mathcal{M}$ **do**
  case Multinomial
    $u_n \sim \text{UNIFORM}(0,1), \quad \forall n \in 1,\ldots,N$
  case Stratified
    $u_n \sim \text{UNIFORM}\left(\frac{n-1}{N}, \frac{n}{N}\right), \quad \forall n \in 1,\ldots,N$
  case Systematic
    $u_1 \sim \text{UNIFORM}\left(0, \frac{1}{N}\right)$
    $u_n \leftarrow u_1 + \frac{n-1}{N}, \quad \forall n \in 2,\ldots,N$
end switch
Normalize weights $\bar{w}_n \leftarrow w_n / \left(\sum_\ell^N w_\ell\right), \quad \forall n \in 1,\ldots,N$
$P_n \leftarrow \sum_{\ell=1}^n \bar{w}_\ell, \quad \forall n \in 1,\ldots,N$ ▷ $P_0 = 0$
$a_n = \{\ell \in \{1,\ldots,N\} \colon P_{\ell-1} \leq u_n < P_\ell\}, \quad \forall n \in \{1,\ldots,N\}$
$\tilde{\theta}_n = \hat{\theta}_{a_n}, \quad \forall n \in \{1,\ldots,N\}$
return $\left\{\tilde{\theta}_n\right\}_{n=1}^N$

and probabilities $p_1,\ldots,p_K$ is to make N independent draws from a unit uniform distribution, $u_n \overset{i.i.d.}{\sim} \text{UNIFORM}(0,1)\ \forall n \in 1,\ldots,N$ and then to bin these into the intervals between the cumulative probabilities $P_0 = 0,\ P_j = \sum_{\ell=1}^j p_\ell,\ \forall j \in 1,\ldots,K$. Thus the result of each trial a_n corresponds to which bin the respective u_n falls into:

$$a_n = \{j \in \{1,\ldots,K\} \colon P_{j-1} \leq u_n < P_j\}, \quad \forall n \in \{1,\ldots,N\} \tag{5.36}$$

and counts for each event, k_j, is the number of u_n satisfying $P_{j-1} \leq u_n < P_j$. In the context of multinomial resampling, $K = N$ and a_n and k_j are as per (5.34) and (5.35) respectively, noting that the corresponding requirements for their distributions are trivially satisfied. Stratified resampling and systematic resampling differ only in how the u_n are drawn. For stratified resampling, each u_n is independently sampled from a different strata of the full $[0,1]$ space such that $u_n \sim \text{UNIFORM}(\frac{n-1}{N}, \frac{n}{N})$, which enforces that $u_1 \leq u_2 \leq \cdots \leq u_N$. For systematic resampling, the only draw made is $u_1 \sim \text{UNIFORM}(0, \frac{1}{N})$, with all other u_n set deterministically from this point using $u_n = u_1 + \frac{n-1}{N},\ \forall n \in 2,\ldots,N$. By also randomly permuting $\{u_n\}_{n=1}^N$ before applying (5.36), each u_n after permutation is still uniformly distributed on $[0,1]$ for both methods and so (5.34) is still satisfied, as is the requirement that $\mathbb{E}[k_n|\{w_n\}_{n=1}^N] = Nk_n$ [Douc and Cappé, 2005]. In practice, the ordering of the samples is usually arbitrary such that this



(a) Sample set size $N = 100$ **(b)** Sample set size $N = 10000$

Figure 5.5: Demonstration of performance of different resampling schemes for two different sample set sizes and different levels of variance in the weights. Plots are generated by first sampling a set of N weights from a symmetric Dirichlet distribution with parameter α, i.e. $\{\bar{w}_n\}_{n=1}^N \sim \text{DIRICHLET}(\alpha)$, such that the higher α is, the more even the weights are on average. The effective sample size (ESS) as defined in (5.33) is then calculated before and after the resampling is carried out (with identical samples collapsed to a single sample), and the ratio reported. Plots show the mean across 1000 independent trials, with the error bars corresponding to the interquartile range.

permutation is not necessary. A summary of these three approaches is given in Algorithm 5.1.

Systematic, stratified, and residual resampling are lower variance than multinomial sampling, with systematic resampling the most widely used due to its simplicity of implementation and generally being the best performing [Doucet and Johansen, 2009]. It should be noted, however, that there can be theoretical complications with systematic resampling, while stratified and residual resampling can be shown to dominate multinomial resampling [Douc and Cappé, 2005]. To give insight into how these methods reduce the variance of eventual estimation, consider a case where all the weights are equal. Because the ancestor variables are drawn independently for multinomial resampling, many of the of the original samples will not be present in the resampled set. More precisely, the probability of any particular sample being present is given by

$$P(\hat{\theta}_n \in \{\tilde{\theta}\}_{n=1}^N) = 1 - \left(\frac{N-1}{N}\right)^N \overset{N\to\infty}{\longrightarrow} 1 - \frac{1}{e} \approx 0.6321.$$

Thus substantial information is lost in the resampling. On the other hand, if we use any of our variance reduction techniques, each of the original samples will appear exactly once in the resampled set and so no information is lost in this scenario. In the other extreme, when all the weight is on a single sample, all the resampling schemes will behave the same (returning only the sample with non-zero weight). In most practical scenarios we will be somewhere between these two extremes, but the high-level intuition, that multinomial resampling throws away more information than the other approaches, will remain the same.

An empirical assessment of the methods is shown in Figure 5.5. This shows the relative effective sample size (ESS) before and after resampling as a function of the uniformity of the weights. To calculate the ESS after the resampling, we collapse the identical duplicated particles down to a single sample with the combined weight of the other samples. This is equivalent to replacing w_n with k_n/N in the ESS calculation given in (5.33). Figure 5.5 shows that systematic resampling is clearly preferable to the alternatives, at least in the terms of the ESS after resampling for this particular problem, with this improvement consistent across the different sample size sets. Similar relative performance improvements are found if one instead uses the proportion of samples that survive the resampling as a performance metric. As expected, multinomial resampling is significantly worse than the other methods, particularly when the weights are more even. The results also suggest that residual resampling outperforms stratified resampling, though this can come at the cost of being more complicated to implement and potentially slower to run.

5.3.3 The Curse of Dimensionality

In this section, we digress from introducing specific inference methods themselves to talk about a common problem faced by most inference methods, the *curse of dimensionality* [Bellman, 1961]. At a high-level, the curse of dimensionality is a tendency of modeling and numerical procedures to get substantially harder as the dimensionality increases, often at an exponential rate. If not managed properly, it can cripple the performance of inference methods and it is the main reason the two procedures discussed so far, rejection sampling and importance sampling, are in practice only used for very low dimensional problems. At its core, it stems from an increase of the size (in an informal sense) of a problem as the dimensionality increases. This is easiest to see for discrete problems. Imagine we are calculating an expectation over a discrete distribution of dimension D, where each dimension has K possible values. The cost of enumerating all the possible combinations scales as K^D and thus increases exponentially with D; even for modest values for K and D this will be prohibitively large.

However, the curse of dimensionality extends far beyond problems of enumeration. It will be felt, to some degree or another, by almost all approaches for inference and modeling more generally, but its effect will be most pronounced for methods that try to explicitly model the target space. As a geometrical demonstration of this, consider the effect of dimensionality on the sampling by rejection approach introduced at the start of Section 5.3.1. For simplicity, we will presume that the target shape is a hypersphere and that the bounding shape is the smallest hypercube that encloses that hypersphere. Our acceptance rate, and thus the efficiency of the

algorithm, will be equal to the ratio of the two hyper-volumes. For an even number of dimensions, the hyper-volume of the D-dimensional hypersphere with radius r is $V_s = \frac{\pi^{D/2} r^D}{(D/2)!}$ and the hyper-volume of the enclosing hypercube is $V_c = (2r)^D$, giving a ratio of $\frac{V_s}{V_c} = \frac{\pi^{D/2}}{(D/2)! 2^D} = \left(\frac{\sqrt{\pi}}{2}\right)^D \frac{1}{(D/2)!}$. The first of these terms decreases exponentially and second super-exponentially with D (noting that $(D/2)! > (D/6)^{(D/2)}$). For example, $D = 10, 20$, and 100 respectively gives ratios of approximately 2.5×10^{-3}, 2.5×10^{-8}, and 1.9×10^{-70}. Consequently, our acceptance rate will diminish super-exponentially with the number of dimensions and our approach will quickly become infeasible in higher dimensions.

An immediate possible criticism of this analysis would be to suggest that the approximation of the target shape provided by our bounding shape is simply increasingly poor as the dimensionality increases and that we should choose a better approximation. Although this is true, the key realization is that achieving a good approximation is increasingly difficult in high dimensions, typically exponentially so. To demonstrate this, imagine we instead use an arbitrary bounding shape defined in polar co-ordinates, such that the proportional difference in the radius at the any given point is at most ε (i.e. the radius of our bounding shape is between r and $r(1+\varepsilon)$ at all points). The hyper-volume in which our approximation might live for an even number of dimensions is given by

$$V_\varepsilon = \frac{\pi^{D/2} r^D (1+\varepsilon)^D}{(D/2)!} - \frac{\pi^{D/2} r^D}{(D/2)!} = V_s \left((1+\varepsilon)^D - 1\right). \tag{5.37}$$

Consequently, we have that the ratio $\frac{V_\varepsilon}{V_s}$ increases exponentially with D and that for sufficiently large D and a fixed ε the amount of space in our tolerance region will become substantially larger than the target hyper-volume, again leading to very low acceptance rates.

Flipping this on its head, we can ask the question how does ε need to vary to ensure that V_ε / V_s remains constant? A quick manipulation shows that $\varepsilon = \left(\left(\frac{V_\varepsilon}{V_s} + 1\right)^{1/D} - 1\right)$ and therefore that $\frac{\log\left(\frac{V_\varepsilon}{V_s}+1\right)}{D} \geq \log(1+\varepsilon) \approx \varepsilon$ for small values of ε. Thus we only need to decrease ε roughly in proportion to $\frac{1}{D}$ to achieve a fixed ratio. Initially, this would not seem so bad. For example, if $D = 1000$ we roughly need $\varepsilon \leq 6.9 \times 10^{-4}$ to get $V_\varepsilon / V_s \leq 1$. However, this misses the key difficulty caused by (5.37): the higher D is, the more difficult it is to accurately model the target shape and keep ε small. It follows from (5.37) that as D increases, the more the hyper-volume of the sphere is concentrated at the surface. This generalizes to non-spherical targets and means that accurate modeling the surface of our target is essential in high dimensions. Unfortunately, this task becomes rapidly more difficult with increasing dimensionality.

Consider, for example, regressing the surface of the target by using a number of inducing points spread over the surface. As the dimensionality increases, these become increasingly far apart from one another and so the more points we need to accurately model the surface. For example, the probability of two points uniformly distributed on the surface of a sphere being within some target distance of one another decreases exponentially with the dimensionality of the sphere. This can be seen by noting that a necessary condition for two points to be within d of each other, is that the discrepancy of each individual dimension must be less than d. In other words, if we denote the overall distance as δ and the distance in each dimension as δ_i then we have

$$P(\delta \leq d) \leq P(\delta_1 \leq d)P(\delta_2 \leq d) \dots P(\delta_D \leq d)$$

and so $P(\delta \leq d)$ must decrease exponentially with D. If the correlation between points is proportional to their euclidean distance, then we will subsequently need an exponentially large number of points in the dimension to model the surface to a given accuracy. Consequently, we see that not only are we increasingly punished for any discrepancies between our approximation and the target as the dimensionality increases, it rapidly becomes harder to avoid these discrepancies in the first place.

One can informally think of the proposals we have introduced thus-far as being approximations to the target distribution: complications with tail behavior aside, it will generally be the case that the better the proposal approximates the target, the better the inference will perform. This typically leads to catastrophically bad performance for importance sampling and rejection sampling in high dimensions, for which this approximation breaks down for the reasons we have just outlined. To give a simple example, imagine that our target is an isotropic unit Gaussian and we use an independent student-t distribution with $\nu = 2$ in each dimension as the proposal. We have that the weights are as follows

$$w(\theta) = \frac{\pi(\theta)}{q(\theta)} = \prod_{i=1}^{D} \frac{\exp(-\theta_d^2/2)/\sqrt{2\pi}}{\frac{\Gamma(1.5)}{\sqrt{2\pi}}\left(1+\theta_d^2/2\right)^{-3/2}} = \prod_{i=1}^{D} \frac{2\exp(-\theta_d^2/2)\left(1+\theta_d^2/2\right)^{3/2}}{\sqrt{\pi}}. \tag{5.38}$$

It follows that the variance of the weights under the proposal increases exponentially with D as

$$\begin{aligned}\mathrm{Var}_{q(\theta)}\left[w(\theta)\right] &= \int w^2(\theta)q(\theta)d\theta - \left(\int w(\theta)q(\theta)d\theta\right)^2 = -1 + \prod_{i=1}^{D}\int_{-\infty}^{\infty} w^2(\theta_d)q(\theta_d)d\theta_d \\ &= -1 + \prod_{i=1}^{D}\int_{-\infty}^{\infty} \frac{\sqrt{2}\exp(-\theta_d^2)\left(1+\theta_d^2/2\right)^{(3/2)}}{\pi}d\theta_d \approx 1.1455^D - 1 \end{aligned} \tag{5.39}$$

where we have used the fact that the integral has a closed form solution. We thus see that our effective sample size will drop exponentially quickly with D and that our inference will break down if the dimensionality is too high.

It is now natural to ask whether we can overcome the curse of dimensionality. Thankfully, the answer in many scenarios is that we can. In many high-dimensional scenarios then our target distribution will typically only have significant mass in a small proportion of the total area, often concentrated around a lower dimensional manifold of the larger space. This means that if we use inference methods that in some way exploit the structure of the target distribution and only search the small subset of the space with significant mass, then effective inference can still be performed. When this is not the case, practical inference will typically be futile in high dimensions anyway and so many inference algorithms are geared towards exploiting a particular type of structure. As we will show in the next section, the effectiveness of MCMC methods is mostly based on exploiting single modality in the target by making local moves that cause the algorithm to have a hill-climbing style behaviour away from the mode and then sticking close to the mode once it is found. Sequential Monte Carlo methods on the other hand rely on using the structure of the target more explicitly, by using a series of stepping-stone distributions and adaptively allocating resources (see Chapter 6). Variational and message passing methods make assumptions or approximations about the structure of the model to break the inference problem down into a number of small problems that can then be combined into an overall estimate. Arguably the key to all advanced inference methods is how well they can exploit structure in higher dimensions, while the relative performance of difference methods tends to come down to how suited the target is to their particular form of structure exploitation.

5.3.4 Markov Chain Monte Carlo

Markov chain Monte Carlo (MCMC) methods [Metropolis et al., 1953; Hastings, 1970; Gilks et al., 1995] form one of the key approaches to circumventing the curse of dimensionality and are perhaps the most widely used class of algorithms for Bayesian inference, though they are also used extensively outside the Bayesian inference setting. The key idea is to construct a valid Markov chain that has the target distribution as its equilibrium distribution. They are suitable for Bayesian inference because this can still be done when the target distribution is only known up to a normalization constant.

The reason that they are often able to overcome, or at least alleviate, the curse of dimensionality, is that rather than trying to independently sample from the target distribution at each iteration, they instead make *local* moves from their current position. As with rejection sampling and importance sampling, they use a proposal distribution, but unlike these alternatives, the proposal is defined conditionally on the current location, namely, they propose according to

$\theta' \sim q(\theta'|\theta)$ where θ is the current state and θ' is the new sampled state. The underlying intuition behind this is that in high dimensions the proportion of the space with significant probability mass is typically very small. Therefore, if the target is single modal (or we have a proposal that is carefully designed to jump between modes), then once we have a sample in the mode, all the other points with significant mass should be close to that point. Therefore we can explore the mode by restricting ourselves to local moves, overcoming the curse of dimensionality by predominantly ignoring the majority of the space which has insignificant probability mass. As the dimensionality increases, the proportion of the space with significant mass decreases, counteracting many of the other complications that arise from the increasing dimension. When away from a mode, MCMC methods often behave like hill-climbing algorithms, emphasizing their close links with simulated annealing methods for optimization [Aarts and Korst, 1988]. Therefore, they can be highly effective for both finding the mode of a posterior and then sticking to that mode.

5.3.4.1 Markov Chains

We first introduced the concept of the *Markov property* in Section 3.3 in the concept of a hidden Markov model, where we explained how in a Markovian system each state is independent of all the previous states given the last state, i.e.

$$p(\Theta_n = \theta_n|\Theta_1 = \theta_1, \ldots, \Theta_{n-1} = \theta_{n-1}) = p(\Theta_n = \theta_n|\Theta_{n-1} = \theta_{n-1}). \tag{5.40}$$

In other words, the system transitions based only on its current state. Here the series $\Theta_1, \ldots, \Theta_n, \ldots$ is known as a Markov chain. We see that a probability of a Markov chain is fully defined by the probability of its initial state $p(\Theta_1 = \theta_1)$ and the probability of its transitions $p(\Theta_n = \theta_n|\Theta_{n-1} = \theta_{n-1})$. If each transition has the same distribution, i.e.

$$p(\Theta_{n+1} = \theta'|\Theta_n = \theta) = p(\Theta_n = \theta'|\Theta_{n-1} = \theta), \tag{5.41}$$

then the Markov chain is known as homogeneous. Most MCMC methods are based on homogeneous Markov chains (the exception being adaptive MCMC methods, see Section 5.3.5) and so we will assume that (5.41) holds from now on. In such situations, $p(\Theta_{n+1} = \theta_{n+1}|\Theta_n = \theta_n)$ is typically known as a *transition kernel* $T(\theta_{n+1} \leftarrow \theta_n)$.

For a Markov chain to converge to a target distribution $\pi(\theta)$, we will need that $\lim_{n\to\infty} p(\Theta_n = \theta) = \pi(\theta)$ for any possible starting position θ_1, i.e. that the chain converges in distribution to the target for all possible starting positions. For this to happen we need two things: $\pi(\theta)$ must be a *stationary distribution* of the Markov chain, such that if $p(\Theta_n = \theta) = \pi(\theta)$ then

$p(\Theta_{n+1} = \theta) = \pi(\theta)$, and all possible starting points θ_1 must converge to this distribution. The former of these will be satisfied if

$$\pi(\theta') = \int T(\theta' \leftarrow \theta)\pi(\theta)d\theta \tag{5.42}$$

where we see that the target distribution is *invariant* under the application of the transition kernel. Thus if $p(\theta_n) = \pi(\theta)$ for some n, all subsequent points will have the desired distribution. The requirement that all starting points converge to the desired target distribution is known as *ergodicity*, which guarantees both the uniqueness of the stationary distribution and that all points converge to this distribution. Ergodicity requires that the Markov chain is *irreducible*, i.e. all points with non-zero probability can be reached in a finite number of steps, and *aperiodic*, i.e. that no states can only be reached at certain periods of time. We will not delve into the specifics of ergodicity in depth, but note only that homogeneous Markov chains that satisfy (5.42) can be shown to be ergodic under very weak conditions, see for example Neal [1993]; Tierney [1994].

5.3.4.2 Detailed Balance

A common sufficient (but not necessary) condition used for constructing valid Markov chains is to ensure that the chain satisfies the condition of *detailed balance*. Chains that satisfy detailed balance are known as *reversible*.[5] For a target $\pi(\theta)$, detailed balanced is defined as

$$\pi(\theta)T(\theta' \leftarrow \theta) = \pi(\theta')T(\theta \leftarrow \theta'). \tag{5.43}$$

It is straightforward to see that Markov chains satisfying detailed balance will admit $\pi(\theta)$ as a stationary distribution by noting that

$$\int T(\theta' \leftarrow \theta)\pi(\theta)d\theta = \int T(\theta \leftarrow \theta')\pi(\theta')d\theta = \pi(\theta'). \tag{5.44}$$

Thus any ergodic Markov chain we construct that satisfies (5.43) will converge to the target distribution. From an inference perspective, this means that we can eventually generate samples according to our desired target by choosing an arbitrary start point Θ_1 and then repeatedly sampling from our transition kernel $T(\Theta_n \leftarrow \Theta_{n-1})$.

[5] An interesting area of current research is the study of non-reversible MCMC algorithms [Bouchard-Côté et al., 2015; Bierkens et al., 2016]. These can be beneficial because traditional reversible processes are only able to move by drifting over time – generally at least half of the proposed samples (and generally much more) will be in a direction at odds to this drift. Forcing multiple consecutive steps in a particular direction can thus help the chain to mix faster.

5.3.4.3 Metropolis Hastings

One of the simplest and most widely used MCMC methods is Metropolis Hastings (MH) [Hastings, 1970]. Given an unnormalized target $\gamma(\theta)$, then at each iteration of the MH algorithm, one samples a new point θ' according to the a proposal $\theta' \sim q(\theta'|\theta_n)$ conditioned on the current point θ_n and then accepts the new sample with probability

$$P(\text{Accept}) = \min\left(1, \frac{\gamma(\theta')q(\theta_n|\theta')}{\gamma(\theta_n)q(\theta'|\theta_n)}\right). \tag{5.45}$$

At iteration n then we set $\theta_{n+1} \leftarrow \theta'$ if the sample is accepted and otherwise set $\theta_{n+1} \leftarrow \theta_n$. Critically this process does not require access to the normalized target $\pi(\theta)$. It is trivial to show that (5.45) satisfies detailed balance and therefore produces a valid Markov chain as follows

$$\begin{aligned}
\pi(\theta_n)T(\theta_{n+1} \leftarrow \theta_n) &= \min\left(1, \frac{\gamma(\theta_{n+1})q(\theta_n|\theta_{n+1})}{\gamma(\theta_n)q(\theta_{n+1}|\theta_n)}\right)\pi(\theta_n)q(\theta_{n+1}|\theta_n) \\
&= \min\left(\gamma(\theta_n)q(\theta_{n+1}|\theta_n), \gamma(\theta_{n+1})q(\theta_n|\theta_{n+1})\right)/Z \\
&= \min\left(\frac{\gamma(\theta_n)q(\theta_{n+1}|\theta_n)}{\gamma(\theta_{n+1})q(\theta_n|\theta_{n+1})}, 1\right)\pi(\theta_{n+1})q(\theta_n|\theta_{n+1}) \\
&= \pi(\theta_{n+1})T(\theta_n \leftarrow \theta_{n+1}).
\end{aligned}$$

Though MH is valid for any reasonable choice of the proposal distribution [Tierney, 1994], the practical performance will depend heavily on this choice. For example, if $q(\theta'|\theta)$ is independent of θ then no information is passed from one iteration to the next and one gets an algorithm that is strictly worse than importance sampling because samples are independently generated in the same way, but information is lost in the accept-reject step. Instead, one will generally want to propose points close to the current point so the advantages of *local moves* can be exploited, namely through the hill climbing behavior we previously discussed. However, this has complications as explained in the next section, while choosing a proposal with the right characteristics is still rather challenging. For example, image we use an isotropic Gaussian proposal. If the variance of our proposal is too high then we will rarely propose good points and so the acceptance rate will become very low, giving few distinct samples. If the variance is too low, the Markov chain will move very slowly as it can only take small steps. This will increase correlation between all our samples and reduce the fidelity of our estimates. Chains that quickly cover the full probability space are said to *mix* quickly.

5.3.4.4 Intuitions, Complications, and Practical Considerations

Though MCMC methods can be exceptionally effective, they are not without their weaknesses. Most of these weakness stem from the fact that all the generated samples are correlated, leading to,

for example, biased estimates. Correlation reduces the amount of distinct information conveyed by each sample and this will reduce the accuracy of the estimator. However, it also causes more fundamental issues. Most of the convergence results we have presented so far have relied on samples being generated in a i.i.d. fashion, which is clearly not the case in the MCMC setting. MCMC methods therefore require their own unique convergence proofs, based in general on ergodic theory (see e.g. [Durrett, 2010, Chapter 6]). Furthermore, whereas importance sampling and rejection sampling lead to unbiased estimates of the marginal likelihood, MCMC produces no natural estimate and methods that do produce marginal likelihood estimates for MCMC are often extremely biased [Chib and Jeliazkov, 2001].

The aforementioned convergence results mean that the bias of estimates made using MCMC samples tends to zero as the number of iterations tends to infinity, but it is often very difficult to estimate the magnitude of the bias for a finite numbers of iterations. Whereas importance sampling and rejection sampling had reasonable diagnostics for the performance of the inference, such as the effective sample size and the acceptance rate respectively, estimating the bias from MCMC samplers is typically fiendishly difficult and it can often look like an MCMC sampler is performing well (e.g. in terms of its acceptance rate) when in fact it is doing disastrously. One of the most common ways this is manifested is in the sampler becoming stuck in a particular mode of the target. Using localized proposals can make it prohibitively difficult to move between modes. Though valid MCMC samplers must eventually visit every mode infinitely often, it can take arbitrarily long to even visit each mode once. Even worse, getting the correct estimate relies on spending the correct relative proportion of time in each mode, which will typically take many orders of magnitude more time to get a reasonable estimate for, than it will just to have the sampler visit each significant mode at least once. The issues associated with multiple modes provides a demonstration of why it is difficult to estimate the bias of an MCMC sampler: we do not generally know if we have missed another mode or whether our sampler has spent an appropriate amount of time in each mode.

Because of these drawbacks, using MCMC on multi-modal problems is dangerous unless an appropriate mechanism for transitioning between the modes can be found. One also tends to throw away some of the earlier samples in the Markov chain to allow the chain to *burn in*, remembering that samples are only distributed according to the target of interest asymptotically and so the earlier samples, which have marginal distributions very far away from the distribution of interest, can add substantial bias to the resultant estimator. Thankfully, there are a surprisingly wide array of models that actually fit these restrictions, particularly in high dimensions or if

we can find an appropriate parameterization of the model. Remembering from Section 2.6 that changing the parameterization of a model changes its probability density function in a non-trivial manner, the performance of MCMC methods is often critically dependent on their parameterization. Changing the parameterization will change the concept of what parameter values are close to which other parameter values. In an ideal world, we would make moves in the raw sample space where all points are equally probable. Typically this is not practical, but it is still usually the case that some parameterizations will tend to be more single-modal and more generally have all points of interest close together in the parameter space. Note that there is often an equivalence here between a good proposal and a good warping of the space to one where an isotropic proposal will be effective. One possible way of achieving a good parameterization is through the use of *auxiliary variables* [Higdon, 1998; Andrieu et al., 2010], which can improve mixing by allowing more degrees of freedom in the proposal, decreasing the chance of getting "stuck", e.g. in a particular mode. Somewhat counterintuitively, projecting to higher dimensional spaces can actually substantially improve the mixing of an MCMC sampler.

As a concrete example of this, Hamiltonian Monte Carlo (HMC) [Duane et al., 1987; Neal, 2011] uses derivatives of the density function and an auxiliary variable to make effective long distance proposals. Given that much of the behavior of MCMC is based on hill-climbing effects, it would perhaps be intuitive to presume that these gradients are used hasten the hill-climbing behavior or try to move between modes. In practice, the intention is exactly the opposite. In high dimensions, most of the mass of a mode is not at its peak but in a thin strip around that peak known as a *typical set* [Betancourt, 2017]. Classical random walk MH methods will both rarely propose samples in the right direction to stay in this level set, giving a low acceptance rate, and be very slow to move around the level set because the reversibility of the proposals mean that this only happens slowly through drift. By moving perpendicular to the gradient, HMC makes proposals that are more likely to stay within the typical set, while also allowing large moves to be made in a single step. Together these mean that it can explore a particular mode much faster than random walk MH strategies. See Betancourt [2017] for an excellent introduction.

5.3.4.5 Gibbs Sampling

Gibbs sampling is an important special case of Metropolis-Hastings that looks to update only some subset of variables in a joint distribution at each iteration. Imagine we have a D distributional target distribution $\pi(\theta)$ where $\theta = \{\theta_1, \theta_2, \ldots, \theta_D\}$. Gibbs sampling incrementally updates one or more of the variables θ_d at each iteration conditioned on the value of the others. Thus it uses

proposals of the form $\theta'_d \sim \pi(\theta'_d|\theta\backslash\theta_d)$ with $\theta\backslash\theta_d$ kept constant from one iteration to the next. There are two reasons for wanting to do this. Firstly changing only one of the variables at a time is a form of local proposal and can be a beneficial way to make updates, particularly if a random walk proposal is inappropriate. Secondly, if we have access to $\pi(\theta_d|\theta\backslash\theta_d)$ exactly, then we will actually accept every sample as noting that $\theta'\backslash\theta'_d = \theta\backslash\theta_d$ we have

$$\frac{\pi(\theta)\pi(\theta'_d|\theta\backslash\theta_d)}{\pi(\theta')\pi(\theta_d|\theta'\backslash\theta'_d)} = \frac{\pi(\theta_d|\theta\backslash\theta_d)\pi(\theta\backslash\theta_d)\pi(\theta'_d|\theta\backslash\theta_d)}{\pi(\theta'_d|\theta'\backslash\theta'_d)\pi(\theta'\backslash\theta'_d)\pi(\theta_d|\theta'\backslash\theta'_d)} = \frac{\pi(\theta_d|\theta\backslash\theta_d)\pi(\theta\backslash\theta_d)\pi(\theta'_d|\theta\backslash\theta_d)}{\pi(\theta'_d|\theta\backslash\theta_d)\pi(\theta\backslash\theta_d)\pi(\theta_d|\theta\backslash\theta_d)} = 1$$

such that the acceptance probability is always 1. In many models, it will be possible to sample from $\theta'_d \sim \pi(\theta'_d|\theta\backslash\theta_d)$ exactly (e.g. when everything is Gaussian) and thus carry out Gibbs sampling steps exactly. We can cycle through each of the θ_d, either in a random order or in sequence, and apply the appropriate updates. The effectiveness of this approach will depend on the level of correlation between the different variables. The more correlated each variable, the smaller the updates will be for each variable conditioned on the values of the others and the slower the chain will mix. In extreme cases, this does impose stricter conditions for convergence for Gibbs samplers than MH [Roberts and Smith, 1994]. For example, consider an exclusive OR style problem where $\pi(\theta_1, \theta_2) = 1$ if $0 \leq \theta_1 \leq 1$ and $0 \leq \theta_2 \leq 1$ or $-1 \leq \theta_1 < 0$ and $-1 \leq \theta_2 < 0$, and $\pi(\theta_1, \theta_2) = 0$ otherwise. Here there is no way to move from the $[0, 1]^2$ square to the $[-1, 0]^2$ square by updating only one of the variables at a time. As such, a Gibbs sampler would end up stuck in either the positive or negative square.

If it is not possible to sample from the conditional distributions exactly or if it is only possible for some of the variables, we can instead use a *Metropolis-within-Gibbs* approach, also known as *component-wise Metropolis-Hastings*, where we approximate one or more $\pi(\theta'_d|\theta\backslash\theta_d)$ with an appropriate proposal. Naturally this means that the acceptance ratio is no longer always 1 and so an accept-reject step becomes necessary. Though the convergence of this approach has been shown by, for example, Jones et al. [2014], additional assumptions are required compared to the standard Gibbs or MH cases.

5.3.5 Proposal Adaptation

Given the critical importance of the proposal to all the methods we have discussed, it is natural to ask whether we can learn proposals adaptively using previous samples. Though this is indeed possible, see e.g. [Gilks and Wild, 1992; Cappé et al., 2004, 2008; Andrieu and Thoms, 2008], proposal adaptation is a road filled with theoretical and practical perils. Though it is beyond the scope of this work to do this complex line of research justice, we note some possible pitfalls as a warning that proposal adaptation should be carried out with extreme care. Firstly, adaptation

breaks i.i.d. assumptions – samples are naturally generated from different distributions and are further dependent because the previous samples are used to choose the proposal for future samples. Though this itself does not necessarily prevent convergence, it does mean various results no longer hold, e.g. marginal likelihood estimates for importance sampling become biased and the weak law of large numbers can no longer be used to prove convergence. Secondly, one can often unintentionally adapt the proposal into an invalid regime, e.g. by overly reducing its variance in an importance sampling case to give an infinite variance estimator. It can similarly be the case that any fixed proposal is valid, but that adaptation never ceases, such that the longer inference is run, the worse the proposal gets and the estimate never converges. Finally, then if the proposal is state dependent, this can easily break the ergodicity of a Markov chain or cause it to have an incorrect invariant distribution, see for example [Andrieu and Thoms, 2008, Section 2].

5.4 Alternatives to Monte Carlo Inference

Though our focus in this chapter has mostly been on Monte Carlo inference methods, we finish by noting that these are far from the only viable approaches. Two key advantages of Monte Carlo methods are their ubiquitous nature, i.e. many can almost always be applied, and that most commonly used Monte Carlo methods are asymptotically exact, such that given enough time, we can always achieve a required level of accuracy. However, in some scenarios, Monte Carlo methods can be problematically slow to converge and so alternative, asymptotically approximate, methods can be preferable such as variational inference [Blei et al., 2016] and messaging passing methods [Lauritzen and Spiegelhalter, 1988]. Of these, variational inference has become an increasingly popular approach. Its key idea is to reformulate the inference problem to an optimization, by learning parameters of an approximation to the posterior. Typically this involves defining some family of distributions within which the posterior approximation can live, e.g. an exponential distribution family, and then optimizing an *evidence lower bound* (ELBO) with respect to the parameters of this approximation. Doing this implicitly minimizes the Kullback-Leiber divergence between the approximation the target. Variational inference often forms a highly efficient means of calculating a posterior approximation, but, in addition to the obvious bias from using a particular family of distributions for the approximation, it typically requires strong structural assumptions to be made about the form of the posterior. Namely most methods make a so-called *mean-field* assumption that presumes that the posterior factorizes over all latent variables. Its effectiveness is thus critically dependent on the reasonableness of these assumptions.

6

Particle-Based Inference Methods

Particle-based inference methods (PBIM), such as sequential Monte Carlo (SMC) [Gordon et al., 1993; Doucet et al., 2001a] and particle Markov chain Monte Carlo (PMCMC) methods [Andrieu and Roberts, 2009; Rainforth et al., 2016c], are a powerful class of inference algorithms based on propagating populations of samples known as *particles*. By working with populations of samples, also known as *particle systems*, at the same time, rather than individual samples in turn, they allow information to be shared across the population and computational resources to be adaptively reallocated to where they are needed. A common feature of PBIMs is that they make use of *intermediate information* available during the inference process and thereby exploit the structure of the problem. For example, SMC uses a series of intermediate target distributions which act as stepping stones to the full posterior. By using the information gathered from these intermediate solutions, they can better allocate computational resources for the next target and combat the curse of dimensionality. As such, PBIMs can be highly effective for problems with rich, exploitable, structures, for example time series models, but are generally less effective for problems that do not lend themselves to a series of intermediate targets. For our purposes, a particularly important feature of particle-based methods is that they can be used for, and are often very effective at, conducting inference in PPS [Wood et al., 2014], as we will explain in the next chapter.

6.1 Sequential Monte Carlo

6.1.1 Non-Markovian State-Space Models

Although SMC can be used for an arbitrary series of targets as we will explain in Section 6.1.4, we will mostly introduce it in the context of non-Markovian state-space models (NMSSMs). NMSSMs are probabilistic models over a set of latent variables $x_t \in \mathcal{X}_t,\ \forall t = 1 : T$ and observed variables $y_t \in \mathcal{Y}_t, \forall t = 1 : T$. They are similar to the HMM introduced in 3.3, but differ by not making the Markov assumption. This leads to the graphical model shown in Figure 6.1. They are fully defined by an initial density $\mu(x_1)$, a series of transition densities $f_t(x_t|x_{1:t-1})$,

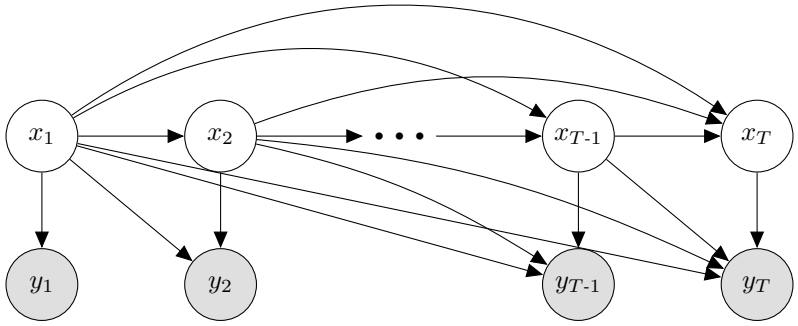


Figure 6.1: DAG for a non-Markovian state space model. Note there are all also multiple dependencies for the nodes summarized by the dots.

and a series of emission densities $g_t(y_t|x_{1:t})$ as follows

$$x_1 \sim \mu(x_1), \tag{6.1a}$$

$$x_t|x_{1:t-1} \sim f_t(x_t|x_{1:t-1}), \tag{6.1b}$$

$$y_t|x_{1:t} \sim g_t(y_t|x_{1:t}). \tag{6.1c}$$

This gives a joint density of

$$p(x_{1:T}, y_{1:T}) = \mu(x_1) \prod_{t=2}^{T} f_t(x_t|x_{1:t-1}) \prod_{t=1}^{T} g_t(y_t|x_{1:t}) \tag{6.2}$$

and the standard relationship that this is proportional to the posterior $p(x_{1:T}|y_{1:T})$. Note the key self-similarity relationship: the intermediate posterior of the first t latent variables given the first t observations is

$$p(x_{1:t}|y_{1:t}) \propto \mu(x_1) \prod_{\tau=2}^{t} f_\tau(x_\tau|x_{1:\tau-1}) \prod_{\tau=1}^{t} g_\tau(y_\tau|x_{1:\tau}).$$

Perhaps surprisingly, this framework can be almost completely general if the x_t and y_t are allowed to take on arbitrary form. For example, if we set $T = 1$, then $x_1 = \theta$ are our variables, $y_1 = \mathcal{D}$ is our data, $\mu(x_1) = p(\theta)$ is our prior, and $g_1(y_1|x_1) = p(\mathcal{D}|\theta)$ is our likelihood. More generally, the initial density and transition densities form terms in the prior, while the emission densities are terms in the likelihood; all of which can take on arbitrary forms. It will often be the case that each "latent variable" is actually a collection of different variables and each "observed variable" is actually a collection of observations. What the NSMSSM formulation allows us to do is express known structure present in the problem: the earlier we are able to put our observations, and thus express their conditional independence from more of the latent variables, the more will be able to exploit this structure.

There are two common tasks that one wishes to carry out for NMSSMs: *filtering* and *smoothing*. Smoothing corresponds to the standard Bayesian inference task where we want to

infer about the latent variables conditioned on all the observations. In filtering we care about the posterior given the observations so far, i.e. $p(x_{1:t}|y_{1:t})$. Filtering is typically done in tasks such as tracking and signal processing where the inference is being done online and the main task is forward prediction. In other words, we do not know all the $y_{1:T}$ upfront but have a series of inference problems where we wish to predict $y_{t+1}, y_{t+2}, \ldots$ given the observations so far $y_{1:t}$. Our focus will be on smoothing, but we note that this is equivalent to the filtering distribution at the last step and so most of the ideas directly transfer.

If the model is in fact Markovian with Gaussian or discrete transition distributions and Gaussian emission distributions, then the posterior can be calculated analytically using the *Rauch-Tung-Striebel* smoother [Rauch et al., 1965] and *forward-backward* algorithm [Rabiner and Juang, 1986] respectively. The former corresponds to the class of Kalman filter [Kalman et al., 1960] and Kalman smoother [Rauch et al., 1965] algorithms. SMC will allow us to perform inference in similar models, amongst many others, without requiring such assumptions to be made. Naturally, this will come at the cost of not having access to a closed-form solution.

6.1.2 Sequential Importance Sampling

In Section 5.3.2.2 we showed that importance sampling weights are multiplicative and thus to sample from a joint distribution, one can first importance sample from a marginal distribution and then importance sample from the respective conditional distribution, with the sample weight corresponding to the product of the two individual weights. More generally, one can carry out *sequential importance sampling* (SIS) to carry out inference on a series of target distributions $(\pi_t(x_{1:t}))_{t=1}^T$ of increasing spaces $\mathbb{X}_1, \ldots, \mathbb{X}_T$ where each $\mathbb{X}_t = \mathcal{X}_1 \times \cdots \times \mathcal{X}_t$, $x_t \in \mathcal{X}_t$ by first making an importance sampling approximation for $\pi(x_1)$ and sequentially update our approximation from $\pi_t(x_{1:t})$ to $\pi_{t+1}(x_{1:t+1})$ by doing importance sampling updates and taking the product of the weights. This is easiest to see in the NMSSM case using the series of targets $p(x_{1:t}|y_{1:t})$, which we can do by first approximating $p(x_1|y_1)$ and then importance sampling each

$$p(x_t|x_{1:t-1}, y_{1:t}) = p(x_t|x_{1:t-1}, y_t) \propto f_t(x_t|x_{1:t-1})g_t(y_t|x_{1:t})$$

noting that x_t is independent of $y_{1:t-1}$ given $x_{1:t-1}$. Presuming a set of proposals $q_1(x_1), q_t(x_t|x_{1:t-1})$, we will calculate importance weights as follows

$$w_1(x_1) = \frac{\mu(x_1)g_t(y_1|x_1)}{q_1(x_1)} \tag{6.3a}$$

$$w_t(x_{1:t}) = w_{t-1}(x_{1:t-1})\frac{g_t(y_t|x_{1:t})f_t(x_t|x_{1:t-1})}{q_t(x_t|x_{1:t-1})}. \tag{6.3b}$$

Algorithm 6.1 Sequential Importance Sampling

Inputs: model $p(x_{1:T}, y_{1:T})$, data $y_{1:T}$, proposals $q_1, \ldots, q_T$, number of samples N
Outputs: weighted samples $\{x^i_{1:T}, w^i_T\}_{i=1}^N$

for $i = 1, \ldots, N$ **do**
  $x^i_1 \sim q_1(x_1)$
  $w^i_1 = \frac{g_1(y_1|x^i_1)\mu(x^i_1)}{q_1(x^i_1)}$
  for $t = 2$ **to** T **do**
    $x^i_t \sim q_t(x_t|x^i_{1:t-1})$
    Set $x^i_{1:t} = (x^i_{1:t-1}, x^i_t)$
    $w^i_t = w^i_{t-1} \frac{g_t(y_t|x^i_{1:t}) f_t(x^i_t|x^i_{1:t-1})}{q_t(x^i_t|x^i_{1:t-1})}$
  end for
end for

Once completed this will produce a set of weighted samples $\{\hat{x}_{1:T}, w_T(\hat{x}_{1:T})\}$ for $p(x_{1:T}|y_{1:T})$. A summary of the SIS process is given in Algorithm 6.1.

Because SIS produces a pure importance sampling estimate, it will share all the desirable properties of importance sampling introduced in Section 5.3.2, including the ability to use self-normalization. In fact, it is just a particular case of importance sampling because we could have produced the same samples and weights by sampling $x_{1:T} \sim q_1(x_1) \prod_{t=2}^T q_t(x_t|x_{1:t-1})$ and then calculating the corresponding importance weight $w_T(x_{1:T})$ in one go. As such, SIS is on its own not very useful. Its utility is realized when it is combined with resampling as we describe next.

6.1.3 SMC for Non-Markovian State Space Models

Sequential Monte Carlo (SMC) [Gordon et al., 1993; Doucet et al., 2001b; Doucet and Johansen, 2009], or particle filtering as it is sometimes known,[1] is a powerful and general purpose inference algorithm that has been successfully applied to a wide range of fields such as signal processing [Candy, 2016], econometrics [Creal, 2012], and probabilistic programming [Wood et al., 2014]. SMC builds on SIS by interleaving the sampling with resampling steps, the latter of which was introduced in Section 5.3.2.4. The key idea is to exploit the structure of a model by breaking down the overall inference problem into a series of target distributions which get incrementally closer to the distribution of interest. Transitioning from one intermediate distribution to the next typically forms a far simpler inference problem than directly approximating the original target. The critical difference to SIS is that the information gained from approximating these intermediate distributions is exploited by reallocating resources to areas likely to have high

[1] We avoid the name particle filtering as it often implies that one is only interested in the filtering distribution, whereas most of the tasks we are interested in will target the smoothing distribution.

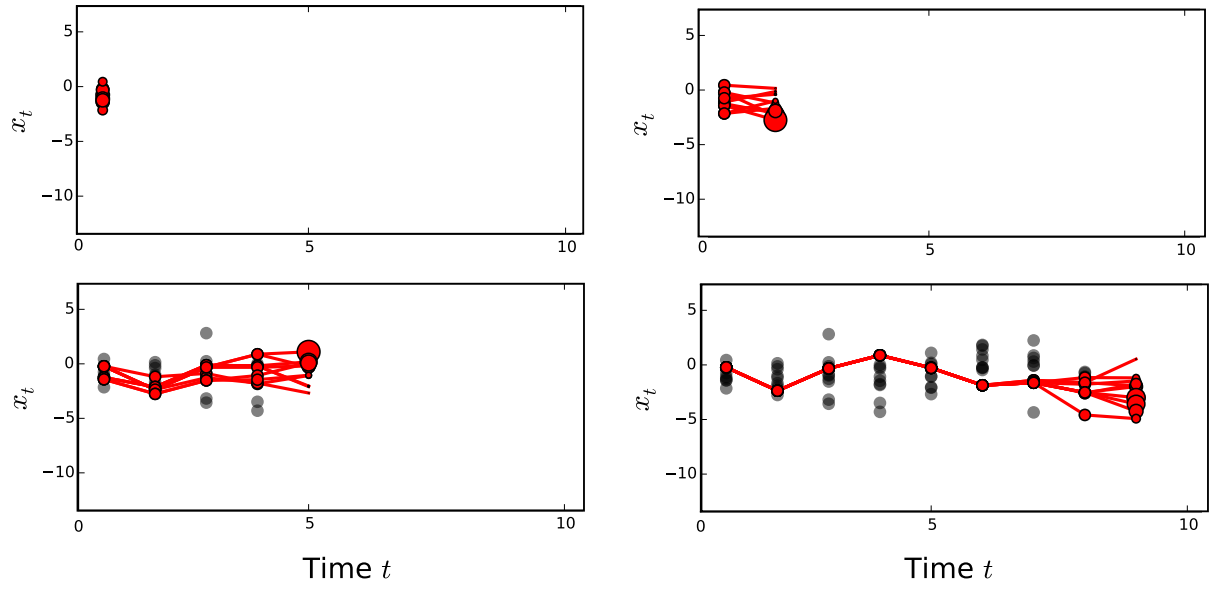


Figure 6.2: Characterization of the SMC method for a state space model. At the first time step (top left) we perform importance sampling to approximate $p(x_1|y_1)$ in the normal way, giving a particle system, or population of weighted samples $\{x_1^i, w_1^i\}_{i=1}^N$. Here each sample is shown by a red blob whose size reflects the weight of the particle. The set of particles are then resampled to produce an unweighted set of particles approximating $p(x_1|y_1)$. For each of these samples, importance sampling is then used again to produce samples for x_2 by sampling from $q_2(x_2|x_1^i)$ and applying a new importance weight (top right). Note that, unlike for SIS, the importance weights are not propagated from one time step to the next. We now again have a weighted particle set $\{x_{1:2}^i, w_2^i\}_{i=1}^N$, for which we again perform a resampling step to produce an unweighted set of particles. The process continues in the same way, eventually returning the final set of samples $\{x_{1:T}^i, \bar{w}_T^i\}_{i=1}^N$, where we have self-normalized the final weights of each trajectory (note that each $\bar{w}_T^i$ applies to the full trajectory $x_{1:T}^i$). Over time, many of the generated samples are discarded by the system (shown in gray) during the resampling step. A consequence of this, which can be seen in the bottom right, is that after many time steps, then our particle set become *degenerate*. Here we have multiple distinct samples for x_9, but all our samples share the same *ancestors* for $t = 1 : 6$, i.e. each x_t^i in our final sample set is the same for $t \leq 6$. We will return to discuss this further in Section 6.2.3.

posterior mass in the final target distribution through resampling. The simplest, but most common form of SMC, which simply interleaves *propagating* samples from one target to the next and resampling the particle system, is sometimes known as *sequential importance resampling* and will be our main focus. A characterization of the method is shown in Figure 6.2.

To see the intuition of why resampling provides the desired resource reallocation, consider what happens to the relative weights of different particles in a population if they are propagated through the SIS algorithm at the same time. It is easy to see that these weights will quickly diverge, typically exponentially quickly (see Section 5.3.3), and our effective sample size (see Section 5.3.2.3) will rapidly diminish. It is now rather pointless to continue to propagate the samples with negligible weight through the system as the chance these will have a significant weight by the end is very low. We, therefore, desire a principled way of "killing off" these low

weight particles and replacing them with higher weight ones. As we showed in Section 5.3.2.4, resampling gives us a method for generating unweighted samples from a population of weighted samples. Though resampling itself only duplicates existing particles, rather than producing new ones, if the duplicate samples are extended independently at the next iteration, this produces distinct samples (albeit heavily correlated ones). In other words, if we have two samples $x_{1:t}^1$ and $x_{1:t}^2$ such that $w_t^2/w_t^1 \approx 0$ then it is more beneficial for us to generate both samples at the next stage using $x_{t+1}^i \sim p(x_{t+1}|x_{1:t}^1, y_{1:t})$ for $i = 1, 2$, than to use $x_{1:t}^2$ to sample x_{t+1}^2. This will not improve our representation of $x_{1:t}$, but it will, in general, give a better representation of the marginal distribution of x_{t+1} and thus the joint distribution $x_{1:t+1}$.

To introduce the SMC method more formally, we consider approximating a sequence of target distributions in same way as for SIS: in our NMSSM case $p(x_{1:t}|y_{1:t}) = p(y_{1:t})^{-1}p(x_{1:t}, y_{1:t}),\ t = 1, \ldots, T$. At each time step t we generate a particle system $\{x_{1:t}^i, w_t^i\}_{i=1}^N$ which provides a weighted approximation to $p(x_{1:t}|y_{1:t})$. Given such a weighted particle system, we convert this to an unweighted particle system by resampling as explained in Section 5.3.2.4. Any of the introduced resampling schemes can be used [Andrieu et al., 2010, Section 4.1] and so in general one should use systematic resampling. When we later introduce PMCMC methods, it will be useful to think about resampling in terms of propagating each particle by drawing an ancestor variable a_{t-1}^i (as per (5.34)) and then using this to propose the new variable for that particle, i.e. $x_t^i \sim q_t(x_t|x_{1:t-1}^{a_{t-1}^i})$. The particle system is then re-weighted as

$$w_t^i = \frac{g_t(y_t|x_{1:t}^i) f_t(x_t^i|x_{1:t-1}^{a_{t-1}^i})}{q_t(x_t^i|x_{1:t-1}^{a_{t-1}^i})}, \tag{6.4}$$

where $x_{1:t}^i = (x_{1:t-1}^{a_{t-1}^i}, x_t^i)$. Note that after resampling, all the particles have the same weight and therefore, unlike for SIS, the weights at the previous step do not feature in the weights at the current time step. This results in a new particle system $\{x_{1:t}^i, w_t^i\}_{i=1}^N$ that approximates $p(x_{1:t}|y_{1:t})$ and the algorithm continues in a self-similar manner up to time T. The overall process is detailed in Algorithm 6.2. Once run, we can use our final, self-normalized, particle set $\{x_{1:T}^i, \bar{w}_T^i\}_{i=1}^N$ as a Monte Carlo estimator in the same manner as self-normalized importance sampling:

$$\mathbb{E}_{\pi(x_{1:T})}\left[f(x_{1:T})\right] \approx \frac{1}{N}\sum_{i=1}^{N} \bar{w}_T^i f(x_{1:T}^i). \tag{6.5}$$

Many intuitions from importance sampling transfer over to SMC. For example, we can construct our posterior estimate and calculate expectations in the same way, while we can also use the effective sample size as a metric of performance (remembering to coalesce identical

Algorithm 6.2 Sequential Monte Carlo (all for $i = 1, \ldots, N$)

Inputs: data $y_{1:T}$, number of particles N, proposals q_t

1: $x_1^i \sim q_1(x_1)$
2: $w_1^i = \frac{g_1(y_1|x_1^i)\mu(x_1^i)}{q_1(x_1^i)}$
3: **for** $t = 2$ **to** T **do**
4: $\quad a_{t-1}^i \sim \text{DISCRETE}\left(\left\{\bar{w}_{t-1}^\ell\right\}_{\ell=1}^N\right)$
5: $\quad x_t^i \sim q_t(x_t|x_{1:t-1}^{a_{t-1}^i})$
6: $\quad$ Set $x_{1:t}^i = (x_{1:t-1}^{a_{t-1}^i}, x_t^i)$
7: $\quad w_t^i = \frac{g_t(y_t|x_{1:t}^i) f_t(x_t^i|x_{1:t-1}^{a_{t-1}^i})}{q_t(x_t^i|x_{1:t-1}^{a_{t-1}^i})}$
8: **end for**

samples). One of the most important features that translates is that SMC produces the unbiased estimate for the marginal likelihood $p(y_{1:T})$

$$\hat{Z} = \prod_{t=1}^{T} \frac{1}{N} \sum_{i=1}^{N} w_t^i. \tag{6.6}$$

Although the original proof of Del Moral [2004] requires advanced techniques we will not touch on here, we will later show that this result can also be shown as a consequence of the proof of correctness for the particle independent Metropolis-Hastings method given in Section 6.2.1. Note that although the marginal likelihood estimate is unbiased, the estimator for a general expectation given by (6.5) is biased in the same way as the self-normalized importance sampling estimator given in (5.26) is. Nonetheless, producing unbiased estimates of the marginal likelihood has numerous significant advantages.

Firstly, it means that one can combine results from difference SMC runs and still have a consistent estimator for the target and a both unbiased and consistent estimate for the marginal likelihood. This is particularly important because the number of particles N, is typically restricted by the availability of memory and we often want to run more than one *SMC sweep* (i.e. single run of SMC). To combine our estimates, one simply weights each sample by the marginal likelihood of its sweep times its local, self-normalized, weight $\bar{w}_T^i$, giving the estimator

$$\mathbb{E}_{\pi(x_{1:T})}\left[f(x_{1:T})\right] \approx \frac{1}{RN} \sum_{r=1}^{R} \sum_{i=1}^{N} \hat{Z}[r] \bar{w}_T^i[r] f(x_{1:T}^i[r]) \tag{6.7}$$

where the notation $[r]$ is used to indicate samples from sweep r. The ability to do this is known as *proper weighting* [Naesseth et al., 2015] and we can justify it by thinking in terms of using a proposal that first samples a full particle set, giving importance weight $\hat{Z}$, and then

sampling a particle from this set, giving importance weight $\bar{w}_T^i$. The product of these separate weights thus gives the weight of each sample.

Secondly, the unbiased marginal likelihood estimate means that we can *nest* SMC within other methods as an unbiased likelihood estimate. Methods that make use of such unbiased likelihood estimates are known as pseudo-marginal methods [Andrieu and Roberts, 2009] and will be discussed in Chapter 10. Some PMCMC methods we introduce in Section 6.2 (namely PIMH and PMMH) can also be interpreted as pseudo-marginal methods.

6.1.4 Sequential Monte Carlo for an Arbitrary Series of Targets

In its most general form, we can use SMC to target an arbitrary series of unnormalized target distributions $\{\gamma_t(x_{1:t})\}_{t=1:T}$, $\gamma_t(x_{1:t}) = Z_t \pi_t(x_{1:t})$, each defined on a space $\mathbb{X}_t$ with strictly increasing dimension[2] $\dim(\mathbb{X}_{t-1}) < \dim(\mathbb{X}_t)$. Here $Z_t := \int \gamma_t(x_{1:t}) dx_{1:t}$ is the normalization constant and $\pi_t(x_{1:t})$ the corresponding normalized target. The only difference that needs to be made from the NMSSM case is in how the weights are calculated, with them now given by

$$w_1^k = \frac{\gamma_1(x_1^k)}{q_1(x_1^k)} \tag{6.8a}$$

$$w_t^k = \frac{\gamma_t(x_{1:t}^k)}{\gamma_{t-1}(x_{1:t-1}^{a_{t-1}^k}) q_t(x_t^i | x_{1:t-1}^{a_{t-1}^i})}. \tag{6.8b}$$

It can be easily seen that the NMSSM weights are a particular case of this, by plugging in $\gamma_t(x_{1:t}) = p(x_{1:t}, y_{1:t})$ into the above expression.

For inference, then our posterior is the last target, $\pi_T(x_{1:T})$, with corresponding marginal likelihood Z_T. Using this more general SMC formulation, we thus see that any series of targets will lead to a valid inference provided that the final target is the desired posterior and that none of our intermediate targets place zero probability mass on events that have non-zero probability mass under the corresponding marginals of the final target

$$\tilde{\pi}_t(x_{1:t}) = \int \pi_T(x_{1:T}) dx_{t+1:T}. \tag{6.9}$$

In fact, the series of targets given for the NMSSM case are actually suboptimal and used because of the fact that $p(x_{1:t}, y_{1:t})$ can, in general, be calculated exactly. It is a well-known result (see, for example, the appendices of Le et al. [2017c]) that the optimal series of proposals (complications with computability and proposals aside) is actually the marginals $\tilde{\pi}_t(x_{1:t}) = p(x_{1:t}|y_{1:T})$ for the NMSSM case. To see this, note that at the t^{th} step, we are generating the samples for x_t and so if each of targets is the marginal of the final distribution $p(x_{1:t}|y_{1:T})$, then the marginal probability

[2]This restriction can be relaxed by using the SMC samplers approach of Del Moral et al. [2006].

of the already sampled variables does not change from one step to the next. Conversely, as $\pi_t(x_{1:t}) \neq \int \pi_{t+1}(x_{1:t+1})dx_{t+1}$ in general, if a different series of targets are used, e.g. $p(x_{1:t}|y_{1:t})$, then the marginal probability of the already sampled variables changes from one iteration to the next, reducing the efficiency of the inference. To give an example of this, consider the case where x_1 includes global parameters that affect each emission distribution, for example it could contain a variance for likelihood terms. In our NMSSM case we can never regenerate samples for x_1 after the first step and so it is clearly better at the first step to target $p(x_1|y_{1:T})$ than $p(x_1|y_1)$, i.e. to condition on all the data rather than the first datapoint. Unfortunately, calculating $p(x_1|y_{1:T})$ itself generally requires a marginalization that is as difficult, or potentially even more difficult, than the original inference itself. Hence the series of targets we introduced in the NMSSM formulation are those that are usually used in practice. However, we note that there are some innovative methods for partially alleviating this problem, such as block sampling [Doucet et al., 2006] methods, which use a combination of lookahead and *auxiliary weighting* schemes [Cappé et al., 2007]. Specifically, they resample using adjusted weights that convey how good the sample is likely to be a few time steps further down the line. The bias this would induce is corrected for by assigning non-even weights to the resampled particles.

Another upshot of the ability to use an arbitrary series of targets is that we can use SMC in scenarios that do not permit a conventional breakdown of the target distribution, but where we can provide guiding distributions to that which we care about. One possible case is when our prior distribution is a generative model that can be evaluated on the target dataset at any point in the generative process. For example, in Janz et al. [2016], we considered doing inference on the structure of a GP kernel. Here we can define a generative model that produces increasingly complex kernels, such that all the intermediate kernels in this generation process are themselves valid kernels that can thus be evaluated. Here we can think of expanding our kernel at each time step of the SMC inference and our targets as being the posterior on kernel structure for the GP, subject to increasingly lax restrictions on the kernel complexity.[3] Another example is provided by Lakshminarayanan et al. [2013], who use SMC for Bayesian learning of decision trees. Here they use the self-similarity of trees, starting with a basic stump and then adding a new split at each time step. For both these problems, SMC has been used to exploit the fact that the overall problem can be broken down into a series of increasingly difficult problems to guide towards good solutions for the final problem.

[3]Technically in this work actually uses population Monte Carlo [Cappé et al., 2004] which is based on using a series of static targets. The intuition, however, predominantly transfers and SMC could have been used instead.

6.1.5 Practical Considerations

6.1.5.1 Use as Many Particles as Possible

The most important practical consideration for SMC is to *use as many particles as possible*. Although, as we showed in 6.1.1, one can run multiple, say R, SMC sweeps with N particles each and then combine them using their marginal likelihood estimates, this is almost exclusively a higher variance (and more skew) estimator than carrying out a single SMC sweep with RN particles, often many orders of magnitude so. It is easy to see why a single sweep is preferable from the point of view of the effect of the different marginal likelihood estimates on the effective sample size. The reason the number of particles can be so critical is that if it is too small, the variance of the marginal likelihood estimate explodes, the effective sample size plummets, and one sweep ends up dominating the estimate. Though the marginal likelihood estimate is unbiased, it has a massive positive skew when insufficient particles as used, such that the probability that a sweep gives a marginal likelihood estimate larger than the true marginal likelihood can be arbitrarily small; after all, the limit $N = 1$ corresponds to importance sampling. As such, if we use insufficient particles and just repeatedly run SMC sweeps, we may have to wait arbitrarily long before getting any useful samples. One way to try and combat this, presuming it is not possible to increase N, is to group the SMC sweeps into a number of islands and then combine estimates in the normal way within the islands, but combine the islands in an unweighted fashion (see for example Lakshminarayanan et al. [2013]). Though often practically useful, this is clearly only trading variance for bias and thus is more of a means of mitigating rather than solving the problem. A more principled means of overcoming this issue is presented by PMCMC methods as discussed in Sections 6.2 and 6.3.

6.1.5.2 Adaptive Resampling

Another important practical consideration is that it is not always beneficial to resample. Resampling obviously throws away information (by reducing the number of distinct particles) and therefore should not be done spuriously. As the reason for doing resampling is to kill off particles with negligible weight, it is generally preferable to avoid the resampling step when most particles have significant weight. This lead to the development of so-called *adaptive resampling* schemes that only perform the resampling step if some criterion is met [Liu and Chen, 1995; Del Moral et al., 2012], e.g. only resampling when the effective sample size falls below a certain threshold, for which $N/2$ is a common choice [Doucet and Johansen, 2009]. When the resampling is omitted, the particles remain weighted, with their weights at the next iteration

updated using (6.3), as per SIS, except that the self-normalized weights $\bar{w}_t^i$ should be used so that the marginal likelihood estimate can be carried out in the same manner. That this still informally forms a valid SMC algorithm for posterior inference can be seen by thinking of these as being sequential importance sampling steps, noting the equivalence to standard importance sampling, and then viewing the process as simply skipping some of the intermediate targets, remembering that it is only the final target that we care about. However, adaptive resampling does cause some complications from a theoretical perspective [Del Moral et al., 2012].

6.1.5.3 Proposals

The practical performance of SMC can be critically dependent on the choice of proposal and so we finish our introduction by considering what constitutes a good proposal, first considering the question of what the theoretically optimal proposal is. Unfortunately, it is not generally possible to establish the optimal proposal for the variance of the final estimate in the finite N case, but we can calculate the so-called *one-step optimal proposal* by minimizing the variance of the individual weights (and by proxy the marginal likelihood estimate) at the next iteration. The one-step optimal proposal is given by

$$q_t^*(x_t|x_{1:t-1}) \propto \frac{\gamma_t(x_{1:t})}{\gamma_{t-1}(x_{1:t-1})} \tag{6.10}$$

$= f_t(x_t|x_{1:t-1})g_t(y_t|x_{1:t})$ for the NMSSM formulation. To see why this is optimal we note that by substituting (6.10) into (6.8) (noting that the proposal needs to be correctly normalized) then the weights are simply the normalization constant for (6.10), namely

$$w_t(x_{1:t}) = \int \frac{\gamma_t(x_{1:t})}{\gamma_{t-1}(x_{1:t-1})} dx_t = \frac{\int \gamma_t(x_{1:t}) dx_t}{\gamma_{t-1}(x_{1:t-1})} \tag{6.11}$$

$= p(y_t|x_{1:t-1})$ for the NMSSM formulation. Consequently, the weights in this case are no longer functions of the sampled value and so the marginal likelihood and effective sample size are deterministic functions of the particle system before proposing the x_t, namely of $\{x_{1:t-1}^i, a_{t-1}^i\}_{i=1}^N$. This also means that we can switch the order of sampling from the proposal and doing the resampling when using $q_t^*(x_t|x_{1:t-1})$, which is preferable as it increases the diversity in the final sample set. Note that the weights are not all the same when using the one-step optimal proposal, because the marginal probability of the particles will change when we incorporate the new observation (or equivalently move to the new target). However, given the expected value of each weight is fixed, i.e. $\mathbb{E}[w_t(x_{1:t})|x_{1:t-1}] = \frac{\int \gamma_t(x_{1:t}) dx_t}{\gamma_{t-1}(x_{1:t-1})}$ regardless of the proposal used, using any proposal other than the optimal one-step proposal only increases the variance of the estimate at the next step. In particular, it is generally not beneficial to use proposals that include some

form of lookahead (e.g. approximating $p(x_t|x_{1:t-1}, y_{1:T})$ instead of $p(x_t|x_{1:t-1}, y_{1:t})$), because the resampling step will correct back to the current target and most likely kill off the "good" samples we generated, simply giving us fewer distinct samples than if we had used the one-step optimal proposal instead.[4] In both the case where the optimal series of targets (6.9) is used and in the large N case, then the one-step optimal proposal is optimal more generally for the whole SMC sweep for a given series of targets. However, because it is not usually possible to analytically calculate the required normalization for $q_t^*(x_t|x_{1:t-1})$, it is not generally tractable. Nonetheless, it provides a good guide for designing, or adapting for, effective proposal distributions.

Another proposal of note, at least for the state space model case, is the so-called bootstrap proposal, which corresponds to sampling from the transition distribution $f_t(x_t|x_{1:t-1})$. The significance of this proposal is that it means that the weight calculation given in (6.4) cancels to

$$w_t^i = g_t(y_t|x_{1:t}^i). \tag{6.12}$$

Consequently, if we use the bootstrap proposal we do not need to be able to evaluate the transition distribution at a particular point, only sample from it. Obviously this can be helpful for computational reasons (though it is rare that the effect of this outweighs using a better proposal if one is known), but it is also important because it can be the case that one only has access to a sampler for the transition and not the ability to evaluate the transition probability at an arbitrary point. In the absence of other information, the bootstrap can also be a reasonable choice of proposal in its own right, as it is almost always more diffuse than the one step posterior. Its efficiency will, in general, depend on how relatively peaked the emission distribution is and the dimensionality of x_t – after all, each step in SMC is in isolation importance sampling. One can usually beneficially use the next observation y_t to help guide the proposal, but doing this in a general purpose manner can be challenging [Gu et al., 2015].

6.2 Particle Markov Chain Monte Carlo Methods

Particle Markov chain Monte Carlo (PMCMC) methods, introduced by Andrieu et al. [2010], make use of sequential Monte Carlo (SMC) algorithms [Gordon et al., 1993; Doucet et al., 2001b] to construct efficient proposals for an MCMC sampler. This allows them to both utilize the ability of SMC to exploit the structure of the target problem and the hill climbing behavior of MCMC methods. The critical difference from running independent SMC sweeps, is that information can be transferred from one sweep to the next. Naturally, this will come with the drawbacks of

[4] Note that lookahead in the proposal is distinct from lookahead using auxiliary weights, thus things like block sampling are still useful.

Algorithm 6.3 Particle Independent Metropolis Hastings

Inputs: number of MCMC iterations R, initial output particle $\mathbf{x}'[0]$ and marginal likelihood $\mathbf{Z}'[0]$ (typically both generated from an SMC sweep)

Outputs: MCMC samples $\{\mathbf{x}'[r]\}_{r=1}^{R}$ where each $\mathbf{x}'[r] \in \mathbb{X}_T$

for $r = 1 : R$ **do**
  Run Algorithm 6.2 (SMC) giving candidate $X_{1:T} = \{x_{1:T}^i, \bar{w}_T^i\}_{i=1}^N$ and $\hat{Z}$
  if $u \leq \min\left(1, \hat{Z}/\mathbf{Z}'[r-1]\right)$ where $u \sim \text{UNIFORM}(0,1)$ **then**
    Sample output particle index b in proportion to weights as per (6.13)
    $\mathbf{x}'[r] \leftarrow x_{1:T}^b \quad \mathbf{Z}'[r] \leftarrow \hat{Z}$ ▷ New sweep accepted
  else
    $\mathbf{x}'[r] \leftarrow \mathbf{x}'[r-1], \quad \mathbf{Z}'[r] \leftarrow \mathbf{Z}'[r-1]$ ▷ New sweep rejected
  end if
end for

MCMC samplers discussed in Section 5.3.4, such as the loss of an unbiased marginal likelihood estimate. However, the advantages will generally outweigh these drawbacks. PMCMC methods will also allow us to explicitly treat global parameters of our system differently to the latent states in a manner that can substantially improve the performance of the model. Though PMCMC methods can be applied to models with arbitrary series of targets in the same manner as SMC, we will stick to the NMSSM case in this section and the next for notational simplicity.

6.2.1 Particle Independent Metropolis Hastings

Particle independent Metropolis Hastings (PIMH) is the simplest PMCMC algorithm. Although in isolation it is not especially useful (as it strictly worse than running independent SMC sweeps and combining them),[5] it is an important theoretical stepping stone to more advanced approaches. The idea from a practical perspective is very simple: use an SMC sweep as an *independent* proposal for an MCMC algorithm targeting $\pi(x_{1:T}) = p(x_{1:T}|y_{1:T})$. From a theoretical perspective, the approach is rather more profound as it can be viewed as a sampler on an *extended space* whose marginal on the returned particles is the target.

The PIMH algorithm is described in Algorithm 6.3. We see that PIMH is an MCMC algorithm where our proposal involves running an independent SMC sweep and then sampling one of the particles from this set in proportion to the weight of the particle. This sample is then accepted or rejected using a MH accept-reject step that utilizes the marginal likelihoods of the sweeps. To be more precise, let $\xi := \{x_t^i\}_{\substack{i=1:N \\ t=1:T}} \bigcup \{a_t^i\}_{\substack{i=1:N \\ t=1:T-1}}$ denote all generated particles and ancestor

[5] There is, though, utility in doing this when including MCMC steps on global parameters Andrieu et al. [2010].

variables of a SMC sampler, and let the associated marginal likelihood estimate be denoted as $\hat{Z}$ which is calculated as per (6.6). Let $\mathbf{x}^i = x_{1:T}^i$ denote one of the final particles and let the index of the final particle we sample be denote as b which is sampled according to

$$b \sim \text{DISCRETE}\left(\left\{\bar{w}_T^\ell\right\}_{\ell=1}^N\right). \tag{6.13}$$

Sampling a particle also indicates an ancestral path which we denote as $\mathbf{b} = (\beta_{1,}, \ldots, \beta_T)$, with $\beta_T = b$ and $\beta_t = a_t^{\beta_{t+1}}$. The result of running a SMC sweep and sampling b thus proposes a $\{\mathbf{x}^b, \mathbf{b}\}$, though in the context of PIMH we only care about the particle $\mathbf{x}^b$. Given this proposal, the acceptance ratio used for PIMH is given by the ratio of the marginal likelihood estimates for the current point and the SMC used to propose the candidate particle. If we denote the MCMC particle sample at iteration r as $\mathbf{x}'[r]$ and associated marginal likelihood as $\mathbf{Z}'[r]$, then our proposed sample $\{\mathbf{x}^b, \hat{Z}\}$ is accepted with probability

$$P(\text{Accept}) = \min\left(1, \frac{\hat{Z}}{\mathbf{Z}'[r-1]}\right) \tag{6.14}$$

forming a MH sampler. Given our MCMC samples, the resulting Monte Carlo estimator is defined in the normal way as

$$\mathbb{E}_{\pi(x_{1:T})}\left[f(x_{1:T})\right] \approx \frac{1}{R}\sum_{r=1}^{R} f(\mathbf{x}'[r]). \tag{6.15}$$

Because our proposal is independent of our current point, this is strictly worse than just running independent SMC sweeps (which can also be thought of as applying waste recycling [Frenkel, 2006] to PIMH). It might seem particularly wasteful to only use one particle from each sweep, but we will be able to use Rao-Blackwellization to justify returning all the particles at each step as we explain in Section 6.2.5.

The key component of proving the correctness of the PIMH algorithm is in showing that it is a valid MH sampler on an *extended target distribution* whose marginal distribution on $\mathbf{x}^b$ is the distribution of interest $\pi(\mathbf{x})$. The density of the distribution induced by running an SMC sweep is given by

$$q_{\text{SMC}}(\xi) = \prod_{i=1}^{N} q_1(x_1^i) \cdot \prod_{t=2}^{T}\prod_{i=1}^{N}\left[\bar{w}_{t-1}^{a_{t-1}^i} q_t(x_t^i | x_{1:t-1}^{a_{t-1}^i})\right] \quad \text{where} \quad \bar{w}_t^i = \frac{w_t^i}{\sum_{\ell=1}^N w_t^\ell} \tag{6.16}$$

and the distribution induced by running an SMC sweep and then choosing particle b

$$q_{\text{PIMH}}(\xi, b) = \bar{w}_T^b q_{\text{SMC}}(\xi) \tag{6.17}$$

Our extended target distribution is now constructed using the following hypothetical process

1. Sample a particle exactly from the target $\mathbf{x}^b \sim \pi(\mathbf{x})$.

2. Sample an ancestral path for the retained particle uniformly at random by independently sampling each $\beta_t \sim \text{UNIFORMDISCRETE}(1, N)$ and setting $\mathbf{b} = (\beta_1, \ldots, \beta_T)$.

3. Sample all of the other particles and ancestor indices conditioned on $\{\mathbf{x}^b, \mathbf{b}\}$ using

$$q_{\text{CSMC}}\left(\xi \backslash \{\mathbf{x}^b, \mathbf{b}\} \mid \mathbf{x}^b, \mathbf{b}\right) = \prod_{i=1, i\neq\beta_1}^{N} q_1(x_1^i) \cdot \prod_{t=2}^{T} \prod_{i=1, i\neq b_t}^{N} \left[\bar{w}_{t-1}^{a_{t-1}^i} q_t(x_t^i | x_{1:t-1}^{a_{t-1}^i})\right], \quad (6.18)$$

which corresponds to $q_{\text{SMC}}(\xi)$ as per (6.16), except for having one particle and path, $\{\mathbf{x}^b, \mathbf{b}\}$, predefined, such that we can think of the sweep as being *conditioned* on $\{\mathbf{x}^b, \mathbf{b}\}$.

We can informally think of this as doing the PIMH process in reverse. Rather than running an SMC sweep and sampling one of the produced particles to give $\{\mathbf{x}^b, \mathbf{b}\}$, it first samples $\{\mathbf{x}^b, \mathbf{b}\}$ and then the rest of the variables in the SMC sweep conditioned on this particle and accompanying ancestral path. Together these steps induce the distribution

$$\tilde{\pi}(\xi, b) = \frac{\pi(\mathbf{x}^b)}{N^T} q_{\text{CSMC}}\left(\xi \backslash \{\mathbf{x}^b, \mathbf{b}\} \mid \mathbf{x}^b, \mathbf{b}\right) \quad (6.19)$$

remembering that as ξ includes all the particles and ancestral paths, sampling ξ and b implies a particular $\{\mathbf{x}^b, \mathbf{b}\}$. By construction (i.e the first step of our hypothetical process), the marginal of this distribution is $\pi(\mathbf{x}^b)$. Therefore, although it is not possible to actually sample from $\tilde{\pi}(\xi, b)$ directly (we started the definition by sampling exactly from the distribution of interest), if we can construct a consistent MCMC estimator on (6.19) then this by proxy produces a consistent estimator for $\pi(\mathbf{x}^b)$, remembering that Monte Carlo samples from a joint distribution have the correct marginal distributions. We can now show that this is exactly what the PIMH algorithm does by explicitly calculating the importance weight implied by targeting $\tilde{\pi}(\xi, b)$ using the proposal $q_{\text{PIMH}}(\xi, b)$

$$\begin{aligned}
\frac{\tilde{\pi}(\xi, b)}{q_{\text{PIMH}}(\xi, b)} &= \frac{\pi(\mathbf{x}^b) q_{\text{CSMC}}\left(\xi \backslash \{\mathbf{x}^b, \mathbf{b}\} \mid \mathbf{x}^b, \mathbf{b}\right)}{N^T \bar{w}_T^b q_{\text{SMC}}(\xi)} = \frac{\pi(\mathbf{x}^b)}{N^T \bar{w}_T^b q_1(x_1^{\beta_1}) \prod_{t=2}^{T} \bar{w}_{t-1}^{\beta_{t-1}} q_t(x_t^{\beta_t} | x_{t-1}^{\beta_{t-1}})} \\
&= \left(\frac{\gamma(\mathbf{x}^b)/Z}{q_1(x_1^{\beta_1}) \prod_{t=2}^{T} q_t(x_t^{\beta_t} | x_{t-1}^{\beta_{t-1}})}\right) \cdot \left(\frac{1}{N^T \bar{w}_T^b \prod_{t=2}^{T} \bar{w}_{t-1}^{\beta_{t-1}}}\right) = \left(\frac{1}{Z} \prod_{t=1}^{T} w_t^{\beta_t}\right) \cdot \left(\frac{1}{N^T \prod_{t=1}^{T} \bar{w}_t^{\beta_t}}\right) \\
&= \left(\frac{1}{Z} \prod_{t=1}^{T} w_t^{\beta_t}\right) \cdot \left(\frac{\prod_{t=1}^{T} \frac{1}{N} \sum_{\ell=1}^{N} w_t^{\ell}}{\prod_{t=1}^{T} w_t^{\beta_t}}\right) = \frac{1}{Z} \prod_{t=1}^{T} \frac{1}{N} \sum_{\ell=1}^{N} w_t^{\ell} = \frac{\hat{Z}}{Z}.
\end{aligned} \quad (6.20)$$

We thus have that $\hat{Z}$ is the importance weight for sampling the unnormalized version of the target $\tilde{\gamma}(\xi, b) = Z\tilde{\pi}(\xi, b)$ and so using the ratio of marginal likelihood estimates is exactly the acceptance ratio required for the MH sampler. We have therefore proven that the PIMH algorithm induces a convergent Markov chain for the target $\pi(x_{1:T})$ and therefore that our estimator given

in (6.15) is consistent. Note that as an aside, this also shows the unbiasedness of the SMC marginal likelihood estimate as we see that

$$\mathbb{E}_{q_{\text{SMC}}(\xi)}\left[\hat{Z}\right] = \mathbb{E}_{q_{\text{PIMH}}(\xi,b)}\left[\hat{Z}\right] = \mathbb{E}_{\tilde{\pi}(\xi,b)}\left[Z\right] = Z. \tag{6.21}$$

6.2.2 Particle Gibbs

One particularly widely used PMCMC algorithm is particle Gibbs (PG). The PG algorithm modifies the SMC step in the PMCMC algorithm to sample the latent variables conditioned on an existing particle trajectory, resulting in what is called a *conditional sequential Monte Carlo* (CSMC) step. The PG method was first introduced as an efficient Gibbs sampler for latent variable models with static parameters [Andrieu et al., 2010]. Since then, the PG algorithm and the extension by Lindsten et al. [2014] have found numerous applications in e.g. Bayesian non-parametrics [Valera et al., 2015; Tripuraneni et al., 2015], probabilistic programming [Wood et al., 2014; van de Meent et al., 2015] and graphical models [Everitt, 2012; Naesseth et al., 2014, 2015]. Whereas PIMH used independent proposals, PG transfers information from one MCMC iteration to the next by using *retained particles*. It, therefore, introduces, typically desirable, hill-climbing behavior, often resulting in improved performance compared with running independent SMC sweeps, particularly for the latter states in the state sequence.

The PG algorithm targets the same extended target $\tilde{\pi}(\xi, b)$ as the PIMH algorithm defined in (6.19), but rather than constructing a MH sampler with independent proposals for this target, it constructs a Gibbs sampler. This is done by iteratively running CSMC sweeps. As described in Algorithm 6.4, a CSMC sweep is similar to a standard, unconditional SMC sweep, except that it starts with an existing *conditional trajectory* $x'_{1:T}$, known as a *retained particle* in the PG context, and runs the sweep conditioned on this conditional trajectory being present in the final sample set. One can think of this as running an SMC sweep conditioned on the predefined particle surviving each of the resampling steps. In other words, the other particles can sample the retained particle as an ancestor but not vice versa, with the values and ancestral path of the retained particle prefixed. Note that running Algorithm 6.4 corresponds to using a fixed choice for the index variables $\mathbf{b} = (N \ldots, N)$. While these indices are used to facilitate the proof of validity of the proposed method, they have no practical relevance (all paths are marginally equivalent by symmetry), and can thus be set to arbitrary values, as is done in Algorithm 6.4, in a practical implementation.

Unlike SMC sweeps, CSMC sweeps can be linked together by conditioning each sweep using a retained particle sampled from the previous sweep with probability proportional to the final particle weights $\bar{w}_T^i$. Therefore, once initialized, say using a standard SMC sweep,

Algorithm 6.4 Conditional sequential Monte Carlo

Inputs: data $y_{1:T}$, number of particles N, proposals q_t, conditional trajectory $x'_{1:T}$

1: $x_1^i \sim q_1(x_1),\ i = 1, \ldots, N-1$ and set $x_1^N = x'_1$
2: $w_1^i = \frac{g_1(y_1|x_1^i)\mu(x_1^i)}{q_1(x_1^i)},\ i = 1, \ldots, N$
3: **for** $t = 2$ **to** T **do**
4: $\quad a_{t-1}^i \sim \text{DISCRETE}\left(\left\{\bar{w}_{t-1}^\ell\right\}_{\ell=1}^N\right),\ i = 1, \ldots, N-1$
5: $\quad x_t^i \sim q_t(x_t|x_{1:t-1}^{a_{t-1}^i}),\ i = 1, \ldots, N-1$
6: $\quad$ Set $a_{t-1}^N = N$ and $x_t^N = x'_t$
7: $\quad$ Set $x_{1:t}^i = (x_{1:t-1}^{a_{t-1}^i}, x_t^i),\ i = 1, \ldots, N$
8: $\quad w_t^i = \frac{g_t(y_t|x_{1:t}^i) f_t(x_t^i|x_{1:t-1}^{a_{t-1}^i})}{q_t(x_t^i|x_{1:t-1}^{a_{t-1}^i})},\ i = 1, \ldots, N$
9: **end for**

one can iterate between sampling a retained particle in the same manner ancestor indices are sampled and running CSMC sweeps conditioned on this retained particle. This is known as iterated CSMC (ICSMC) and, considerations about global parameters aside (see Section 6.2.4), constitutes the PG algorithm. Note that the resampling step must always be done for CSMC sweeps (i.e. the techniques introduced in Section 6.1.5.2 cannot be used), as, unlike in an SMC sweep, the resampling for a CSMC sweep does not maintain the target distribution. Another important difference is that, unlike SMC, a CSMC sweep does not provide an unbiased estimate for the marginal likelihood (expect in the limit $N \to \infty$).

The theoretical justification for the PG algorithm can be shown in a similar manner as the PIMH algorithm. The sampled index b now corresponds the index of the retained particle and we can think of the approach as alternating between Gibbs updates on b and $\xi \backslash \{\mathbf{x}^b, \mathbf{b}\}$. In other words, we alternate between sampling

$$b \sim \tilde{\pi}(b|\xi) = \frac{\tilde{\pi}(\xi, b)}{\int \tilde{\pi}(\xi, b) db} \tag{6.22a}$$

$$\xi \backslash \{\mathbf{x}^b, \mathbf{b}\} \sim \tilde{\pi}(\xi \backslash \{\mathbf{x}^b, \mathbf{b}\}|\mathbf{x}^b, \mathbf{b}) = \frac{\tilde{\pi}(\xi, b)}{\iint \tilde{\pi}(\xi, b) d\xi \backslash \{\mathbf{x}^b, \mathbf{b}\}}. \tag{6.22b}$$

The key step to seeing that the PG algorithm samples from these distributions is noting that (6.18) corresponds to the distribution induced by running a CSMC sweep with retained particle $\{\mathbf{x}^b, \mathbf{b}\}$. By construction then $\tilde{\pi}(\xi \backslash \{\mathbf{x}^b, \mathbf{b}\}|\mathbf{x}^b, \mathbf{b}) = q_{\text{CSMC}}(\xi \backslash \{\mathbf{x}^b, \mathbf{b}\}|\mathbf{x}^b, \mathbf{b})$ and so each CSMC sweep samples exactly from (6.22b). To complete the derivation we need to show that $\tilde{\pi}(b|\xi) = \bar{w}_T^b$ so that choosing the retained particle corresponds exactly to sampling from (6.22a). This

can be done by noting that

$$\int \tilde{\pi}(\xi, b)db = \int q_{\text{PIMH}}(\xi, b)\frac{\hat{Z}}{Z}db = \int \bar{w}_T^b q_{\text{SMC}}(\xi)\frac{\hat{Z}}{Z}db = q_{\text{SMC}}(\xi)\frac{\hat{Z}}{Z}\int \bar{w}_T^b db = q_{\text{SMC}}(\xi)\frac{\hat{Z}}{Z}$$

and therefore

$$\tilde{\pi}(b|\xi) = \frac{\tilde{\pi}(\xi, b)}{q_{\text{SMC}}(\xi)\frac{\hat{Z}}{Z}} = \frac{q_{\text{PIMH}}(\xi, b)\frac{\hat{Z}}{Z}}{q_{\text{SMC}}(\xi)\frac{\hat{Z}}{Z}} = \bar{w}_T^b$$

as required. We thus have that both (6.22a) and (6.22b) are valid Gibbs updates on our extended target $\tilde{\pi}(\xi, b)$. Now applying each of (6.22a) and (6.22b) once can change all of the variables in our target as it can change the retained particle, the index b, and all the other particles.[6] Therefore the combination of the two updates forms a valid Gibbs sampler for the target $\tilde{\pi}(\xi, b)$. Because $\tilde{\pi}(\xi, b)$ has a marginal distribution on the returned samples $\pi(\mathbf{x}^b)$ which is equal to the desired target, we have thus proven that using the PG algorithm and calculating estimates in the manner defined in (6.15) leads to a consistent estimator.

6.2.3 Path Degeneracy

A drawback of particle-based methods, and in particular the PG algorithm, is that they can be adversely affected by *path degeneracy* in the (C)SMC sweep. Path degeneracy is the tendency of the number of unique samples to diminish as one follows the ancestry backward through the sweep. In other words, because of the resampling, there will typically be fewer (and never more) unique instances of x_{t-1} than of x_t in the final particle set $\{x_{1:T}^i, \bar{w}_T^i\}_{i=1}^T$ and therefore there are often few distinct samples for the earlier steps by the end of the sweep. It is not uncommon for the ancestral paths to collapse back to a single sample for small t, an example of which is demonstrated in the bottom right of Figure 6.2 where there is a collapse to a single trajectory for $t \leq 6$. Degeneracy can be especially detrimental when the posterior on the early latent variables is significantly affected by later observations, e.g. when some of the early latent variables correspond to global parameters, as this creates a significant mismatch between the intermediate targets. Consequently, observations should be made as early as possible in the state sequence when taking a particle-based inference approach, e.g. placing `observe` statements as early as possible in probabilistic programming query.

For methods that run independent SMC sweeps such as SMC itself and PIMH, the detrimental impact of path degeneracy is generally seen through high variance of the marginal likelihood estimates, resulting in low effective sample sizes and or slow mixing rates. For PG then

[6] Note also that applying the update cycle twice can potentially completely remove the retained particle from the system and so we need not worry about the possibility of the retained particle simply being "passed around".

path degeneracy can be even more catastrophic. Conditioning on an existing trajectory means that whenever resampling of the trajectories results in a common ancestor, this ancestor must correspond to this trajectory. For a PG sampler, any common ancestor is, by construction, guaranteed to be equal to the retained particle, such that the early part of the trajectory in the retained particle cannot change from one iteration to the next if the particle set has coalesced to a single ancestor. This results in high correlation between the samples, and poor mixing of the Markov chain. Approaches for alleviating this degeneracy include resample-move methods [Chopin et al., 2013], which rejuvenate the particle set by interleaving resampling with MCMC updates for the *full* trace, and, for the particular case of PG, ancestor sampling [Lindsten et al., 2014] methods, which sample new ancestors for the retained particle as the CSMC sweep is run. Though highly effective, for many models these methods cause a quadratic increase in the computational cost with the length of the state space, meaning that they are often prohibitively expensive in practice. In Section 6.3 we introduce an alternative, and often complementary, means of overcoming degeneracy problems in the form of the interacting PMCMC algorithm [Rainforth et al., 2016c].

6.2.4 Global Parameters

An important feature of PMCMC methods that we have thus far omitted is that they allow for distinct treatment of global parameters. Imagine we are interested in sampling from $p(\theta, x_{1:T}|y_{1:T})$ where θ are some global parameters that are of particular importance as they directly affect all the other variables, e.g. because they are effect each transition and or emission distribution. As we explained in Section 6.2.3, SMC / PMCMC methods can do relatively poorly on sampling the earlier parameters, particularly if these have long-range dependencies. One may often therefore wish to use a different sampling scheme for the global parameters, which is possible in PMCMC methods using either Gibbs sampling or Metropolis-within-Gibbs sampling to update θ, rather than including them in the SMC sweep.

Perhaps the simplest way of doing this is using the *particle marginal Metropolis-Hastings* (PMMH) sampler. This extends the PIMH sampler by using a standard MH update proposal $q(\hat{\theta}|\theta = \theta[r-1])$ before each SMC sweep such that the overall proposal is

$$q_{\text{PMMH}}(\xi, \hat{\theta}|\theta[r-1]) = q(\hat{\theta}|\theta[r-1])q_{\text{PIMH}}(\xi|\hat{\theta}) \tag{6.23}$$

and samples are now accepted with probability

$$P(\text{Accept}) = \min\left(1, \frac{\hat{Z}q(\theta[r-1]|\hat{\theta})}{\mathbf{Z}'[r-1]q(\hat{\theta}|\theta[r-1])}\right). \tag{6.24}$$

Typically more effectively, one can also use manual updates for θ in the PG algorithm by alternating between running CSMC sweeps, sampling the index of the retained particle, and updating θ given the retained particle (note the difference to PMMH where θ is updated independently of $\mathbf{x}'[r]$). Here one either relies on being able to sample from $\theta \sim p(\theta|x_{1:T} = \mathbf{x}'[r], y_{1:T})$, which is occasionally possible, or uses a MH step to update θ targeting $p(\theta|x_{1:T} = \mathbf{x}'[r], y_{1:T})$ as a Metropolis-with-Gibbs update. In the latter case ,it is typically preferable to carry out multiple update steps on the parameters between each CSMC sweep to ensure fast mixing, noting that the latter is generally at least a factor of N more times expensive.

6.2.5 Rao-Blackwellization

At each MCMC iteration r, we generate N full particle trajectories. Using only one of these as in (6.15) might seem a bit wasteful. We can, however, make use of all particles to estimate expectations of interest for all of the PMCMC methods introduced so far by analytically averaging over the possible values of b that could be sampled at each iteration. We can do this noting that $\mathbb{E}\left[\frac{1}{R}\sum_{r=1}^{R} f(\mathbf{x}'[r])\right] = \frac{1}{R}\sum_{r=1}^{R}\mathbb{E}\left[\mathbb{E}\left[f(\mathbf{x}'[r])|\xi\right]\right]$, and then replacing $f(\mathbf{x}'[r])$ in (6.15) by the analytically calculable

$$\mathbb{E}\left[f(\mathbf{x}'[r])|\xi\right] = \mathbb{E}\left[f(x^b_{1:T})\middle|\xi\right] = \sum_{i=1}^{N} \bar{w}^i_{T,m} f(\mathbf{x}^i_m), \tag{6.25}$$

where the expectation is over the sampling of b and the second equality follows from the definition of the expectation of a discrete distribution, remembering $b \sim \text{DISCRETE}\left(\{\bar{w}^i_T\}_{i=1}^N\right)$. This procedure is known as *Rao-Blackwellization* of a statistical estimator and is (in terms of variance) never detrimental [Casella and Robert, 1996]. At a high level, we are taking an analytic expectation over part of the randomness of the system, namely the sampling of b.

6.3 Interacting Particle Markov Chain Monte Carlo

In this section, we introduce *interacting particle Markov chain Monte Carlo* (iPMCMC) [Rainforth et al., 2016c], a PMCMC method based on an interacting pool of standard and conditional sequential Monte Carlo samplers. Like other PMCMC methods, iPMCMC is a Markov chain Monte Carlo sampler on an extended space. In iPMCMC we run a pool of CSMC and unconditional SMC algorithms as parallel processes that we refer to as nodes. After each run of this pool, we apply successive Gibbs updates to the indexes of the CSMC nodes, such that the indices of the CSMC nodes changes. Hence, the nodes from which retained particles are sampled can change from one MCMC iteration to the next, reducing the sensitivity to path degeneracy relative to PG by trading off exploration (SMC) and exploitation (CSMC)

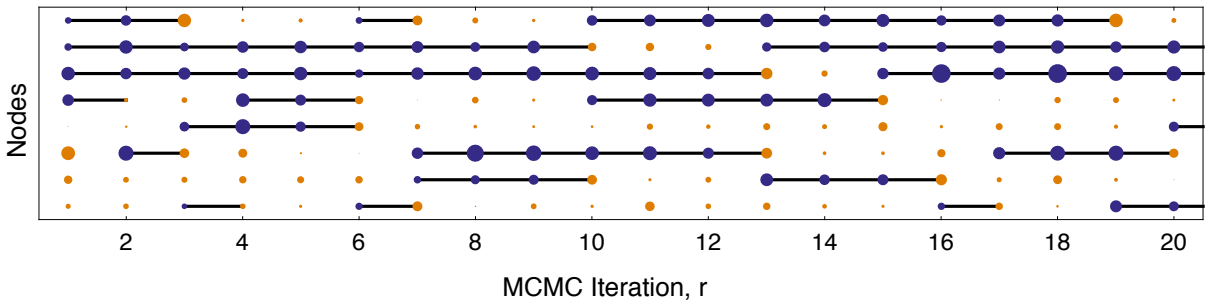


Figure 6.3: Changing of conditional node indices in iPMCMC algorithm. Blue/orange nodes respectively run CSMC/SMC at the next iteration. Circle sizes correspond to the marginal likelihood estimate produced by the sweep $\hat{Z}_m$. Lines signify passing a retained particle. We thus see that each of the nodes has breaks in the passing of retained particles, which is desirable as whenever a node switches from SMC to CSMC, we must generate an entirely new retained particle. We also see that nodes with higher marginal likelihood estimates are more likely to become CSMC nodes at the next iteration.

to achieve improved mixing of the Markov chains. Crucially, the pool provides numerous candidate indices at each Gibbs update, giving a significantly higher probability that an entirely new retained particle will be "switched in" than in non-interacting alternatives. A high-level characterization of iPMCMC is given in Figure 6.3.

We prove that iPMCMC is a partially collapsed Gibbs sampler on the extended space containing the particle sets for all nodes. In the special case where iPMCMC uses only *one* CSMC node, it can in fact be seen as a non-trivial and unstudied instance of the α-SMC-based [Whiteley et al., 2016] PMCMC method introduced by Huggins and Roy [2015]. However, with iPMCMC we extend this further to allow for an arbitrary number of CSMC and standard SMC algorithms with interaction.

The interaction between nodes requires only minimal communication; each node must report an estimate of the marginal likelihood and receive a new role (SMC or CSMC) for the next sweep. This means that iPMCMC can be run in a distributed manner on multiple computers. However, the advantages of iPMCMC go far beyond simple parallelization: our experimental evaluation shows that iPMCMC outperforms both equivalent non-interacting PMCMC samplers as well as a single PG sampler with the same number of particles run longer to give a matching computational budget. An implementation of iPMCMC is provided in the probabilistic programming system *Anglican*[7] [Wood et al., 2014] as we discuss in Chapter 7.4. A general purpose toolbox for carrying out iPMCMC and other SMC / PMCMC inference in MATLAB is also provided by the custom-made *probabilistic MATLAB* package.[8]

[7] `http://www.robots.ox.ac.uk/~fwood/anglican`
[8] `http://github.com/twgr/probabilistic_matlab`

Algorithm 6.5 iPMCMC sampler

Inputs: number of nodes M, conditional nodes P and MCMC steps R, initial $\mathbf{x}'_{1:P}[0]$

1: **for** $r = 1$ **to** R **do**
2: Workers $1 : M \backslash c_{1:P}$ run Algorithm 6.2 (SMC)
3: Workers $c_{1:P}$ run Algorithm 6.4 (CSMC), conditional on $\mathbf{x}'_{1:P}[r-1]$ respectively.
4: **for** $j = 1$ **to** P **do**
5: Select a new conditional node by simulating c_j according to (6.26).
6: Set new MCMC sample $\mathbf{x}'_j[r] = \mathbf{x}_{c_j}^{b_j}$ by simulating b_j according to (6.28)
7: **end for**
8: **end for**

6.3.1 Method

The main goal of iPMCMC is to increase the efficiency of PMCMC, in particular, PG. As we explain in Section 6.2.3, PG is especially susceptible to the *path degeneracy* effect of SMC samplers because the early time steps in the state-sequence can become stuck in the same position for long periods if the ancestral paths in the CSMC sweeps typically coalesce to a single sample, which is by construction guaranteed to be the retained particle. This results in high correlation between the samples, and poor mixing of the Markov chain. To counteract this we might need a very high number of particles to get good mixing for all latent variables $x_{1:T}$, which can be infeasible due to e.g. limited available memory. iPMCMC can alleviate this issue by, from time to time, switching out a CSMC particle system with a completely independent SMC one, resulting in improved mixing.

iPMCMC, summarized in Algorithm 6.5, consists of M interacting separate CSMC and SMC algorithms, exchanging only very limited information at each iteration to draw new MCMC samples. We will refer to these internal CSMC and SMC algorithms as nodes, and assign an index $m = 1, \ldots, M$. At every iteration, we have P nodes running local CSMC algorithms, with the remaining $M - P$ nodes running independent SMC. The CSMC nodes are given an identifier $c_j \in \{1, \ldots, M\},\ j = 1, \ldots, P$ with $c_j \neq c_k,\ k \neq j$ and we write $c_{1:P} = \{c_1, \ldots, c_P\}$. Let $\mathbf{x}_m^i = x_{1:T,m}^i$ be the internal particle trajectories of node m.

Suppose we have access to P trajectories $\mathbf{x}'_{1:P}[0] = (\mathbf{x}'_1[0], \ldots, \mathbf{x}'_P[0])$ corresponding to the initial retained particles, where the index $[\cdot]$ denotes MCMC iteration. At each iteration r, the nodes $c_{1:P}$ run CSMC (Algorithm 6.4) with the previous MCMC sample $\mathbf{x}'_j[r-1]$ as the retained particle. The remaining $M - P$ nodes run standard (unconditional) SMC, i.e. Algorithm 6.2. Each node m returns an estimate of the marginal likelihood $\hat{Z}_m$ for the internal

particle system calculated as per (6.6). Note that whereas for the SMC sweeps this is an unbiased estimate of the marginal likelihood, this a biased estimator for the CSMC sweeps that is being specifically defined for our purposes.

The new conditional nodes are then set using a single loop $j = 1 : P$ of Gibbs updates, sampling new indices c_j where

$$\mathbb{P}(c_j = m | c_{1:P\setminus j}) = \hat{\zeta}_m^j \tag{6.26}$$

$$\text{and} \quad \hat{\zeta}_m^j = \frac{\hat{Z}_m \mathbb{1}_{m \notin c_{1:P\setminus j}}}{\sum_{n=1}^M \hat{Z}_n \mathbb{1}_{n \notin c_{1:P\setminus j}}}, \tag{6.27}$$

defining $c_{1:P\setminus j} = \{c_1, \ldots, c_{j-1}, c_{j+1}, \ldots, c_P\}$. We thus loop once through the conditional node indices and resample them from the union of the current node index and the unconditional node indices[9], in proportion to their marginal likelihood estimates. This is the key step that lets us switch completely the nodes from which the retained particles are drawn.

One MCMC iteration r is concluded by setting the new samples $\mathbf{x}'_{1:P}[r]$ by simulating from the corresponding conditional node's, c_j, internal particle system in the same way as b was sampled for the PG algorithm

$$\mathbb{P}(b_j = i | c_j) = \bar{w}_{T,c_j}^i, \quad \mathbf{x}'_j[r] = \mathbf{x}_{c_j}^{b_j}. \tag{6.28}$$

The potential to pick from updated nodes c_j, having run independent SMC algorithms, decreases correlation and improves mixing of the MCMC sampler. Furthermore, as each Gibbs update corresponds to a one-to-many comparison for maintaining the same conditional index, the probability of switching is much higher than in an analogous non-interacting system.

The theoretical justification for iPMCMC is independent of how the initial trajectories $\mathbf{x}'_{1:P}[0]$ are generated. One simple and effective method (that we use in our experiments) is to run standard SMC sweeps for the "conditional" nodes at the first iteration.

The iPMCMC samples $\mathbf{x}'_{1:P}[r]$ can be used to estimate expectations for test functions $f : \mathsf{X}^T \mapsto \mathbb{R}$ in the standard Monte Carlo sense, with

$$\mathbb{E}[f(\mathbf{x})] \approx \frac{1}{RP} \sum_{r=1}^R \sum_{j=1}^P f(\mathbf{x}'_j[r]). \tag{6.29}$$

However, we can improve upon this using Rao-Blackwellization if we have access to all particles generated by the algorithm, see Section 6.3.3.

[9] Unconditional node indices here refers to all $m \notin c_{1:P}$ at that point in the loop. It may thus include nodes who just ran a CSMC sweep, but have been "switched out" earlier in the loop.

We note that iPMCMC is suited to distributed and multi-core architectures. In practice, the particle to be retained, should the node be a conditional node at the next iteration, can be sampled upfront and discarded if unused. Therefore, at each iteration, only a single particle trajectory and normalization constant estimate need be communicated between the nodes, whilst the time taken for calculation of the updates of $c_{1:P}$ is negligible. Further, iPMCMC should be amenable to an asynchronous adaptation under the assumption of a random execution time, independent of $\mathbf{x}'_j[r-1]$ in Algorithm 6.5. We leave this asynchronous variant to future work.

6.3.2 Theoretical Justification

In this section, we will give some crucial results to justify the proposed iPMCMC sampler. We start by defining some additional notation, much of which is duplicated from the PIMH introduction. Let $\xi := \{x_t^i\}_{\substack{i=1:N\\t=1:T}} \bigcup \{a_t^i\}_{\substack{i=1:N\\t=1:T-1}}$ denote all generated particles and ancestor variables of a (C)SMC sampler. We write ξ_m when referring to the variables of the sampler local to node m. Let the conditional particle trajectory and corresponding ancestor variables for node c_j be denoted by $\{\mathbf{x}_{c_j}^{b_j}, \mathbf{b}_{c_j}\}$, with $\mathbf{b}_{c_j} = (\beta_{1,c_j}, \ldots, \beta_{T,c_j})$, $\beta_{T,c_j} = b_j$ and $\beta_{t,c_j} = a_{t,c_j}^{\beta_{t+1,c_j}}$. Let the posterior distribution of the latent variables be denoted by $\pi(\mathbf{x}) := p(x_{1:T}|y_{1:T})$ with normalization constant $Z := p(y_{1:T})$. Finally we note that the SMC and CSMC algorithms induce the respective distributions over the random variables generated by the procedures are given by (6.16) and (6.18)

Now we are ready to state the main theoretical result.

Theorem 6.1. *The interacting particle Markov chain Monte Carlo sampler of Algorithm 6.5 is a partially collapsed Gibbs sampler [Van Dyk and Park, 2008] for the target distribution*

$$\tilde{\pi}(\xi_{1:M}, c_{1:P}, b_{1:P}) = \frac{1}{N^{PT}\binom{M}{P}} \prod_{\substack{m=1\\m\notin c_{1:P}}}^{M} q_{SMC}(\xi_m) \cdot \prod_{j=1}^{P} \left[\pi\left(\mathbf{x}_{c_j}^{b_j}\right) \mathbb{1}_{c_j \notin c_{1:j-1}} q_{CSMC}\left(\xi_{c_j} \backslash \{\mathbf{x}_{c_j}^{b_j}, \mathbf{b}_{c_j}\} \mid \mathbf{x}_{c_j}^{b_j}, \mathbf{b}_{c_j}\right)\right]. \quad (6.30)$$

Proof. The proof follows similar ideas as our derivation for the PIMH and PG samplers and thus follows similar ideas to those introduced by Andrieu et al. [2010]. We prove that the interacting particle Markov chain Monte Carlo sampler is, in fact, a standard partially collapsed Gibbs sampler [Van Dyk and Park, 2008] on an extended space

$$\Upsilon := \mathsf{X}^{\otimes MTN} \times [N]^{\otimes M(T-1)N} \times [M]^{\otimes P} \times [N]^{\otimes P}.$$

With $\tilde{\pi}(\cdot)$ with as per (6.30), we will show that following the Gibbs sampler on Υ

$$\xi_{1:M} \backslash \{\mathbf{x}_{c_{1:P}}^{b_{1:P}}, \mathbf{b}_{c_{1:P}}\} \sim \tilde{\pi}(\cdot \,|\mathbf{x}_{c_{1:P}}^{b_{1:P}}, \mathbf{b}_{c_{1:P}}, c_{1:P}, b_{1:P}), \tag{6.31a}$$

$$c_j \sim \tilde{\pi}(\cdot \,|\xi_{1:M}, c_{1:P\backslash j}), \;\; j = 1, \ldots, P, \tag{6.31b}$$

$$b_j \sim \tilde{\pi}(\cdot \,|\xi_{1:M}, c_{1:P}), \;\; j = 1, \ldots, P, \tag{6.31c}$$

is equivalent to the iPMCMC method laid out in Algorithm 6.5.

First, the initial step (6.31a) corresponds to sampling from

$$\tilde{\pi}(\xi_{1:M} \backslash \{\mathbf{x}_{c_{1:P}}^{b_{1:P}}, \mathbf{b}_{c_{1:P}}\} | \mathbf{x}_{c_{1:P}}^{b_{1:P}}, \mathbf{b}_{c_{1:P}}, c_{1:P}, b_{1:P}) = \prod_{m=1, m \notin c_{1:P}}^{M} q_{\text{SMC}}(\xi_m) \prod_{j=1}^{P} q_{\text{CSMC}}\left(\xi_{c_j} \backslash \{\mathbf{x}_{c_j}^{b_j}, \mathbf{b}_{c_j}\} \mid \mathbf{x}_{c_j}^{b_j}, \mathbf{b}_{c_j}, c_j, b_j\right).$$

Given the conditional trajectories, this just corresponds to steps 3–4 in Algorithm 6.5, i.e. running P CSMC and $M - P$ SMC algorithms independently. We continue with a reformulation of (6.30) which will be useful to prove correctness for the other two steps. Here we first note that

$$\tilde{\pi}(\xi_{1:M}, c_{1:P}, b_{1:P}) = \frac{1}{\binom{M}{P}} \prod_{m=1}^{M} q_{\text{SMC}}(\xi_m) \cdot \prod_{j=1}^{P} \left[\mathbb{1}_{c_j \notin c_{1:j-1}} \bar{w}_{T,c_j}^{b_j} \frac{\pi\left(\mathbf{x}_{c_j}^{b_j}\right) q_{\text{CSMC}}\left(\xi_{c_j} \backslash \{\mathbf{x}_{c_j}^{b_j}, \mathbf{b}_{c_j}\} \mid \mathbf{x}_{c_j}^{b_j}, \mathbf{b}_{c_j}, c_j, b_j\right)}{N^T \bar{w}_{T,c_j}^{b_j} q_{\text{SMC}}\left(\xi_{c_j}\right)} \right]. \tag{6.32}$$

Now by self similarity to importance weight calculated for the PIMH derivation in (6.20)

$$\frac{\pi\left(\mathbf{x}_{c_j}^{b_j}\right) q_{\text{CSMC}}\left(\xi_{c_j} \backslash \{\mathbf{x}_{c_j}^{b_j}, \mathbf{b}_{c_j}\} \mid \mathbf{x}_{c_j}^{b_j}, \mathbf{b}_{c_j}, c_j, b_j\right)}{N^T \bar{w}_{T,c_j}^{b_j} q_{\text{SMC}}\left(\xi_{c_j}\right)} = \frac{\hat{Z}_{c_j}}{Z} \tag{6.33}$$

and therefore

$$\tilde{\pi}(\xi_{1:M}, c_{1:P}, b_{1:P}) = \frac{1}{\binom{M}{P}} \prod_{m=1}^{M} q_{\text{SMC}}(\xi_m) \cdot \prod_{j=1}^{P} \frac{\hat{Z}_{c_j}}{Z} \mathbb{1}_{c_j \notin c_{1:j-1}} \bar{w}_{T,c_j}^{b_j}. \tag{6.34}$$

By marginalizing this reformulation over $b_{1:P}$ we get

$$\tilde{\pi}(\xi_{1:M}, c_{1:P}) = \frac{1}{\binom{M}{P}} \prod_{m=1}^{M} q_{\text{SMC}}(\xi_m) \prod_{j=1}^{P} \frac{\hat{Z}_{c_j}}{Z} \mathbb{1}_{c_j \notin c_{1:j-1}}.$$

From this it is easy to see that $\tilde{\pi}(c_j | \xi_{1:M}, c_{1:P\backslash j}) = \hat{\zeta}_{c_j}^{j}$, which corresponds to sampling the conditional node indices, i.e. step 6 in Algorithm 6.5. Finally, from (6.34) we can see that simulating $b_{1:P}$ can be done independently as follows

$$\tilde{\pi}(b_{1:P} | \xi_{1:M}, c_{1:P}) = \frac{\tilde{\pi}(b_{1:P}, \xi_{1:M}, c_{1:P})}{\tilde{\pi}(\xi_{1:M}, c_{1:P})} = \prod_{j=1}^{P} \bar{w}_{T,c_j}^{b_j}.$$

This corresponds to step 7 in Algorithm 6.5. We now have that the procedure defined by (6.31) is a partially collapsed Gibbs sampler, derived from (6.30), and we have shown that it is exactly

equal to the iPMCMC sampler described in Algorithm 6.5. □

Remark 1. *The marginal distribution of* $(\mathbf{x}_{c_{1:P}}^{b_{1:P}}, c_{1:P}, b_{1:P})$*, with* $\mathbf{x}_{c_{1:P}}^{b_{1:P}} = (\mathbf{x}_{c_1}^{b_1}, \ldots, \mathbf{x}_{c_P}^{b_P})$*, under* (6.30) *is by construction given by*

$$\tilde{\pi}\left(\mathbf{x}_{c_{1:P}}^{b_{1:P}}, c_{1:P}, b_{1:P}\right) = \frac{\prod_{j=1}^{P} \pi\left(\mathbf{x}_{c_j}^{b_j}\right) \mathbb{1}_{c_j \notin c_{1:j-1}}}{N^{PT}\binom{M}{P}} \tag{6.35}$$

because our extended target is formulated as being the process of sampling exactly from (6.35) *and then running the* P *CSMC sweeps and* $M - P$ *SMC sweeps. This means that each trajectory* $\mathbf{x}_{c_j}^{b_j}$ *is marginally distributed according to the posterior distribution of interest,* π*. Indeed, the* P *retained trajectories of iPMCMC will in the limit* $R \to \infty$ *be independent draws from* π*. Note similarly that it follows that* (6.29) *is a consistent estimator (as* $R \to \infty$*).*

Note that adding a backward or ancestor simulation step can drastically increase mixing when sampling the conditional trajectories $\mathbf{x}'_j[r]$ [Lindsten and Schön, 2013]. In the iPMCMC sampler, we can replace simulating from the final weights on line 7 by a backward simulation step. Another option for the CSMC nodes is to replace this step by internal ancestor sampling [Lindsten et al., 2014] steps and simulate from the final weights as normal.

6.3.3 Using All the Particles

As we discussed for the PIMH and PG cases in Section 6.2.5, it can be wasteful to throw away most of our generated samples. Thankfully we can again carry out a Rao-Blackwellization for iPMCMC to make use of all particles when estimating expectations of interest. To do this we average over the possible new values for the conditional node index c_j and corresponding particle index b_j at each Gibbs update j. We can do this by replacing $\frac{1}{P}\sum_{j=1}^{P} f(\mathbf{x}'_j[r])$ in (6.29) by

$$\frac{1}{P}\sum_{j=1}^{P} \mathbb{E}\left[f(\mathbf{x}'_j[r])\middle|\xi_{1:M}, c_{1:P\setminus j}\right] = \frac{1}{P}\sum_{m=1}^{M}\left[\left(\sum_{j=1}^{P}\hat{\zeta}_m^j\right)\cdot\left(\sum_{i=1}^{N}\bar{w}_{T,m}^i f(\mathbf{x}_m^i)\right)\right]$$

where the expectation is over b_j and c_j. We highlight that each $\hat{\zeta}_m^j$, as defined in (6.27), depends on which indices are sampled earlier in the index reassignment loop.

To derive this we first note that for iteration r we calculate $\frac{1}{P}\sum_{j=1}^{P} f(\mathbf{x}'_j[r]) = \frac{1}{P}\sum_{j=1}^{P} f(\mathbf{x}_{c_j}^{b_j})$ where we can Rao-Blackwellize the selection of the retained particle along with each individual Gibbs update and thereby replace $f(\mathbf{x}_{c_j}^{b_j})$ with its expectation over c_j and b_j as follows

$$\begin{aligned}\frac{1}{P}\sum_{j=1}^{P} \mathbb{E}\left[f(\mathbf{x}_{c_j}^{b_j})\middle|\xi_{1:M}, c_{1:P\setminus j}\right] &= \frac{1}{P}\sum_{j=1}^{P} \mathbb{E}\left[\sum_{i=1}^{N}\bar{w}_{T,c_j}^i f(\mathbf{x}_{c_j}^i)\middle|\xi_{1:M}, c_{1:P\setminus j}\right] \\ &= \frac{1}{P}\sum_{j=1}^{P}\sum_{i=1}^{N}\sum_{m=1}^{M}\hat{\zeta}_m^j\bar{w}_{T,m}^i f(\mathbf{x}_m^i) = \frac{1}{P}\sum_{m=1}^{M}\left[\left(\sum_{j=1}^{P}\hat{\zeta}_m^j\right)\cdot\left(\sum_{i=1}^{N}\bar{w}_{T,m}^i f(\mathbf{x}_m^i)\right)\right].\end{aligned}$$

Here we have made use of the knowledge that the internal particle system $\{(\mathbf{x}_m^i, \bar{w}_{T,m}^i)\}$ does not change between Gibbs updates of the c_j's, whereas the $\hat{\zeta}_m^j$ do. We emphasize that this is a separate Rao-Blackwellization of each Gibbs update of the conditional node indices, such that each is conditioned upon the actual update made at $j-1$, rather than a simultaneous Rao-Blackwellization of the full batch of P updates. Though the latter also has analytic form and should theoretically be lower variance, it suffers from inherent numerical instability and so is difficult to calculate in practice. We found that empirically there was not a noticeable difference between the performance of the two procedures. Furthermore, one can always run additional Gibbs updates on the c_j's and obtain an improved estimate on the relative sample weightings if desired.

6.3.4 Choosing P

Before jumping into the full details of our experimentation, we quickly consider the choice of P. Intuitively we can think of the independent SMC's as particularly useful if they are selected as the next conditional node. The probability of the event that at least one conditional node switches with an unconditional, is given by

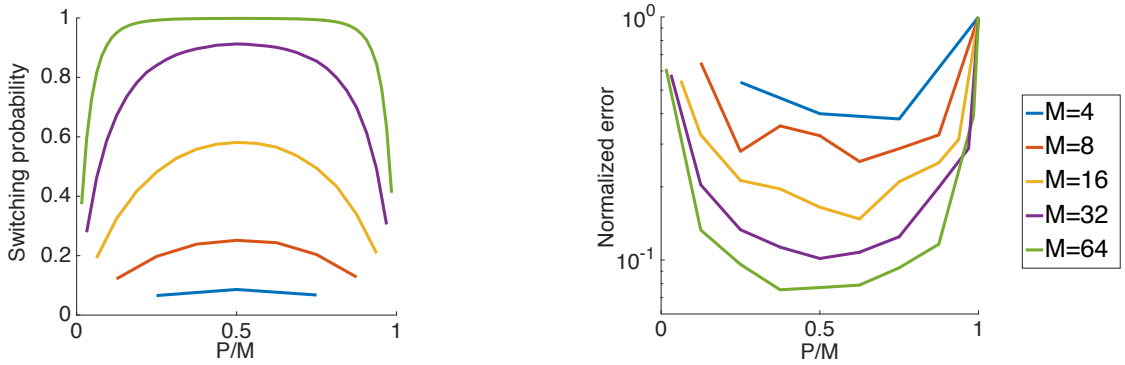
$$\mathbb{P}(\{\text{switch}\}) = 1 - \mathbb{E}\Big[\prod_{j=1}^{P} \frac{\hat{Z}_{c_j}}{\hat{Z}_{c_j} + \sum_{m \notin c_{1:P}}^{M} \hat{Z}_m}\Big]. \tag{6.36}$$

There exist theoretical and experimental results [Pitt et al., 2012; Bérard et al., 2014; Doucet et al., 2015] that show that the distributions of the normalization constants are well-approximated by their log-Normal limiting distributions. With σ^2 ($\propto \frac{1}{N}$) being the variance of the (C)SMC estimate, we have $\log\left(Z^{-1}\hat{Z}_{c_j}\right) \sim \mathbb{N}(\frac{\sigma^2}{2}, \sigma^2)$ and $\log\left(Z^{-1}\hat{Z}_m\right) \sim \mathbb{N}(-\frac{\sigma^2}{2}, \sigma^2)$, $m \notin c_{1:P}$ at stationarity, where Z is the true normalization constant. Under this assumption, we can accurately estimate the probability (6.36) for different choices of P an example of which is shown in Figure 6.4a along with additional analysis given in the supplementary material of Rainforth et al. [2016c]. These provide strong empirical evidence that the switching probability is maximised for $P = M/2$.

In practice we also see that best results are achieved when P makes up roughly half of the nodes, see Figure 6.4b for performance on the state space model introduced in (6.37). Note also that the accuracy seems to be fairly robust with respect to the choice of P. Based on these results, we set the value of $P = M/2$ for the rest of our experiments.

6.3.5 Experiments

To demonstrate the empirical performance of iPMCMC we report experiments on two state-space models. Although both the models considered are Markovian, we emphasise that iPMCMC goes far beyond this and can be applied to arbitrary graphical models and even probabilistic



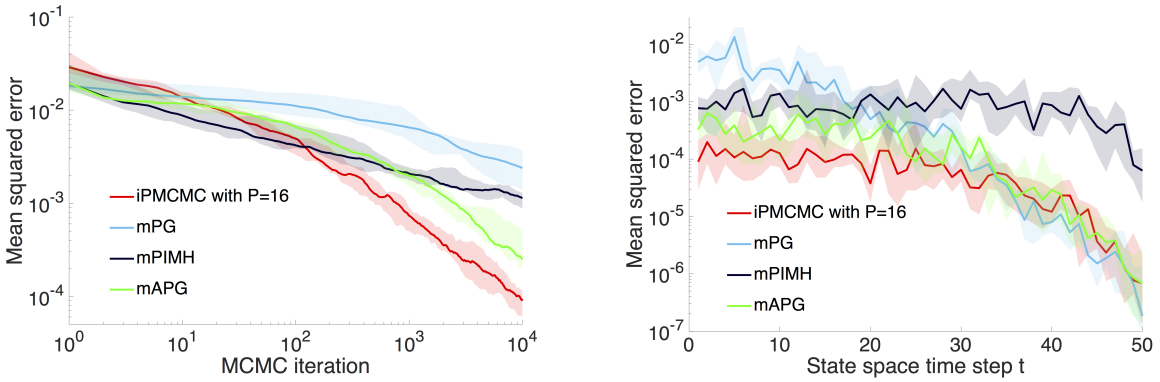
(a) Limiting log-Normal **(b)** Gaussian state-space model

Figure 6.4: a) Estimation of switching probability for different choices of P and M assuming the log-Normal limiting distribution for $\hat{Z}_m$ with $\sigma = 3$. b) Median error in mean estimate for different choices of P and M over 10 different synthetic datasets of the linear Gaussian state space model given in (6.37) after 1000 MCMC iterations. Here errors are normalized by the error of a multi-start PG sampler which is a special case of iPMCMC for which $P = M$ (see Section 6.3.5).

programs as explained in Section 7.4.2.3. We will focus our comparison on the trivially distributed alternatives, whereby M independent PMCMC samplers are run in parallel–these are PG, particle independent Metropolis-Hastings (PIMH) [Andrieu et al., 2010] and the alternate move PG sampler (APG) [Holenstein, 2009]. Comparisons to other alternatives, including independent SMC, serialized implementations of PG and PIMH, and running a mixture of independent PG and PIMH samplers, are provided in the supplementary material of Rainforth et al. [2016c]. None outperformed the methods considered here, with the exception of running a serialized PG implementation with an increased number of particles, requiring significant additional memory ($O(MN)$ as opposed to $O(M + N)$).

APG interleaves PG steps with PIMH steps in an attempt to overcome the issues caused by path degeneracy in PG. We refer to the trivially distributed versions of these algorithms as multi-start PG, PIMH and APG respectively (mPG, mPIMH and mAPG). We use Rao-Blackwellization, as described in 6.3.3, to average over all the generated particles for all methods, weighting the independent Markov chains equally for mPG, mPIMH and mAPG. We note that mPG is a special case of iPMCMC for which $P = M$. Because our interest is in alleviating degeneracy, we used the bootstrap proposal and multinomial resampling in our experiments.[10] $M = 32$ nodes and $N = 100$ particles were used unless otherwise stated. Initialization of the retained particles for iPMCMC and mPG was done by using standard SMC sweeps.

[10]However, we recommend using systematic resampling, as is the default in the provided implementations.



(a) Convergence in mean for full sequence

(b) Final error in mean for latent marginals

Figure 6.5: Mean squared error averaged over all dimensions and steps in the state sequence as a function of MCMC iterations (left) and mean squared error after 10^4 iterations averaged over dimensions as a function of position in the state sequence (right) for (6.37) with 50 time sequences. The solid line shows the median error across the 10 tested synthetic datasets, while the shading shows the upper and lower quartiles. Ground truth was calculated using the Rauch–Tung–Striebel smoother Rauch et al. [1965].

6.3.5.1 Linear Gaussian State Space Model

We first consider a linear Gaussian state-space model (LGSSM) with 3-dimensional latent states $x_{1:T}$, 20-dimensional observations $y_{1:T}$ and dynamics given by

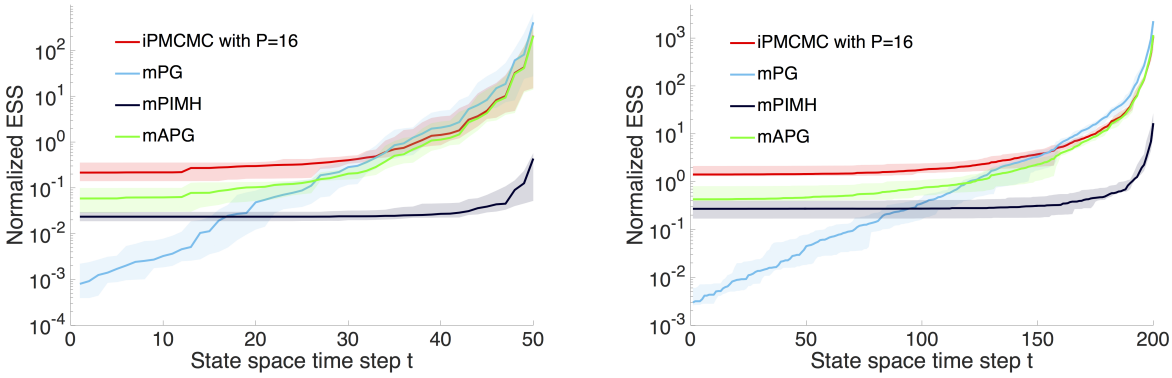
$$x_1 \sim \mathcal{N}(\mu, V) \tag{6.37a}$$

$$x_t = \alpha x_{t-1} + \delta_{t-1} \qquad \delta_{t-1} \sim \mathcal{N}(0, \Omega) \tag{6.37b}$$

$$y_t = \beta x_t + \varepsilon_t \qquad \varepsilon_t \sim \mathcal{N}(0, \Sigma). \tag{6.37c}$$

We set $\mu = [0, 1, 1]^T$, $V = 0.1\ \mathbf{I}$, $\Omega = \mathbf{I}$ and $\Sigma = 0.1\ \mathbf{I}$ where $\mathbf{I}$ represents the identity matrix. The constant transition matrix, α, corresponds to successively applying rotations of $\frac{7\pi}{10}$, $\frac{3\pi}{10}$ and $\frac{\pi}{20}$ about the first, second and third dimensions of x_{t-1} respectively followed by a scaling of 0.99 to ensure that the dynamics remain stable. A total of 10 different synthetic datasets of length $T = 50$ were generated by simulating from (6.37a)–(6.37c), each with a different emission matrix β generated by sampling each column independently from a symmetric Dirichlet distribution with concentration parameter 0.2.

Figure 6.5a shows convergence (in MCMC iterations) in the estimate of the latent variable means to the ground-truth solution for iPMCMC and the benchmark algorithms. It shows that iPMCMC comfortably outperforms the alternatives from around 200 iterations onwards, with only iPMCMC and mAPG demonstrating behavior consistent with the Monte Carlo convergence rate, suggesting that mPG and mPIMH are still far from the ergodic regime. Figure 6.5b shows the same errors after 10^4 MCMC iterations as a function of position in state sequence. It demonstrates that iPMCMC outperformed all the other algorithms for the early stages of the state sequence, for



(a) LGSSM **(b)** NLSSM

Figure 6.6: Normalized effective sample size (NESS) for LGSSM (left) and NLSSM (right).

which mPG performed particularly poorly. Toward the end of state sequence, iPMCMC, mPG and mAPG all gave similar performance, whilst that of mPIMH was significantly worse.

6.3.5.2 Nonlinear State Space Model

We next consider the one-dimensional nonlinear state-space model (NLSSM) considered by, among others, Gordon et al. [1993]; Andrieu et al. [2010]

$$x_1 \sim \mathcal{N}\left(\mu, v^2\right) \tag{6.38a}$$

$$x_t = \frac{x_{t-1}}{2} + 25\frac{x_{t-1}}{1+x_{t-1}^2} + 8\cos\left(1.2t\right) + \delta_{t-1} \tag{6.38b}$$

$$y_t = \frac{{x_t}^2}{20} + \varepsilon_t \tag{6.38c}$$

where $\delta_{t-1} \sim \mathcal{N}\left(0, \omega^2\right)$ and $\varepsilon_t \sim \mathcal{N}\left(0, \sigma^2\right)$. We set the parameters as $\mu = 0$, $v = \sqrt{5}$, $\omega = \sqrt{10}$ and $\sigma = \sqrt{10}$. Unlike the LGSSM, this model does not have an analytic solution and therefore one must resort to approximate inference methods. Further, the multi-modal nature of the latent space makes full posterior inference over $x_{1:T}$ challenging for long state sequences.

To examine the relative mixing of iPMCMC we calculate an effective sample size (ESS) for different steps in the state sequence as described in Section 5.3.2.3, taking care to condensed identical samples. To be precise, let

$$u_t^k \in \{x_{t,m}^i[r]\}_{m=1:M}^{i=1:N,r=1:R}, \quad \forall k \in 1\ldots K,\ t \in 1\ldots T$$

denote the unique samples of x_t generated by all the nodes and sweeps of particular algorithm after R iterations, where K is the total number of unique samples generated. The weight assigned to these unique samples, v_t^k, is given by the combined weights of all particles for

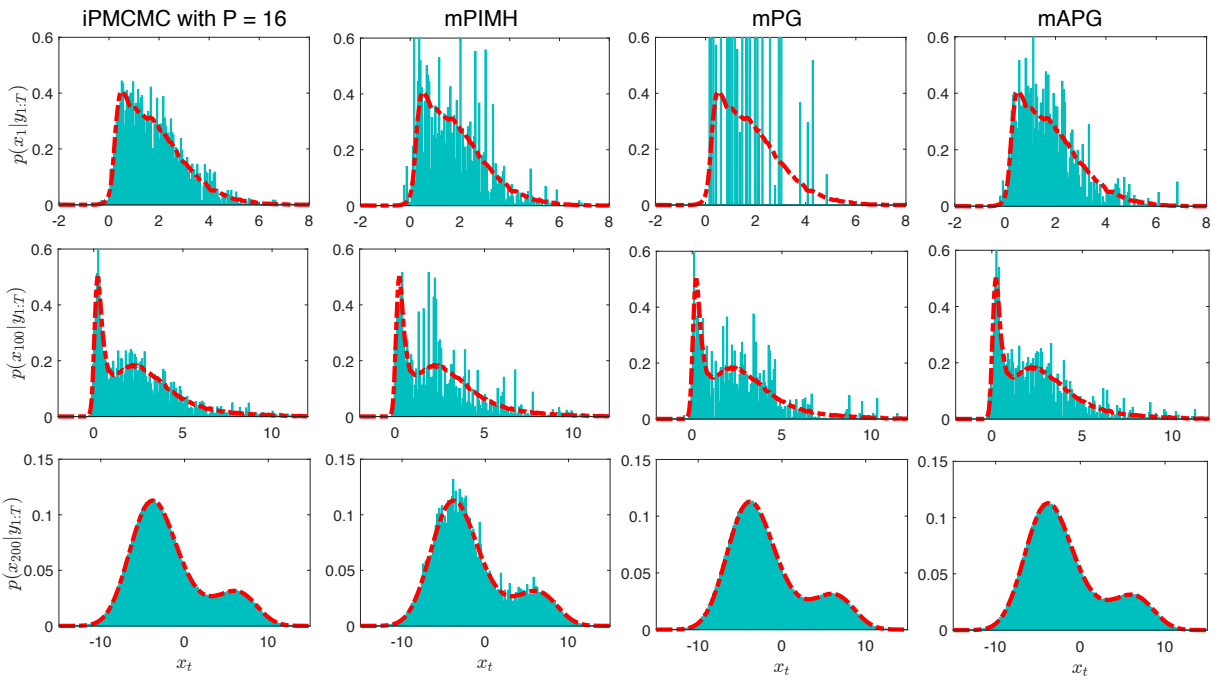


Figure 6.7: Histograms of generated samples at $t = 1, 100,$ and 200 for a single dataset generated from (6.38) with $T = 200$. Dashed red line shows an approximate estimate of the ground truth, found by running a kernel density estimator on the combined samples from a small number of independent SMC sweeps, each with 10^7 particles.

which x_t takes the value u_t^k:

$$v_t^k = \sum_{r=1}^{R} \sum_{m=1}^{M} \sum_{i=1}^{N} \bar{w}_{t,m}^{i,r} \eta_m^r \delta_{x_{t,m}^i[r]}(u_t^k) \tag{6.39}$$

where $\delta_{x_{t,m}^i[r]}(u_t^k)$ is the Kronecker delta function and η_m^r is a node weight. For iPMCMC the node weight is given by as per the Rao-Blackwellized estimator described in Section 6.3.3. For mPG and mPIMH, η_m^r is simply $\frac{1}{RM}$, as samples from the different nodes are weighted equally in the absence of interaction. Finally we define our effective sample size as $\text{ESS}_t = \left(\sum_{k=1}^{K} \left(v_t^k\right)^2\right)^{-1}$.

Figure 6.6 shows the ESS for the LGSSM and NLSSM as a function of position in the state sequence. For this, we omit the samples generated by the initialization step as this SMC sweep is common to all the tested algorithms. We further normalize by the number of MCMC iterations so as to give an idea of the rate at which unique samples are generated. These show that for both models the ESS of iPMCMC, mPG and mAPG is similar towards the end of the space sequence, but that iPMCMC outperforms all the other methods at the early stages. The ESS of mPG was particularly poor at early iterations. PIMH performed poorly throughout, reflecting the very low observed acceptance ratio of around 7.3% on average.

It should be noted that the ESS is not a direct measure of performance for these models. For example, the equal weighting of nodes is likely to make the ESS artificially high for mPG, mPIMH

and mAPG, when compared with methods such as iPMCMC that assign a weighting to the nodes at each iteration. To acknowledge this, we also plot histograms for the marginal distributions of a number of different position in the state sequence as shown in Figure 6.7. These confirm that iPMCMC and mPG have similar performance at the latter state sequence steps, whilst iPMCMC is superior at the earlier stages, with mPG producing almost no more new samples than those from the initialization sweep due to the degeneracy. The performance of PIMH was consistently worse than iPMCMC throughout the state sequence, with even the final step exhibiting noticeable noise.

6.3.6 Discussion

The iPMCMC sampler overcomes degeneracy issues in PG by allowing the newly sampled particles from SMC nodes to replace the retained particles in CSMC nodes. Our experimental results demonstrate that, for the models considered, this switching in rate is far higher than the rate at which PG generates fully independent samples. Moreover, the results in Figure 6.4b suggest that the degree of improvement over an mPG sampler with the same total number of nodes increases with the total number of nodes in the pool. The mAPG sampler performs an accept-reject step that compares the marginal likelihood estimate of a single CSMC sweep to that of a single SMC sweep. In the iPMCMC sampler, the CSMC estimate of the marginal likelihood is compared to a population sample of SMC estimates, resulting in a higher probability that at least one of the SMC nodes will become a CSMC node. Since the original PMCMC paper [Andrieu and Roberts, 2009] there have been several papers studying and improving upon the basic PG algorithm [Chopin and Singh, 2015; Lindsten et al., 2015; Whiteley et al., 2010; Lindsten and Schön, 2013; Lindsten et al., 2014], many of which also be used to improve the iPMCMC method even further.

7

General Purpose Inference for Probabilistic Programs

In Chapter 4 we showed how probabilistic programming systems (PPSs) provide an expressive framework for specifying probabilistic models. We now consider the other major component for PPSs: automating inference for any model the user is allowed to specify using general-purpose inference engines. For most PPSs this requires two things – the inference engine itself and either an interpreter controlling the probabilistic semantics or a compiler to convert the *query* to a suitable form for input to the inference engine. Our focus will be on the compiled case.

For the inference driven PPSs discussed in Section 4.2.1, the inference engine typically comprises of a standard Bayesian inference method for graphical models such as those discussed in Chapters 3 and 6. Developing these in a way to robustly work for a wide range of problems typically requires careful engineering and algorithmic innovation – e.g. because many inference methods require the definition of a proposal, upon which performance can critically depend – but does not generally require the development of approaches distinct to those used outside a probabilistic programming context. In these systems, the inference algorithm(s) is usually chosen first, with the language and its restrictions built around it. Therefore the challenges of designing the system are generally rooted in generalizing and increasing the robustness of the specific inference method(s) used. Similarly, the language itself and associated compiler is generally built around providing the easiest representation to work with the target model class, while the fact that most of these such systems do not support higher order functions usually substantially simplifies the compilation process.

Because the design of these systems is very much driven by the particular inference algorithm used, it is beyond the scope of this thesis to do the associated literature justice. Our focus will instead mostly be on conducting inference for universal PPSs, though some of what we introduce will apply to both. Unfortunately, we will find that there are very few (known) inference methods which can actually cope with the most general possible models as we introduced in Section 4.3.2, all of which suffer particularly badly from the curse of dimensionality. Consequently, it will be necessary to make certain (mostly very small) concessions in generality to achieve any reasonable

performance on non-toy models. We will focus on conducting inference in Anglican [Wood et al., 2014; Tolpin et al., 2016] to give us a basis for explanation, but much of what we discuss will still be relevant to other universal systems, in particular, those which are also built around **`sample`**-**`observe`** syntaxes, such as VentureScript [Mansinghka et al., 2014], WebPPL [Goodman and Stuhlmüller, 2014], and Probabilistic C [Paige and Wood, 2014].

7.1 A High-Level Introduction to General Purpose Inference

Before getting into the nitty-gritty of designing a compiler and inference engine for a universal PPL, we first consider at a high-level how we might hope to do inference in such systems. The simplest thing we could do is self-normalized importance sampling (SNIS) using the generative model specified by the **`sample`** statements as a proposal. This strategy is sometimes known as *likelihood weighting* in the probabilistic programming literature and simply involves directly sampling from the forward model and accumulating weights from the **`observe`** conditioning statements in a sequential importance sampling fashion (see Section 6.1.2). In Section 4.3.2, we specified with (4.3) the unnormalized conditional distribution of the program using execution trace. If we sample from the generative model then it is clear that all samples will constitute valid traces, i.e. they will satisfy $\mathcal{B}(x_{1:n_x}, \lambda) = 1$, while the support of our proposal will clearly cover that of the posterior provided the conditioning density terms are always finite. This is also a proposal we can always sample from because our **`observe`** terms will only affect the probability of a trace, not the variables it generates. Using the notation from Section 4.3.2 this approach leads to the following SNIS characterization of the posterior

$$p(x_{1:n_x}|\lambda) \approx \hat{p}(x_{1:n_x}|\lambda) := \sum_{n=1}^{N} \bar{w}_n \delta_{\hat{x}^n_{1:n_x}}(x_{1:n_x}) \quad \text{where} \tag{7.1}$$

$$\hat{x}^n_{1:n_x} \sim \prod_{j=1}^{n_x} f_{a_j}(x_j|\eta_j); \quad \bar{w}_n = \frac{w_n}{\sum_{n=1}^{N} w_n}; \quad w_n = \prod_{k=1}^{\hat{n}^n_y} g_{\hat{b}^n_k}(\hat{y}^n_k|\hat{\psi}^n_k); \tag{7.2}$$

$\delta_{\hat{x}^n_{1:n_x}}(\cdot)$ is a delta function centered at $\hat{x}^n_{1:n_x}$; η_j and a_j are deterministic functions of $\hat{x}^n_{1:n_x}$ and λ; and $\hat{n}^n_y$, $\hat{b}^n_k$, $\hat{y}^n_k$, and $\hat{\psi}^n_k$ are random variables that are deterministic functions of $\hat{x}^n_{1:n_x}$ and λ. Here we are sampling $\hat{x}^n_{1:n_x}$ from the generative model defined by the program, which is equivalent to a running the forward program ignoring all of the **`observe`** statements. Our unnormalized likelihood weights w_n correspond to the product of all the probabilities accumulated from the **`observe`** terms. We remind the reader that each x_j corresponds to the direct output of a sample statement, not an explicit variable in the program, with the program variables and outputs being deterministic functions of $x_{1:n_x}$ (see Section 4.4.2). In addition to providing a

characterization of the conditional distribution through $\hat{p}(x_{1:n_x}|\lambda)$, we can also use our samples to construct consistent estimates from the outputs of program Ω, remembering that these are a deterministic function of $x_{1:n_x}$ and so an expectation over Ω can be represented as an expectation over $x_{1:n_x}$. This convergence follows in the standard way for SNIS provided our program terminates with probability 1.

In practice, our importance sampling inference engine will not sample a full $\hat{x}^n_{1:n_x}$ before evaluating its weight in one go – it will use a sequential importance sampling strategy whereby we sample directly from any **`sample`** statement as we encounter it and accumulate weights from **`observe`** statements as we go (see Figure 4.3 in Section 4.3.2). Another way of viewing this is that we will repeatedly run our query as if it were an ordinary program, except for the fact that we accumulate a weight as a side effect that is then returned with the produced sample. Such a guess-and-check strategy will obviously be ineffective except in very low dimensions. However, it will both be a go-to strategy for the most extreme possible problems that fail to satisfy any of the required assumptions for more advanced methods and a basis for more complex inference strategies. For example, as we will show in Section 7.4, we can convert this to an SMC strategy, provided n_y is fixed, by running a number of separate executions in parallel and performing resampling at each of the **`observe`** statements. Alternatively, we can develop component-wise MCMC strategies by proposing changes to individual x_j and then re-evaluating the weights. However, doing this will require our inference strategy to have more control than simply forward sampling the program in its entirety. More specifically, we will need both a compiler (or interpreter) and an interface for inference, which will be the focus of the next two sections.

7.2 Compiling Queries

Non-probabilistic languages can be either compiled, whereby high-level source code is converted to a lower level language (e.g. byte-code) before being evaluated, or interpreted, whereby an interpreter reads the program and directly evaluates it based on the language semantics. The same is true for PPLs, for which compilation usually comprises of converting the query to a model artifact in a host language in which the inference algorithm itself is written. Though its original implementation was interpreted, the current version of Anglican is compiled and our focus will be on this compiled approach. Rather than just being a technical hurdle, compilation is also often an important tool in the ability of probabilistic programming systems to perform effective inference as, in addition to offering potential speed improvements, it can often be used to establish helpful salient features in the model, such as dependency structures, variable types, features of the

model that can guide which inference algorithm is likely to be most successful, or even elements of the model that can be calculated analytically. Hakaru [Narayanan et al., 2016; Zinkov and Shan, 2016] is a good example of what can be achieved in probabilistic programming through compilation alone, as it, for example, uses the computer algebra system Maple to automatically simplify models when possible. Once a compiler is written for a PPL, the abstraction it provides often substantially simplifies the process of writing new inference algorithms by providing a common representation of models and allowing inheritance from existing algorithms.

7.2.1 What Do We Want to Achieve Through Compilation?

So what are the aims of our PPL compiler? First and foremost, we need to perform a source-to-source transformation of the program to produce some representation in the conventional programming language in which the inference engine is written, such that execution of the program is possible and inference can be carried out. As a simple example, consider the case of an Anglican program where we wish only to run importance sampling using the generative model as a proposal (i.e. the query with all the `observe` statements removed). Here we need only produce a Clojure program that samples directly from the generative model and accumulates weights from the `observe` terms as a side-effect. We can then rerun this program arbitrarily many times, each returning a sampled output and accompanying weight, to produce a sequence of importance samples which can then be used to make consistent Monte Carlo estimations. Such a strategy would be naïve though as we would be able to use our Clojure program for little else than importance sampling with this particular proposal. Clearly, we want to compile to a more general purpose representation that permits a wider array of inference algorithms and proposals, and which ideally allows features of the model, such as its dependency structure, to be exploited.

One desirable feature of our compiled representation is an ability to make partial program evaluations. This will allow us to re-evaluate elements of the program in isolation in a Metropolis within Gibbs fashion and it will allow us to interrupt the execution of the program, giving us the ability to add in things such as the resampling step in SMC. Supporting partial evaluation can also allow us to avoid gratuitous re-executions of elements of the program whose result is already known, by using databases to store and recall the effect of previous executions.

Another desirable feature is to avoid being restricted to only directly sampling from `sample` statements. We may wish to sample from a different distribution as a proposal or to evaluate the probability density of the `sample` statement producing a particular output, e.g. as part of an acceptance ratio calculation for an MCMC scheme. The key requirement for both of

these is the knowledge of density function associated with the `sample` statement, rather than just access to a black-box sampling scheme. This is the motivation behind why we defined the syntax of `sample` as taking as input a distribution object which encodes both the density function and means of sampling from that density.

It will also often be helpful for our compiler to delineate between probabilistic and deterministic elements of the code. Other than potentially trying to make efficiency gains by avoiding repeated computation, we know that any deterministic parts of our code, namely code in between consecutive `sample` and or `observe` statements, can be safely run without the need to worry about the implications this has on the inference (remembering that we have presumed that our program has no side effects of than those caused by `sample` and `observe`). The execution of these segments of the program in isolation can thus be as per the semantics of the language being compiled to, without needing to worry about the probabilistic implication. On the other hand, we will generally desire the identification of *checkpoints* for positions in the program where `sample` or `observe` statements are made so that the behavior of the program at these points can be handed over to the inference engine.

This example list of desirable features for our compiler is far from exhaustive. In particular, there will be many inference specific features required for some systems, such as the need to have a representation of the derivations, to carry out Hamiltonian Monte Carlo inference [Carpenter et al., 2015] or common variational inference methods [Kucukelbir et al., 2015]. This is often done using automatic differentiation [Baydin et al., 2015]. As we said earlier, we will often also want our compilation to, when possible, pick out salient features of the model, carry out simplifications, and even automatically establish the most suitable inference algorithm for a particular problem. The compiler (or equivalently interpreter) is an integral part of any PPL and there is no one best approach for all situations. One of the key distinguishing features between different PPLs is how they approach this compilation problem, with different design choices inevitably leading to systems geared towards different models or inference algorithms.

7.2.2 Compilation of Anglican Queries

As it is infeasible to detail the inner workings of a PPL compiler in a general manner, we now provide a more in-depth introduction to the compilation employed by Anglican. Our introduction is inevitably not exhaustive, focusing more on intuition than being exactly true to the implementation details. We refer the reader to [Tolpin et al., 2016] for a more complete

introduction, emphasizing that what follows is an exposition of their excellent work, rather than a novel contribution of our own.

As we explained in Section 4.4.2, Anglican programs, or queries, are compiled using the macro `query` which provides a Clojure function that can be passed to one of the provided inference algorithms. The key element of this compilation for providing the desirable properties discussed in the last section is that Anglican compiles queries to *continuation passing style* (CPS) [Appel and Jim, 1989] Clojure functions.[1] At a high level, a continuation is a function that represents the rest of the program. CPS is a style of functional programming that uses a series of continuations to represent the program through a series of nested function calls, where the program is run by evaluating each function and then passing the output to the continuation which invokes the rest of the program. This is perhaps easiest to see through example. Consider the simple function `+`, which in Clojure has syntax `(+ a b)`. The CPS-transformed version of `+`, which we will call `+&` takes an extra input of the continuation $\mathcal{P}$ and invokes it after evaluation such that we have `(defn +& [a b` $\mathcal{P}$`] (`$\mathcal{P}$ `(+ a b)))`. More generally, for any anonymous function `f`, we have that its CPS transformation is `(defn f& [args` $\mathcal{P}$`] (`$\mathcal{P}$ `(f args)))`. We will use this notation of adding an `&` to an expression name to denote its CPS transformation throughout. To give a more detailed example, the CPS transform of the program `(max 6 (* 4 (+ 2 3)))` would be

```
(fn [P] (+& 2 3 (fn [x] (*& 4 x (fn [y] (max& 6 y P)))))
```

where `*&` and `max&` are analogous to `+&` defined as before. Here we respectively have that `(fn [x] (*& 4 x (fn [y] (max& 6 y` $\mathcal{P}$`))))` and `(fn [y] (max& 6 y` $\mathcal{P}$`))` are our first and second continuations. Note that the CPS transformed code is itself a function because it takes a continuation.

Things are a little trickier for general expression that are neither literals nor simple first order functions, for example, binding forms like `let`, Anglican special forms like `sample`, and branching statements like `if`. For these expressions, the high-level idea is the same, but the CPS transformation is, unfortunately, expression specific and must, in general, be implemented on a case-by-case basis. For example, one can CPS transform `let` by going from `(let [x (foo1 4) y (foo2 8)] (foo3 x y))` to

```
(fn [P] (foo1& 4 (fn [x] (foo2& 8 (fn [y] (foo3& x y P))))))
```

[1] Note WebPPL also does a CPS style transformation, namely to a purely functional subset of Javascript.

noting that within the `(fn [x] .)` closure, `x` is bound to the input of the function, giving behavior synonymous to the original `let` block. Another special case of particular note is `loop-recur` blocks. These can be CPS transformed by explicitly redefining them as a self-recursive function which can then be transformed in the normal way. For example,

```
(loop [x 10] (if (> x 1) (recur (- x 2)) x))
```

becomes

```
(fn [P] ((fn foo [x] (if (> x 1) (foo (- x 2)) (P x))) 10))
```

where we have exploited the fact that `fn` allows the function to be named (in this case to `foo`) so that it can call itself.

In CPS-style code, functions never return (until the final tail call) and every function takes an extra input corresponding to the continuation. This would be a somewhat awkward method for writing programs directly, as the whole program must be written as a single nested function. However, it can be a very useful form to compile to as the execution of the program becomes exceptionally simple and just involves evaluating functions and passing the output to the next continuation – it explicitly linearizes the computation. For our purposes, having access to functions representing the rest of the program in the form of continuations will be particularly useful as it will allow for partial program evaluations. It will also be convenient for adding checkpoints at particular points in the program – namely at the `sample` and `observe` calls – where control is handed over to the inference algorithm.

Compilation of an Anglican query, triggered through the `query` macro, is done recursively in a top-down manner. The key function in doing this, called `cps-of-expression`, dispatches to the individual CPS transformations by matching types of expression, or, if necessary, directly by name. Individual CPS transformations then recursively call `cps-of-expression` until the full query is transformed. These individual Anglican CPS transformations are marginally more complicated than the framework we have laid out thus far. This is because it is necessary to not just run the program, but also track its state from a probabilistic perspective, storing things such as sample weights and other probabilistic side-effects. To deal with this, the transformed Anglican continuations take as input an *internal state*, which we will denote as $\mathcal{S}$, in addition to the computed value. Thus, for example, the transformation for a generic function `f` becomes

```
(defn f& [args P S] (P (f args) S)).
```

More generally, we will simply pass on $\mathcal{S}$ unchanged except for the Anglican special forms which can use and manipulate the internal state. $\mathcal{S}$ itself is defined to be a hash map, initialized as follows

```
(def initial-state {:log-weight 0.0 :result nil ...})
```

where ... includes some fields we will not directly consider at the moment, some of which are algorithm specific. Here the field `:log-weight` allows the program to be assigned a weight as accumulated by, for example, `observe` statements or when sampling from a different distribution than that provided to the `sample` statement. The return for each sample in our inference will effectively be the $\mathcal{S}$, hence the inclusion of the `:result` field which is set to the output of the query at its tail call.

One of the key components of the Anglican compilation is the transformation applied to the special forms and in particular the probabilistic forms `sample` and `observe`. These are respectively transformed to the Clojure record constructors `->sample` and `->observe` which have the call syntaxes of `(->sample id dist` $\mathcal{P}$ $\mathcal{S}$`)` and `(->observe id dist value` $\mathcal{P}$ $\mathcal{S}$`)` respectively, where `id` is a unique checkpoint identifier (e.g. the $\{1, \ldots, n_s\}$ and $\{1, \ldots, n_o\}$ identifiers we used in 4.3.2) set at compilation time, and `dist` and `value` are the original inputs to the `sample` and `observe` statements. The final return of the program is similarly transformed to the record constructor `(->result` $\mathcal{S}$`)`. At a high level, one can think of a Clojure record as defining a new class type (here `trap.sample`, `trap.observe`, and `trap.result` respectively) with given fields (here `:id`, `:dist` etc). Our constructors thus create an object of the given type with appropriately set fields. The significance of this is that it allows definition of a multimethod, which we call `checkpoint`, to provide a runtime polymorphism (exposing a single interface) for dispatching depending on the checkpoint type and the inference algorithm. In other words, we will have one function, `checkpoint`, whose behavior can be redefined for different checkpoint types and inference algorithms. There are many consequences of this. Firstly, our compiler does not need to be inference algorithm specific because we can use `checkpoint` to distribute the behavior to the required inference algorithm at run time. Secondly, it creates an abstraction barrier for writing inference algorithms – implementing an inference algorithm now only requires us to implement, along with a top-level function `infer` that we will discuss later, new methods describing the behavior of `checkpoint` at `sample`, `observe`, and `result` checkpoints. Furthermore, these methods can be inherited from other inference algorithms, for example, the `observe` checkpoints for particle Gibbs are inherited from SMC in Anglican's implementation.

A slightly more subtle consequence of compiling the Anglican special forms to a constructor with explicit output type is that we can use this to catch the checkpoints themselves. Once compiled, individual instances of a program in the Anglican inference engines are run using a function called `exec`. The role of `exec` is to run the program until it reaches a checkpoint that

requires control to be transferred to the top-level inference function of a particular method, i.e. the **infer** function, such as when a particular trace has finished running or when interaction is required between different samples, e.g. the resampling step in SMC. In Anglican this is achieved using *trampolining*. In functional programming languages, trampolining is a process of looping execution where if an iteration of a loop returns a function, that function is immediately evaluated without passing forward any arguments, with this process continuing until a non-functional output is returned. The primary use of trampolining in functional languages is generally for managing stack sizes by constructing a nested call structure of *thunks* (i.e. functions which require no input arguments) that when called by the trampolining function (called **trampoline** in Clojure) invokes the full set of nested calls without requiring a stack to be constructed or stored, as would be necessary for an ordinary nested function structure. In the context of our CPS transformations, we can do this by simply wrapping each call with an anonymous function, namely converting `(foo args` $\mathcal{P}$ $\mathcal{S}$`)` to `(fn [] (foo args` $\mathcal{P}$ $\mathcal{S}$`))` to delay execution until trampolined, which will be carried out for all continuations except the checkpoints. Though a large part of the motivation for doing this in Anglican is similarly to maintain the stack size, it is also highly useful for using the checkpoints to trigger control to be transferred to the **infer** function. To be precise, the **exec** function is defined as

```
(defn exec [algorithm prog value S]
  (loop [step (trampoline prog value S)]
    (let [next (checkpoint algorithm step)]
     (if (fn? next) (recur (trampoline next)) next))))
```

where `algorithm` specifies the inference algorithm; `prog` is the program to call (which is always either a full program or a continuation); `value` is the input required by `prog`, comprising of either the original program inputs or the value passed to the continuation; and $\mathcal{S}$ is the program state as before. Here we see that **exec** first creates the variable `step` by making a trampolined call to `prog`. As all continuations are now represented by anonymous functions with no inputs except for our checkpoints, this will run until it hits one of those checkpoints, returning the corresponding constructed checkpoint object. In other words, this causes the program to run until it reaches a **sample** or **observe** statements, or it reaches the end of the program. This is a highly desirable behavior, as it means the deterministic elements of our program can be run as normal using a single function call. Once returned, our multimethod **checkpoint** is called using `algorithm` and `step`, which distributes to the appropriate **checkpoint** implementation based on the value of `algorithm` and the type of `step` (e.g. calling to the **sample** checkpoint method of an algorithm if it is of type `trap.sample`). Invoking the appropriate checkpoint

```
(query [y]
 (let [s (sample (gamma 1 2))]
  (observe (normal 0 s) y)
  s))
```

```
(fn [y S]
 (->sample 'S30744
  (gamma 1 2)
  (fn var30743 [s S]
   (->observe 'O30742
     (normal 0 s)
     y
     (fn do30741 [_ S]
      (->result s S))
     S))
  S))
```

Figure 7.1: Example compilation of an Anglican query (above) to a CPS Clojure function (right). Here `'S30744` and `'O30742` provide unique identifiers for the (lexical) sample and observe statement respectively, while `var30743` and `do30741` are function identifiers.

method will cause inference algorithm specific behavior, e.g. updating a weight in $\mathcal{S}$, and provide a return value `next`. If `next` is a function, the loop is recurred, with `step` now replaced by the output of a trampolined call of `next`. If `next` is not a function, the **`exec`** function terminates, returning `next`. The reason for this looping is that it allows different inference algorithms to specify when control needs to be passed back to the **`infer`** function in an algorithm specific way, by defining the checkpoint methods to either return a checkpoint object (for which the **`exec`** call will terminate) or the continuation wrapped in a thunk (for which execution will continue). For example, when running SMC, action distinct to running the program forwards only needs to be taken at the **`observe`** statement checkpoints. As different traces may have a different number of **`sample`** statements between observations, it is therefore convenient to a have a function that does not terminate at the **`sample`** checkpoints, returning only once an **`observe`** or the end of the program is reached. On the other hand, for MCMC samplers, we will need to externally control the sampling behavior and so it may be necessary to return control at each **`sample`** statement.

We have now demonstrated how an Anglican query is compiled, a summarizing example for which is provided in Figure 7.1. We have also, by proxy, laid out a framework for writing inference algorithms in Anglican: we need to implement methods of the **`checkpoint`** multimethod and a top level inference function **`infer`**. In the rest of this Chapter we will explain how this abstraction allows us to write inference algorithms suitable for arbitrary programs.

7.3 Writing Inference Algorithms in Anglican

Taking stock, what has our compilation given access to and what do we not have access to? The key upshots of our compilation are that we have access to a continuation representing the rest of the program at any point during the execution and the interactions between the inference back-end and the query code happens only through the **`sample`** and **`observe`** statements and the final returned $\mathcal{S}$ of each execution. Consequently, we have means of running the program

forwards and controlling the behavior at each **sample** and **observe** statement. We can stop or restart the program at any one of our checkpoints and chose whether to do so adaptively, for example, by killing of certain evaluations or duplicating others. Once a program is run, we can test the effect of changing one or more of our sampled variables in an MCMC fashion, though it may be difficult to statically determine an appropriate global proposal as we can only establish the structure of the target through evaluation. Although we might be able to do so with other code analysis, we do not directly have access to any information about independence relationships or even knowledge under what conditions a certain variable will exist. In theory, we can query the unnormalized density $\gamma(x_{1:n_x}, \lambda)$ of any fixed set of parameter values $x_{1:n_x}$, given fixed inputs λ, by effectively making **sample** operate as an **observe** statement. However, it may be difficult, or even impossible, to find values of $x_{1:n_x}$ that have non-zero probability unless they are generated by running the program forwards. In particular, it might be arbitrarily hard to find a trajectory that satisfies $\mathcal{B}(x_{1:n_x}, \lambda) = 1$, or construct an external proposal more generally, without forward sampling from the program.

These availabilities and restrictions suggest that Anglican is, and universal PPSs more generally are, best suited to *evaluation based* inference methods. At a high level, we can think of such methods as acting like a controller wrapping around the standard program execution (or potentially multiple simultaneous executions). When the program hits a checkpoint, control is transferred to the inference engine, providing information about the current state through $\mathcal{S}$ and about the current checkpoint command (i.e. checkpoint type, distribution object, etc). The inference engine then decides exactly how the checkpoint command should be executed (e.g. how to sample the new variable for a **sample** statement), updates the $\mathcal{S}$ appropriately, and decides if and how the execution should continue. Once an execution is complete (i.e. the return checkpoint is reached), the inference controller decides what to run next, which could, for example, be an entirely new independent execution or an MCMC update to the current execution. Two important things the inference controller does not do is alter the deterministic elements of the program (which we can think of as being single deterministic functions from one checkpoint to the next) or do further analysis on the program source code itself.

There are four functions required to define an Anglican inference algorithm more generally: a top-level **infer** function and three **checkpoint** implementations for types `trap.sample`, `trap.observe`, and `trap.return` . The **infer** implementation must always be written anew, but the **checkpoint** implementations can be inherited from another inference algorithm of default behavior. Other than some top-level book keeping, the user inference command

doquery directly calls the **infer** function of the specified inference algorithm, providing as input the CPS-transformed query, the fixed query inputs, and inference algorithm specific options. The **infer** function is required to then return a lazy infinite sequence of samples, each in the form of a $\mathcal{S}$. To carry out inference, **infer** uses the **exec** function introduced earlier to run (partial) executions of the program, which will run until they hit a checkpoint that returns something other than a thunk (see below).

All of the **checkpoint** implementations take as their only input of note an instance of the relevant Clojure record, i.e. a `trap.sample`, `trap.observe`, or `trap.result`. Here `trap.sample` has fields **:id** which is a unique identifier for the corresponding lexical **sample** statement, **:dist** which is the distribution object input, **:cont** which is the continuation function $\mathcal{P}$, and **:state** which is the current instance of $\mathcal{S}$. Meanwhile, `trap.observe` has the same fields and additionally a **:value** field giving the observation and `trap.result` has only the **:state** field. Each **checkpoint** method should either return a thunk, in which case the **exec** function will resume running, or an instance of the relevant Clojure record, in which case **exec** terminates and returns control to the **infer** function (or some intermediate function used by the **infer** function). For example, in SMC methods then we will want to return control to **infer** at an **observe** point to do the resampling, while all methods return control for a **return** checkpoint. Default behaviors are provided for each checkpoint that dictate their behavior unless overwritten by a particular inference algorithm. The default behavior of **sample** is to sample from the distribution object and return a thunk that calls the continuation with this sample and $\mathcal{S}$ as input arguments. The default behavior of **observe** is to update the **:log-weight** field in $\mathcal{S}$ with the current observation term and then returns a thunk which calls the continuation with inputs **nil** and $\mathcal{S}$. The default behavior of **return** simply returns the `trap.result` object.

7.4 Inference Strategies

As we previously alluded to, the simplest inference strategy we can carry out is importance sampling. This is in fact so simple in our current framework that it uses the default behavior of all the checkpoints, while the **infer** function only involves constructing a lazy infinite sequence of the output from independent calls of **exec** on the full program. Keeping the default behavior for the **sample** implicitly means that our inference will use a bootstrap proposal (i.e. the generative model is taken as the proposal). Though not technically required, this is still a highly convenient choice of proposal as amongst other things, this ensures that we can always

sample from the proposal and that the proposal is valid in terms of its tail behavior (presuming the conditional probabilities are bounded).

7.4.1 MCMC Strategies

We have already explained why it might be difficult to construct a global MH proposal for our program due to the difficulties in varying dimensionality and unknown variable supports. One solution to this is to try and use compilation to establish this support so that a valid proposal can be specified. However, doing this in a general manner is somewhat challenging and remains an open problem in the field. Another more immediately viable approach is to use a proposal that looks to update particular x_j in the trace in a component-wise MH manner (see Section 5.3.4.5), potentially rerunning the rest of the program after that choice if necessary. In the context of PPSs such approaches are generally known as either single-site MH, random database sampling, or lightweight MH (LMH), and were originally suggested by [Wingate et al., 2011].[2] There are two key factors that make such approaches viable. Firstly, if we update a term x_j in our trace, there is no need to update any of the factors in our trace probability that occur before the sampling x_j, while we may be able to avoid evaluating many of those after as well by establishing conditional independence. Secondly, the support for a particular x_j given $x_{1:j-1}$ is typically known (as we have access to the appropriate distribution object) and so it is generally possible to specify an appropriate proposal for an individual x_j. Even if that transpired not to be possible, simply using the prior as the proposal is still often reasonable as individual x_j terms will typically be low dimensional. However, a key downside of this approach is that changing a particular x_j might lead to an invalid trace and checking for this might require revaluation of the entire rest of the trace, making it an $O(n_x + n_y)$ operation. Perhaps even more problematically, if n_x is not fixed then, unless we make adjustments to the algorithm, the approach will not produce a valid Markov chain. Similarly, as we showed in Section 5.3.4.5, there are models for which component-wise MH approaches lead to reducible Markov chains that no longer admit the correct target. The emphasis on branching in universal PPSs further means that the chance of falling into this category of invalid models is relatively high. As noted by [Kiselyov, 2016]), this is a rather serious issue missed (or at least not acknowledged) by Wingate et al. [2011] and various follow-up implementations. See also issues highlighted by [Hur et al., 2015] in other implementations. The problem is not shared by all implementations though, with the Anglican LMH method, amongst others, not suffering from the issue. More generally, there are means

[2]The MH acceptance ratio in the original version of this work is incorrect so we point the reader to the following updated version `https://stuhlmueller.org/papers/lightweight-mcmc-aistats2011.pdf`.

of potentially overcoming both the reducibility and computational issues of LMH as we will discuss later, but they still represent noticeable practical and theoretical hurdles.

As a simple illustrative approach that avoids some of the theoretical pitfalls associated with LMH approaches, consider an MH sampler whose proposal simply chooses one of the **sample** statements, m, uniformly at random from $\{1, \ldots, n_x\}$, proposes a new x'_m using a local reversible MH kernel for that **sample** statement $\kappa(x'_m|x_m)$ (which can always be independently sampling from $f_m(x_m|\eta_m)$ if necessary) and then reruns the entire rest of the program from that point [Wood et al., 2014]. This equates to using the proposal[3]

$$\begin{aligned} q(x'_{1:n'_x}|x_{1:n_x}) &= q(m|x_{1:n_x})q(x'_m|x_m, m)q(x'_{m+1:n_x}|x_{1:m-1}, m, x'_m)\mathbb{I}(x'_{1:m-1} = x_{1:m-1}) \\ &= \frac{1}{n_x}\kappa(x'_m|x_m)\mathbb{I}(x'_{1:m-1} = x_{1:m-1}) \prod_{j=m+1}^{n'_x} f_{a'_j}(x'_j|\eta'_j) \end{aligned} \tag{7.3}$$

which in turn gives an acceptance probability of

$$\begin{aligned} P(\text{Accept}) &= \min\left(1, \frac{\gamma(x'_{1:n'_x}, \lambda)q(x_{1:n_x}|x'_{1:n'_x})}{\gamma(x_{1:n_x}, \lambda)q(x'_{1:n'_x}|x_{1:n_x})}\right) \\ &= \min\left(1, \frac{\kappa(x_m|x'_m)f_{a_m}(x'_m|\eta_m)}{\kappa(x'_m|x_m)f_{a_m}(x'_m|\eta_m)} \frac{n_x}{n'_x} \frac{\prod_{k=k_0(x_{1:m})+1}^{n'_y} g_{b'_k}(y'_k|\psi'_k)}{\prod_{k=k_0(x_{1:m})+1}^{n_y} g_{b_k}(y_k|\psi_k)}\right) \end{aligned} \tag{7.4}$$

where $k_0(x_{1:m})$ is equal to the number of **observe** statements encountered by the partial trace $x_{1:m}$. Here we have used the fact that the trace probability factors for terms before x_m cancel exactly, while the **sample** terms for $m+1$ onwards cancel between the target and the proposal. We can intuitively break down the terms in (7.4) by first noting that the first ratio of terms, $\frac{\kappa(x_m|x'_m)f_{a_m}(x'_m|\eta_m)}{\kappa(x'_m|x_m)f_{a_m}(x'_m|\eta_m)}$, is what we would get from doing MH targeting $f_{a_m}(x_m|\eta_m)$. The next ratio of terms, n_x/n'_x reflects the fact that if our new trace is longer it is less likely that we would have chosen the point m to resample and vice versa. The final ratio of terms reflects the ratio of the likelihood weight for the new generated section of the trace over the existing section of the trace.

This approach avoids the reducibility issues because it includes as a possible step generating a completely new trace from the generative model. However, it will clearly be heavily limited in practical performance because each iteration is effectively using importance sampling in $n_x - m$ dimensions such that, once the sampler has burnt in, its acceptance rate will typically decrease dramatically with dimension. Each iteration is also $O(n_x + n_y)$ as the complete new trace needs to be proposed each time. To get around these issues, we would like to have some concept of whether we can update x_m without invalidating the trace. This can sometimes be done using code analysis or compilation to establish the Markov blanket of x_m [Yang et al., 2014;

[3]We omit our trace validity term because it always holds when sampling from the generative model.

Mansinghka et al., 2014; Ritchie et al., 2016b], thereby providing a representation of which terms will be unaffected by an update. One needs to be careful, however, not to reintroduce reducibility issues and, in general, correctness of LMH schemes is far from trivial and perhaps requires further consideration in the literature.

Even if we do not have a convenient means of extracting Markov blankets, it will still generally be desirable to reuse $x_{m+1:n_x}$ if it produces a valid trace, i.e. if $\mathcal{B}([x_{1:m-1}, x'_m, x_{m+1:n_x}], \lambda) = 1$, and reducibility issues can be guarded against, in order to avoid the terrible scaling in the acceptance rate of (7.4) with $n_x - m$. One way to do this would be to propose a new x'_m in the same way but then deterministically run the program forward with the old $x_{m+1:n_x}$ to evaluate whether the trace is valid, noting that if it is, it must also be the case that any extension or reduction of the trace, i.e. increasing or decreasing n_x, is invalid as the program is deterministic given $x_{1:n_x}$. If $x_m \to x'_m$ gives a valid trace, then $\gamma([x_{1:m-1}, x'_m, x_{m+1:n_x}], \lambda)$ will be well defined (though not necessarily non-zero) and we can use the update as a proposal without needing to regenerate $x_{m+1:n_x}$. Note though that as the a_j, b_k, η_j, and ψ_k terms downstream of m can change, the probability of the new trace still needs recalculating. In this scenario, our acceptance ratio becomes

$$P(\text{Accept}) = \min\left(1, \frac{\gamma(x'_{1:n_x}, \lambda)\kappa(x_m|x'_m)}{\gamma(x_{1:n_x}, \lambda)\kappa(x'_m|x_m)}\right) \tag{7.5}$$

where all terms in $\gamma(x'_{1:n_x}, \lambda)$ (i.e. both **`sample`** and **`observe`**) will need to be evaluated for $j \geq m$, as will $\gamma(x_{1:n_x}, \lambda)$ if these are not already known.

If, on the other hand, our trace becomes invalid at any point (noting that we can evaluate the validity of sub-traces as we rerun them), we can resort to resampling the trace anew from the required point onwards. Note that we can do this validity evaluation and regeneration during the same single forward pass through trace and that whether we do regeneration is deterministic for given $\{x_m, x'_m\}$ pair (and thus does not affect the MH acceptance ratio by symmetry). If we regenerate from **`sample`** ℓ onwards then we now have

$$P(\text{Accept}) = \min\left(1, \frac{\gamma(x'_{1:n_x}, \lambda)\kappa(x_m|x'_m)n_x \prod_{j=\ell}^{n_x} f_{a_j}(x_j|\eta_j)}{\gamma(x_{1:n_x}, \lambda)\kappa(x'_m|x_m)n'_x \prod_{j=\ell}^{n'_x} f_{a'_j}(x'_j|\eta'_j)}\right). \tag{7.6}$$

We can further overcome reducibility issues with such a scheme by forcing a fresh trace generation not only when we realize the trace is invalid, but also when we find a term in our trace with probability zero. Once this occurs, it is clear that the full trace probability must be zero, even if it produces a valid path. We could just immediately reject the trace, but we would not solve the reducibility issues. By instead regenerating the rest of the trace in this scenario, at each iteration

we can propose any value of x_1 that has non-zero marginal mass under $\gamma(x_{1:n_x}, \lambda)$ regardless of our current state, so we clearly have mixing on x_1. For a given value of x_1, we can similarly propose any value of x_2 with non-zero density under the marginal conditional distribution of $x_2|x_1$. As we have mixing of x_1, this now implies we have mixing on $x_{1:2}$ as well. By induction, our method now leads to mixing on all of $x_{1:n_x}$, which coupled with detailed balance provides an informal demonstration of the consistency of the method.

We can refine this process further by using the concept of a database. To do this we mark each **sample** statement in the trace with a unique identifier that is common to all traces that evaluate the same **sample** statement at the same point, i.e. points in the trace for which both the **sample** number j and the lexical **sample** identifier a_j are the same. Our database can be used to store previous samples and deterministically return them when revisiting a **sample** statement with the same identifier if this old value still constitutes a valid sample with non-zero probability, regenerating it if not. If we use $\mathbb{D}(j) = 0$ to denote terms taken from the database and $\mathbb{D}(j) = 1$ to indicate terms that are redrawn, then our acceptance ratio now becomes

$$P(\text{Accept}) = \min\left(1, \frac{\gamma(x'_{1:n_x}, \lambda)\kappa(x_m|x'_m)n_x \prod_{j=m+1}^{n_x}\left(f_{a_j}(x_j|\eta_j)\right)^{\mathbb{D}(j)}}{\gamma(x_{1:n_x}, \lambda)\kappa(x'_m|x_m)n'_x \prod_{j=m+1}^{n'_x}\left(f_{a'_j}(x'_j|\eta'_j)\right)^{\mathbb{D}(j)}}\right). \tag{7.7}$$

Note the importance of points in the database being defined by both j and a_j – if we hit the same lexical sample statement at a different point in the program, we always need to redraw it. Care is also needed to ensure superfluous terms are removed from the database at each iteration – it should contain only terms from the current trace.

Note, however, that our identification scheme for points in the database, namely points for which both j and a_j are the same have the same point in the database, is potentially stronger than necessary and there may be more useful addressing schemes. Remembering back to Section 4.3.2, each of the **sample** and **observe** statements are exchangeable up to the required inputs being in scope. Now consider the case where the update $x'_m \leftarrow x_m$ triggers an inconsequential extra **sample** to be invoked between points j and $j+1$, with everything else staying the same. Under our current system then all points after j would need to be resampled. However, our program would define the same distribution if this superfluous **sample** statement came at the end of the program, in which case our method would mean that $x_{j+1:n_x}$ no longer necessarily needs updating. From a more practical perspective, we can consider using more useful naming strategies for our database entries that exploit this exchangeability such that x'_{j+2} could, for example, inherit from x_{j+1} [Wingate et al., 2011].

We finish the section by briefly discussing some choices in the proposal. The simplest choice for $\kappa(x'_m|x_m)$ is just to redraw a new value from the prior. In this case then all the $\frac{\kappa(x_m|x'_m)f_{a_m}(x'_m|\eta_m)}{\kappa(x'_m|x_m)f_{a_m}(x'_m|\eta_m)}$ terms will cancel in our acceptance ratios and giving a trivially valid proposal, other than the aforementioned reducibility issues. We will refer to this strategy simply as LMH in the rest of the thesis. For models with significant prior-posterior mismatch, such an approach could be slow to mix as it does not allow for any locality in the moves (i.e. $\kappa(x'_m|x_m)$ is independent of x_m). As suggested by, for example, Le [2015], one can sometimes achieve improvements by instead using the type of the distribution object associated with x_m to automatically construct a valid random walk proposal that allows for improved hill climbing behavior on the individual updates. We will refer to this method as RMH elsewhere in the thesis.[4] Thus far, we have presumed that which `sample` statement to update at each iteration is selected uniformly at random. This is not actually a necessary assumption, with Tolpin et al. [2015a] demonstrating that one can construct an adaptive LMH (ALMH) that adaptively updates proposal probabilities for which term in the trace to update.

7.4.2 Particle-Based Inference Strategies

7.4.2.1 Sequential Monte Carlo

Going from importance sampling to SMC in our framework is remarkably simple from an implementation perspective [Wood et al., 2014; Paige and Wood, 2014]. The behavior of the `sample` and `result` checkpoints are kept as per the default. The `observe` checkpoints are redefined to carry out the same operations, but return a record rather than a thunk, returning control to the `infer` function. This means that calling `exec` for the SMC checkpoint setup will run the program up to and including the next `observe` statement. Consequently, if we run multiple threads of `exec` at once, each corresponding to a separate particle, these will all stop exactly when the next resampling point is required for SMC. Thus all the `infer` function needs to do for SMC, other than some bookkeeping, is alternate between mapping an `exec` call across all of the particles and performing resampling steps (remembering to reset the internal weights for the traces to be the same). The marginal likelihood estimate can also be calculated in the standard way, so the required lazy infinite sequence of output samples can be produced by running independent SMC sweeps and setting the weights to the product of the sweep marginal likelihood and local sample weight.[5]

[4] Note that the Anglican RMH implementation uses a mixture proposal where it samples from the prior, as per LMH, half the time and from the random walk proposal the other half.

[5] In practice, Anglican resamples after the last observation, so the local sample weights are all actually the same.

From a theoretical perspective, running SMC in Anglican requires us to make one small model assumption – that the number of observations n_y is fixed. In practice, this assumption is usually satisfied, particularly if there are no observations of internally sampled variables. Violations are caught at run time. Given a fixed n_y, we can define the series of targets for SMC as being the distributions induced by running the program up to the t^{th} `observe` statement, namely

$$\gamma_t(x_{1:n_x}, \lambda) = \begin{cases} \prod_{j=1}^{n_x} f_{a_j}(x_j|\eta_j) \prod_{k=1}^{t} g_{b_k}(y_k|\psi_k) & \text{if } \mathcal{B}_t(x_{1:n_x}, \lambda) = 1 \\ 0 \quad \text{otherwise} \end{cases} \tag{7.8}$$

where $\mathcal{B}_t(x_{1:n_x}, \lambda)$ is a function establishing the validity of the partial program trace. More formally, we can define $\mathcal{B}_t(x_{1:n_x}, \lambda)$ as being a function indicating validity of a trace for transformation of the original program that terminates after making its t^{th} observe. It may be that executions corresponding to different particles have not gone through the same `sample` and `observe` statements at any particular point, but this not a problem, from a theoretical perspective, as provided that n_y is fixed, (7.8) still defines an appropriate series of targets for SMC inference. Although changing the position of the `observe` statements in our program does not change the final distribution targeted by running SMC, we note that it can change the intermediate target distributions, by adjusting at what point during the series of targets the `sample` statements are introduced. Consequently, changing the position of the `observe` statements can have a dramatic effect on the practical performance of the inference, e.g. placing all the `observe` statements just before the program returns will cause the algorithm to reduce to basic importance sampling. The earlier the `observe` statements are in the program, or more precisely the later variables are sampled relative to the `observe` statements, the better inference will perform as less information is lost in the resampling. Tricks such as lazily sampling variables (such that `sample` statements are only invoked when needed) can, therefore, lead to substantial performance gains.

7.4.2.2 Particle Gibbs

Provided one does not try to support the special treatment of global variables that particle Gibbs allows (i.e. restricting to the iterated CSMC case), extending SMC to particle Gibbs is relatively straightforward in our framework. From a theoretical perspective, the algorithm extends from the SMC case in the same way as outside of the probabilistic programming framework. From a practical perspective, there are two distinct challenges. Firstly, resampling for CSMC sweeps does not maintain the target distribution in the same way as SMC [Holenstein, 2009] and so one has to be careful that there is no possibility for gratuitous resampling or missing a required resampling step (e.g. we cannot use the adaptive resampling discussed in Section 6.1.5.2). At

first this would make it seem like one would need to take care not to resample after the n_y^{th} observation. However, resampling and choosing the retained particle uniformly at random from the present particles turns out to be identical to sampling the retained particle in proportion to weight and so this is, in fact, not a problem. Secondly, we need a method of storing and retrieving the state of the retained particle in a manner that allows other particles to inherit from it. Given our inference methods do not use a stack, this is done by storing the raw $x_{1:n_x}$ samples within $\mathcal{S}$ for each particle and then retrieving them for the retained particle. The retained particle is thus re-run in a deterministic fashion from a stored $x_{1:n_x}$, regenerating all the variables in the program and the continuation. This is achieved by editing the `sample` checkpoint so that it deterministically retrieves the x_j for the retained particle, but samples normally for other particles. For all particles it also stores the current x_j in case this particle becomes the retained particle at the next sweep. Provided one is running a reasonably large number of particles the extra computational cost for re-running the retained particle is negligible, but the need to store all the $\{\hat{x}_{1:n_x}^n\}_{n=1}^N$ can noticeably increase the memory requirements. Other than these minor complications, the extension from SMC to particles Gibbs in the probabilistic programming setting is identical to that for conventional settings as discussed in Section 6.2.2.

7.4.2.3 Interacting PMCMC

Given the particle Gibbs and SMC implementations, iPMCMC can be implemented relatively simply by using those implementations for the CSMC and SMC sweeps respectively. All checkpoint implementations are inherited from particle Gibbs and the extension does not have any distinct challenges unique to probabilistic programming. We note though that the Anglican implementation of iPMCMC exploits the algorithmic ability for parallelization by creating a pool of threads to distribute computation of the different nodes. Consequently, the implementation has substantial computational benefits over that of say particle Gibbs, in addition to the improved per-sample performance demonstrated in Section 6.3.

7.4.3 Other Methods

Though it would be impractical to do justice to all of the methods one can use for general purpose in universal PPSs, we briefly lay out some other algorithms of particular note.

The particle cascade [Paige et al., 2014] is an asynchronous and anytime variant of SMC that can provide improved in-sweep parallelization compared to SMC and an alternative to PMCMC methods for overcoming memory restrictions in SMC. It has the crucial advantage over PMCMC methods of maintaining an unbiased estimate of the marginal likelihood, but it can

suffer instability issues in the number of particles produced by the schedule strategy during a sweep. It also does not permit any distinct treatment of global variables.

Particle Gibbs with ancestor sampling (PGAS) [Lindsten et al., 2014; van de Meent et al., 2015] looks to alleviate degeneracy for the particle Gibbs algorithm by performing ancestor sampling for the retained particle. This can substantially improve mixing for the early samples, but can result in very large computational overheads, sometimes requiring $O(Nn_y^2)$ computation (compared to $O(Nn_y)$). This can be partly mitigated by using database techniques to avoid unnecessary repeated computation [van de Meent et al., 2015], but is still typically substantially more computationally intensive than standard particle Gibbs.

Some black-box variational methods [Ranganath et al., 2014] have been developed for general purpose PPSs [Kucukelbir et al., 2015] including Anglican [van de Meent et al., 2016; Paige, 2016]. Though these can suffer from difficulties in finding effective addressing schemes and or high variance of the ELBO gradients, they can still form an effective approximate approach for certain problems.

7.4.4 Which Anglican Inference Algorithm Should I Use?

The question of which inference algorithm one should use is a critical, but somewhat subjective and problem dependent question. We now provide some (inevitably biased) practical recommendations for which inference algorithms to use in Anglican in its current form. These should be taken as a rough starting point rather than clear-cut truth.

Firstly, except for very low dimensional problems, importance sampling should be avoided as it is almost universally worse than, e.g. SMC. Similarly, there is very little reason to use PIMH instead of SMC or other PMCMC methods, while RMH or ALMH should generally be preferable to vanilla LMH. The relative performance of the MCMC based and particle-based (including PMCMC) methods will depend on the amount of structure that can be exploited in the system by the respective approaches. The particle-based approaches will generally be substantially preferable when the `observe` statements are interlaced with the `sample` statements so that the intermediate information can be exploited. However, if all the variables need to be sampled before making observations, or if all the variables are completely independent of one another given the data, SMC will reduce to importance sampling, while the MCMC methods can still take advantage of hill-climbing effects.

If using SMC or PMCMC based methods, then, as per Section 6.1.5.1, the most important thing is not the exact algorithm, but ensuring that sufficient particles are used. Because PMCMC

methods in Anglican do not use separate updates for global parameters, using enough particles can be even more important than in conventional settings. Presume that your computational budget is M particles and your memory budget is N particles. Our recommendation is that if $M < N$, you should look to run a single SMC sweep with M particles – all the more advanced algorithms are mostly setup to avoid issues that are easily solved by running more particles when you can. For larger values of M, then you should keep the number of particles as high as possible (ideally N). Our recommendation here is to use iPMCMC as the go-to algorithm as this will rarely be noticeably worse than the other particle-based approaches and sometimes substantially better, both in terms of per-sample performance and its support for parallelization. Two possible exceptions to this are that PGAS and the particle cascade can sometimes still be better when $M \gg N$, with the latter also supporting effective parallelization of computation. For example, PGAS can be very effective in models where there are extreme restrictions on the number of particles that can be run, e.g. due to calling an expensive external simulator.

8

A Predominantly Bayesian Approach to Optimization

From designing engineering components to managing portfolios and tuning control systems, optimization is used ubiquitously throughout engineering and the sciences. Whereas inference dealt with cases where we desire to calculate an expectation or approximate a probability distribution given data, optimization considers cases where only a single output is desired. For example, in a decision problem only a single action can be taken, meaning a single optimal action must be found. It is also often used in machine learning as a tractable alternative when full inference is infeasible [Murphy, 2012]. In the next Chapter, we discuss a method for introducing optimization into the probabilistic programming framework. Before doing this though, we provide background on optimization in general and detail a particular approach called Bayesian optimization [Jones et al., 1998; Shahriari et al., 2016b].

8.1 Optimization in Probabilistic Machine Learning

In general, all optimization problems can be expressed in the form

$$\begin{aligned} \theta^* &= \arg\max_{\theta \in \vartheta} f(\theta) \\ \text{s.t. } & g_i(\theta) = c_i \quad \forall i \in \{1, \ldots, n_e\} \\ & h_i(\theta) \geq d_i \quad \forall i \in \{1, \ldots, n_i\} \end{aligned} \tag{8.1}$$

where $f \colon \vartheta \to \mathbb{R}$ is the target function with input variables θ, ϑ is the space of permissible solutions, $g_i(\theta) = c_i$ are equality constraints, $h_i(\theta) \geq d_i$ are inequality constraints, and θ^* is the optimum value of θ. We note that careful definition of ϑ can remove the need to define constraints explicitly, a fact which we exploit in Chapter 9, while minimization problems can be dealt with by maximizing $-f(\theta)$ instead of $f(\theta)$.

A large range of problems are covered by (8.1). For example, there are arbitrary different types ϑ might correspond to from the set of real values to the setup of race car. There is also a wide range of information that might be known about f – we might have access to the exact function and all its derivatives, or it might be that the only way to evaluate it is to conduct an expensive real-world experiment with stochastic results at a given value of θ. Naturally this

leads to a plethora of different techniques to suit the different problems at hand such as dynamic programming [Bellman, 2013], evolutionary algorithms [Bäck, 1996], combinatorial optimization methods [Papadimitriou and Steiglitz, 1982], gradient-based methods [Boyd and Vandenberghe, 2004], and Monte Carlo based methods [Robert, 2004]. Such optimization algorithms can be split into two key categories: local optimization approaches and global optimization approaches. The latter strictly aims to solve the true target as laid out in (8.1), but is in often infeasible in practice. The former instead only tries to find a point where there is no better point in its immediate vicinity. Local optimization is generally used either when the problem is known or expected to be convex (such that there is only one such local optimum), or when global optimization is infeasible and a local optimum is expected to still for a good approximate solution, as is typically the case in, for example, neural network training.

In a probabilistic machine learning context, common optimization problems include maximum likelihood (ML) estimation, maximum a posterior (MAP) estimation, maximum marginal likelihood (MML) estimation, marginal maximum a posteriori (MMAP) estimation, and risk minimization. ML estimation looks to find the most probable set of parameters for a given distribution, namely given data Y they look to find

$$\theta^* = \arg\max_{\theta \in \vartheta} p(Y|\theta). \tag{8.2}$$

Many classical statistical and machine learning techniques on based on ML estimation as it is intuitively the most probable set of parameters given data and a model. However, this can be prone to overfitting and so it typically requires some form of regularization [Hastie et al., 2001].

MAP estimation corresponds to maximizing the posterior probability, which is equivalent to extending ML estimation to also include a prior term, noting that $p(Y)$ is a constant

$$\theta^* = \arg\max_{\theta \in \vartheta} p(\theta|Y) = \arg\max_{\theta \in \vartheta} p(Y|\theta)p(\theta). \tag{8.3}$$

This provides regularization compared to ML estimation (indeed many priors and ML regularization methods are exactly equivalent [Bishop, 2006]), but still has a number of drawbacks compared to full inference when the latter is possible. For example, the position of the MAP estimate is dependent of the parametrization of the problem (see Section 2.6). Using a MAP estimate also, of course, incorporates less information into the predictive distribution than using a fully Bayesian approach. Nonetheless, MAP estimation is still an important tool of Bayesian machine learning, necessary when a single estimate or decision is required, or when

inference is infeasible. Note that when ϑ is bounded, then ML estimation is recovered from MAP estimation by using a uniform prior.

MML estimation is often referred to using a number of different names such as expectation maximization, type-II maximum likelihood estimation, hyperparameter optimization, and empirical Bayes. Despite the multitude of names, the aim of all these methods is the same: to maximize the likelihood of a model marginalized over some latent variables X, namely

$$\theta^* = \arg\max_{\theta\in\vartheta} p(Y|\theta) = \arg\max_{\theta\in\vartheta} \mathbb{E}\left[p(Y, X|\theta)|Y, \theta\right] \tag{8.4}$$

where the expectation is with respect to $p(X|\theta)$. Common examples of MML problems include parameter tuning, decision making, model learning, and policy search [Deisenroth et al., 2013].

MMAP estimation varies from MML in the same manner as MAP from ML: the inclusion of a prior on the target parameters. Specifically it aims to optimize

$$\theta^* = \arg\max_{\theta\in\vartheta} p(\theta|Y) = \arg\max_{\theta\in\vartheta} p(Y, \theta) = \arg\max_{\theta\in\vartheta} \mathbb{E}\left[p(Y, X|\theta)p(\theta)|Y, \theta\right]. \tag{8.5}$$

As with MML and MMAP estimation, risk minimization problems consider the optimization of an expectation. The key difference is in the fact that they are a *minimization* rather than a *maximization*, namely they look to find

$$\theta^* = \arg\min_{\theta\in\vartheta} \mathbb{E}\left[L(X, Y; \theta)\right] \tag{8.6}$$

for some loss function L which is parametrized by θ and is typically positive only. Depending on the context, Y can either be taken as fixed, or the expectation can be over both X and Y. The latter corresponds to the classical risk taken in the frequentist view of supervised learning. At first, the change from maximization to minimization might seem trivial. However, the fact that probabilities are positive only means that the two problems behave very differently. For example, imagine the problem of designing a bridge and let $Y = \emptyset$, X be the loadings the bridge will see in its lifetime, and θ the design. Even though the probability of failure will generally be very low, the loss in the case where the bridge collapses is very large. Therefore the expected loss is dominated by rare adversarial events and the desire is to choose a "safe" value for θ. Such a problem cannot be encoded effectively as a MML or MMAP estimation problem where we are maximizing the probability of the bridge not collapsing, as this expected probability cannot be dominated by the rare adversarial loading cases. In essence, expectations over probabilities are "optimistic" and by construction focus on the more probable cases.

In the rest of this chapter, we will consider in more depth a particular approach for carrying out such optimizations known as Bayesian optimization (BO).

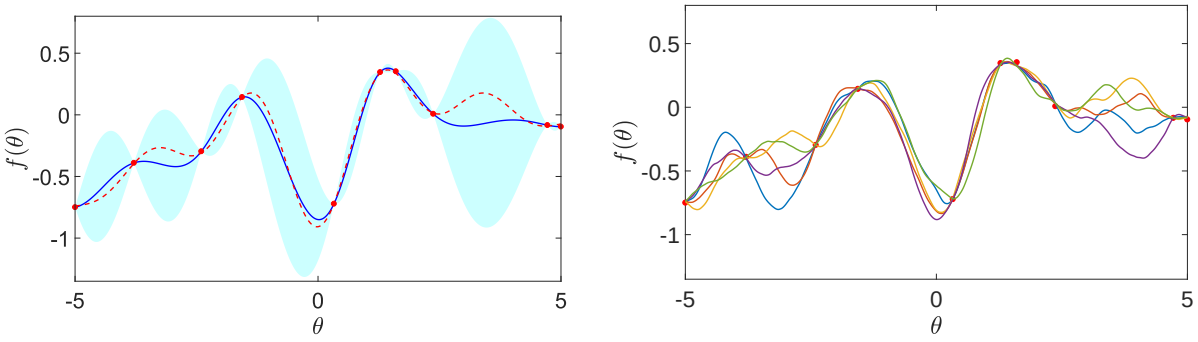


Figure 8.1: Example GP regression for points shown in left dots. Shown left is GP where the solid dark blue line is the mean of the GP and the light blue shading is the mean ± 2 standard deviations. Shown right is 4 example functions drawn from the GP with covariance function as per (8.8).

8.2 Gaussian Processes

Before introducing BO directly, we first digress to introduce Gaussian processes (GPs) as they are an essential precursor to much of the BO literature. As we will show in the next section, BO relies on modeling the target with a surrogate, which can then be used to estimate the expected value and uncertainty estimates for the value of a function away from points that have been evaluated. The most common surrogate taken in the BO literature is a GP and the one we will use later in Chapter 9, hence their central importance to the BO method.

Informally one can think of a GP [Rasmussen and Williams, 2006] as being a nonparametric distribution[1] over functions. Equivalently, one can think of it as a generalization of a Gaussian distribution to (uncountably) infinite dimensions. Practically, they are powerful tools for regression, classification, and feature extraction [Kuss and Rasmussen, 2005; Lawrence, 2004]. We focus here on regression as it is generally the simplest usage and the most relevant to our needs.

8.2.1 Function-Space View

A GP is fully specified by a mean function $\mu \colon \vartheta \to \mathbb{R}$ and covariance function $k \colon \vartheta \times \vartheta \to \mathbb{R}$, the latter of which must be a bounded (i.e. $k(\theta, \theta') < \infty,\ \forall \theta, \theta' \in \vartheta$) and reproducing kernel (we will return to this in depth later in Section 8.2.3). We can informally describe a function f as being distributed according to a GP:

$$f(\theta) \sim GP(\mu(\theta), k(\theta, \theta')) \tag{8.7}$$

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

[1] Technically speaking they are stochastic processes, not distributions.

Figure 8.1 shows an example GP regression with a small number of data points $\{\theta_j, v_j\}_{j=1:m}$. Here we wish to regress a function f from θ to y. To do this, we place a GP prior on f

$$f_{\text{prior}}(\theta) \sim \text{GP}\left(\mu_{\text{prior}}(\theta), k_{\text{prior}}(\theta, \theta')\right),$$

and take $\mu_{\text{prior}}(\theta) = 0 \; \forall \theta$, noting the generality of this choice as one can simply regress $y - \mu_{\text{prior}}(\theta)$ if a different μ_{prior} is desired. We further take for the covariance function

$$\begin{aligned} k_{\text{prior}}(\theta, \theta') =& \sigma_{3/2}^2 \left(1 + \sqrt{3}\,\frac{\theta - \theta'}{\rho_{3/2}}\right) \exp\left(-\sqrt{3}\,\frac{\theta - \theta'}{\rho_{3/2}}\right) + \\ & \sigma_{5/2}^2 \left(1 + \sqrt{5}\,\frac{\theta - \theta'}{\rho_{5/2}} + \frac{5}{3}\left(\frac{\theta - \theta'}{\rho_{5/2}}\right)^2\right) \exp\left(-\sqrt{5}\,\frac{\theta - \theta'}{\rho_{5/2}}\right) \end{aligned} \tag{8.8}$$

which corresponds to a combination of a Matérn-3/2 and Matérn-5/2 kernel with hyperparameters $\sigma_{3/2}, \sigma_{5/2}, \rho_{3/2}$, and $\rho_{5/2}$. We will return to how to choose the kernel function and these parameters later, the latter of which are set to $\sigma_{3/2} = 0.04, \sigma_{5/2} = 0.47, \rho_{3/2} = 1.89$, and $\rho_{5/2} = 0.98$ in Figure 8.1. For now, we simply note that the choice of covariance function and its parameters will heavily influence the nature of functions generated by the GP, dictating things such as smoothness and characteristic length scales of variation.

An important property of a GP that we now exploit is that it is conjugate with a Gaussian likelihood. Let $\Theta = \{\theta_j\}_{j=1:m}$ and $V = \{v_j\}_{j=1:m}$ be the set of observed inputs and outputs respectively. We use a separable Gaussian likelihood function

$$p(V|\Theta, f) = \prod_{j=1}^{m} p(v_j|f(\theta_j)) = \prod_{j=1}^{m} \frac{1}{\sigma_n\sqrt{2\pi}} \exp\left(-\frac{\left(v_j - f(\theta_j)\right)^2}{2\sigma_n^2}\right) \tag{8.9}$$

where σ_n is an observation noise, set to 0.001 in our example. Combining this with the GP prior we previously defined leads to GP process posterior and predictive distribution. To see this we can consider the joint distribution between the points we have considered so far, and a new set of points $\{\Theta^*, V^*\}$. Introducing the shorthand $k_{\text{prior}}(\Theta, \Theta^*) = \begin{bmatrix} k_{\text{prior}}(\theta_1, \theta_1^*) & k_{\text{prior}}(\theta_1, \theta_2^*) & \dots \\ k_{\text{prior}}(\theta_2, \theta_1^*) & k_{\text{prior}}(\theta_2, \theta_2^*) & \dots \\ \dots & \dots & \dots \end{bmatrix}$ then we have by the fact that any finite realizations of points is Gaussian

$$\begin{bmatrix} V \\ f^* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} k_{\text{prior}}(\Theta, \Theta) + \sigma_n^2 I & k_{\text{prior}}(\Theta, \Theta^*) \\ k_{\text{prior}}(\Theta^*, \Theta) & k_{\text{prior}}(\Theta^*, \Theta^*) \end{bmatrix}\right) \tag{8.10}$$

where I is the identity matrix, $\mathbf{0}$ is a vector of zeros, and f^* is the true function values at Θ^* (such that $V^* - f^* \sim \mathcal{N}(0, \sigma_n^2 I)$). We can now use standard results for Gaussians (see e.g. Petersen

et al. [2008]) to get the conditional distribution for f^* given all the other variables

$$\begin{aligned} f^*|\Theta, V, \Theta^* &\sim \mathcal{N}\left(\mu_{\text{post}}\left(\Theta^*\right), k_{\text{post}}\left(\Theta^*, \Theta^*\right)\right) \quad \text{where} \\ \mu_{\text{post}}\left(\Theta^*\right) &= k_{\text{prior}}\left(\Theta^*, \Theta\right)\left[k_{\text{prior}}\left(\Theta, \Theta\right) + \sigma_n^2 I\right]^{-1} V \\ k_{\text{post}}\left(\Theta^*, \Theta^*\right) &= k_{\text{prior}}\left(\Theta^*, \Theta^*\right) - k_{\text{prior}}\left(\Theta^*, \Theta\right)\left[k_{\text{prior}}\left(\Theta, \Theta\right) + \sigma_n^2 I\right]^{-1} k_{\text{prior}}\left(\Theta, \Theta^*\right). \end{aligned} \tag{8.11}$$

Now as Θ^* are arbitrary points, this corresponds to our predictive distribution. Further, as the predictive distribution is still a GP, we can refer to our model as having a GP posterior

$$f_{\text{post}}(\theta)|\Theta, V \sim \text{GP}\left(\mu_{\text{post}}(\theta), k_{\text{post}}(\theta, \theta')\right).$$

Going back to Figure 8.1, we see the result of this process. Our GP regression gives us a posterior mean function that represents the expected value at every possible input points, along with a variance that represents our subjective uncertainty in the value of the function at that point. It is important to note that, as is always the case in Bayesian modeling, these uncertainty estimates are generally an underestimate, as they do not account for the "unknown unknowns". Figure 8.1 also shows that we can draw from the GP by choosing some set of evaluation points Θ^* and then drawing from (8.11). We see that there is a larger variation in the function values away from the points that have been evaluated, as would be expected.

An important point of note is that the marginal likelihood for our model is also analytic, namely

$$\log p(V|\Theta) = -\frac{1}{2} V^T \left[k_{\text{prior}}\left(\Theta, \Theta\right) + \sigma_n^2 I\right]^{-1} V - \frac{1}{2}\log\left|k_{\text{prior}}\left(\Theta, \Theta\right) + \sigma_n^2 I\right| - \frac{m}{2}\log 2\pi \tag{8.12}$$

where m is the number of points in Θ. This is important as it means it will be tractable to optimize or do inference over the GP hyperparameters.

8.2.2 Kernels and Hyperparameters

As we showed in the last section, GPs form powerful and expressive priors over functions. In this section, we show that their practical behavior varies substantially depending on the choice of the covariance function, aka kernel, and the hyperparameters. Informally, we can think of the kernel as expressing the similarity between the evaluation of the function at two different input points θ and θ'. When $k(\theta, \theta')$ is large, these evaluations are strongly correlated such that the evaluations will have similar values. When it is small, there is little correlation between the points and so their evaluations will be roughly independent. Note that it is possible for $k(\theta, \theta') < 0$ provided $\theta \neq \theta'$, but as kernels must be positive definition functions as explained in the next section, $k\left(\Theta, \Theta\right)$ is always a positive definite matrix. Though it is not necessary for the kernel to be stationary (i.e.

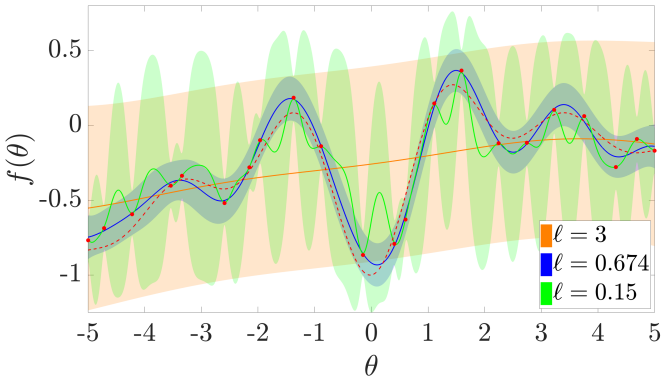


Figure 8.2: Effective of changing the length scale for Matérn-5/2 kernel.

that it only depends on $\theta - \theta'$ rather than the absolute values), we will not consider non-stationary cases further here (see Rasmussen and Williams [2006] for further information).

[Duvenaud, 2014, Figure 2.1] shows some simple example kernels. Note x here is equivalent to θ in our notation. The figure shows that the qualitative behavior of the GP changes substantially with changes to the kernel. All of these kernels also behave substantially differently to the compound Matérn kernel we considered in the last section. The choice of kernel is therefore critical to the performance of GPs. Though there is work, including some of our own [Janz et al., 2016], that examines methods for learning the kernel directly [Duvenaud et al., 2013; Lloyd et al., 2014; Wilson et al., 2014], this is typically too computationally intensive to carry out in practice. One must therefore either use prior knowledge about the problem, or use a relatively general purpose kernel to ensure that the nuances of the data are not overwhelmed.

A particularly common problem in the choice of kernel is in the desired level of smoothness. At the one extreme, one can use the squared exponential kernel

$$k_{\text{se}}(\theta, \theta') = \sigma_f^2 \exp\left(-\frac{\|\theta - \theta'\|_2^2}{2\rho^2}\right) \tag{8.13}$$

which is infinitely differentiable. Here σ_f^2 is the "signal standard deviation" – a hyperparameter that affects the scaling of the output from the GP: the larger σ_f, the higher the average standard deviation of the function. Meanwhile, ρ is a hyperparameter that dictates the characteristic length scale of variation for the function – the higher ρ is, the slower the correlation decreases as one moves away from the current point, and therefore the less "wiggly" the function is. Both these hyperparameters are shared by most commonly used kernels. Although the presented kernel is isotropic, this is easily relaxed by having a different ρ for each dimension.

In many practical problems, the squared exponential kernel is too smooth. One possible alternative in these scenarios are the Matérn kernels. The Matérn-ν kernel, given by

$$k_\nu(\theta, \theta') = \sigma_f \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu}}{\rho} \|\theta - \theta'\| \right)^\nu K_\nu \left(\frac{\sqrt{2\nu}}{\rho} \|\theta - \theta'\| \right) \tag{8.14}$$

where K_ν is the modified Bessel function of second kind order ν and $\Gamma(\cdot)$ is the gamma function, is $\lfloor \nu - 1 \rfloor$ times differentiable and therefore different values of ν can be used to express different levels of smoothness. Typically ν is set to $n + 1/2$ for some integer n, for which it has a simple closed form [Rasmussen and Williams, 2006].

Though the choice of kernel function is critical for a GP, the choice of hyperparameters such as the scaling and length scale can have equally significant impact on the behavior. Figure 8.2 shows the effect of changing the length scale of a Matérn-5/2 kernel. When the length scale is too large as shown in orange, the regressed function is overly smooth and does not capture the data. When the length scale is too small, as shown in green, wild fluctuations are permitted in between the datapoints such that there is very high uncertainty between points and also potentially overfitting as the regressor can model all the fluctuations in the data, including those originating simply from noisy evaluations. As the hyperparameters are rarely known upfront, one must typically either optimize them, or perform inference over them to produce a mixture of GPs.

8.2.3 Weight-Space View and Reproducing Kernel Hilbert Space

One of the key advantages of GPs is that they can be used effectively without needing to know the intricacies for why they work. The derivations we introduced in the last section and in particular the calculations required to derive the GP posterior were spectacularly simple, albeit computationally intensive (the need for a matrix inversion to calculate (8.11) means that training is $O(m^3)$ for m training points). However, what is going on beneath the surface is substantially more complicated, which is perhaps not surprising when one remembers that they are defined over uncountably infinite many variables. To truly understand the underlying rationale and assumptions for GPs, and to properly appreciate the restrictions on possible kernels, it is necessary to delve into GPs more formally, taking a so-called weight-space view. In this section, we, therefore, outline a more formal derivation of a GP from the viewpoint of reproducing kernels [Hofmann et al., 2008] and show how they can be thought of as Bayesian linear regression using an infinite number of features. We start with the following definitions.

Definition 8.1. *An inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ is a function $\mathcal{H} \times \mathcal{H} \to \mathbb{R}$ associated with a vector space $\mathcal{H}$ that is symmetric $\langle u, v \rangle_{\mathcal{H}} = \langle v, u \rangle_{\mathcal{H}}$, linear $\langle au_1 + bu_2, v \rangle_{\mathcal{H}} = a\langle u_1, v \rangle_{\mathcal{H}} + b\langle u_2, v \rangle_{\mathcal{H}}$, and positive definitive $\langle u, u \rangle_{\mathcal{H}} \geq 0,\ \langle u, u \rangle_{\mathcal{H}} = 0 \Leftrightarrow u = \mathbf{0}$.*

Definition 8.2. *A Hilbert space, $\mathcal{H}$, is a, potentially uncountably infinite, vector space associated with an inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ which is complete with respect to the norm $\|u\|_{\mathcal{H}} = \sqrt{\langle u, u \rangle_{\mathcal{H}}}$.*

Less formally, we can think of Hilbert spaces as being a generalization of Euclidean space (i.e. the space vectors live in) to include functions. Using Hilbert spaces, we can think of a function as being an uncountably long vector defining the value of the function at each possible input point.

Consider now the linear (in the weights $\mathbf{w}$) model

$$f(\theta) = \mathbf{w}^T Z(\theta) + \mu(\theta), \quad \mathbf{w} \sim \mathcal{N}(\mathbf{0}, C) \tag{8.15}$$

where $Z(\cdot) = [\zeta_1(\cdot), \ldots, \zeta_m(\cdot)]^T$, $\zeta_a \colon \vartheta \to \mathbb{R}$ for $a = 1, \ldots, m$ is a feature map in m-dimensional Hilbert space $\mathcal{H}$. For any set of points $\theta_1, \ldots, \theta_t$ then $F = [f(\theta_1), \ldots, f(\theta_t)]^T$ will be distributed according to a $t-$dimensional Gaussian with mean $[\mu(\theta_1), \ldots, \mu(\theta_t)]^T$ and covariance $k(\theta_i, \theta_j) = Z(\theta_i)^T C Z(\theta_j),\ \forall i, j \in \{1, \ldots, t\}$. Note that

$$Z(\theta)^T C Z(\theta') = \sum_{a=1}^{m} \sum_{b=1}^{m} C_{ab} \zeta_a(\theta) \zeta_b(\theta') \geq 0, \quad \forall \theta, \theta' \in \vartheta \tag{8.16}$$

as C must be positive semi-definite for the distribution on $\mathbf{w}$ to be well defined[2]. This also means that the Cholesky decomposition $C^{1/2}$ exists and we can define $\Psi(\cdot) = C^{1/2} Z(\cdot) = [\psi_1(\cdot), \ldots, \psi_m(\cdot)]^T$ to give a covariance function $k \colon \vartheta \times \vartheta \to \mathbb{R}$ of the form

$$k(\theta, \theta') = \langle \Psi(\theta), \Psi(\theta') \rangle_{\mathcal{H}} = \sum_{a=1}^{m} \psi_a(\theta) \psi_a(\theta') \leq \sqrt{\sum_{a=1}^{m} (\psi_a(\theta))^2} \sqrt{\sum_{a=1}^{m} (\psi_a(\theta'))^2} \tag{8.17}$$

where the inequality trivially follows using Cauchy-Schwarz. Now consider the case where m is uncountably infinite such that $\mathcal{H}$ represents a functional space along with an inner product. If $\Psi(\cdot)$ remains in L^2 space, i.e.

$$\sum_{a=1}^{\infty} (\psi_a(\theta))^2 < \infty, \quad \forall \theta \in \vartheta, \tag{8.18}$$

then $k(\theta, \theta') < \infty,\ \forall \theta, \theta' \in \vartheta$ using the inequality from equation (8.17), and our covariance will remain bounded with infinitely many basis functions.

The key realization is now that the distribution of F only depends on Z and C through the inner product $\langle \Psi(\theta), \Psi(\theta') \rangle_{\mathcal{H}}$ such that we need never compute the (potentially infinite)

[2] $Z(\theta_i)^T C Z(\theta_j) = 0$ is only possible if $\mathbf{w}$ lives in a lower dimensional subspace of $\mathbb{R}^m$

mapping $\Psi(\theta)$ if we can find a way to implicitly compute the inner product, for example if k has the *reproducing property*

$$\langle u(\cdot), k(\cdot,\theta)\rangle_{\mathcal{H}} = u(\theta), \quad \forall\theta\in\vartheta,\ \forall u(\cdot)\in\mathcal{H}. \tag{8.19}$$

This is called the kernel trick, where we have used the representation $\Psi(\theta) = k(\cdot,\theta)$ noting that any feature map can be thought of as parameterizing a mapping $\mathcal{H}\to\mathbb{R}$ through the inner product. We can now introduce the notion of *reproducing kernel Hilbert space* (RKHS) [Aronszajn, 1950].

Definition 8.3. *Given a Hilbert space $\mathcal{H}$, then a function $k\colon \vartheta\times\vartheta\to\mathbb{R}$ is a reproducing kernel and $\mathcal{H}$ is a reproducing kernel Hilbert space if $k(\cdot,\theta)\in\mathcal{H},\ \forall\theta\in\vartheta$ and the reproducing property described in equation* (8.19) *is satisfied.*

The significance of RKHSs for GPs is that their covariance functions correspond to reproducing kernels and so a GP is defined over an RKHS. Most RKHSs do not contain all possible functions – e.g. the RKHS associated with the squared exponential kernel contains only infinitely differentiable functions – and so the choice of GP kernel will dictate the range functions it is capable of encapsulating. Note that as $k(\theta,\theta') = \langle k(\cdot,\theta), k(\cdot,\theta')\rangle = k(\theta',\theta)$, all reproducing kernels are symmetric.

Going back to our GP derivation, then for the realization of $k(\theta,\theta')$ at a finite number of points to be a valid covariance matrix we further require it to be positive definite, i.e:

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\beta_i\beta_j k(\theta_i,\theta_j)\geq 0, \quad \forall n\geq 1,\ \forall\{\beta_1,\ldots,\beta_n\}\in\mathbb{R}^n,\ \forall\{\theta_1,\ldots,\theta_n\}\in\vartheta^n \tag{8.20}$$

which can be proved to be the case given the restrictions we have placed on k by noting

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\beta_i\beta_j k(\theta_i,\theta_j) = \sum_{i=1}^{n}\beta_i\Psi(\theta_i)^T\sum_{j=1}^{n}\beta_j\Psi(\theta_j) = \left\|\sum_{i=1}^{n}\beta_i\Psi(\theta_i)\right\|_{\mathcal{H}}^2 \geq 0. \tag{8.21}$$

Consequently, (8.18) is a sufficient condition for the distribution of F to be a multivariate $t-$dimensional Gaussian with a valid and finite covariance matrix when we consider an uncountable infinite number of basis functions. Furthermore, we note that many choices for Ψ will lead to closed-form analytic expressions of $\langle\Psi(\theta),\Psi(\theta')\rangle_{\mathcal{H}}$ without the need to ever calculate $\Psi(\theta)$ explicitly.

To show a simple example of this consider the one-dimensional case where $\theta\in\mathbb{R}$ and basis functions of the form $\psi_a = \sigma_w\exp\left(-\frac{(\theta-c_a)^2}{2\rho^2}\right)$ where $c_a\in[c_{\min},c_{\max}]$ represents the center of the basis function and ρ is a common length scale. Further specify that $\sigma_w^2 = \frac{\beta(c_{\max}-c_{\min})}{m}$, i.e.

that σ_w^2 is proportional to the number of functions per unit length, then we have

$$k(\theta, \theta') = \sum_{a=1}^{m} \frac{\beta(c_{\max} - c_{\min})}{m} \exp\left(-\frac{(\theta - c_a)^2}{2\rho^2}\right) \exp\left(-\frac{(\theta' - c_a)^2}{2\rho^2}\right). \tag{8.22}$$

If we assume that the basis functions are evenly space in $[c_{\min}, c_{\max}]$, then in the limit $m \to \infty$ we recover the integral

$$k(\theta, \theta') = \beta \int_{c_{\min}}^{c_{\max}} \exp\left(-\frac{(\theta - c)^2}{2\rho^2}\right) \exp\left(-\frac{(\theta' - c)^2}{2\rho^2}\right) dc. \tag{8.23}$$

Now if we further take $c_{\min} \to -\infty$ and $c_{\max} \to \infty$ then we have the analytic solution

$$k(\theta, \theta') = \sqrt{\pi}\rho\beta \exp\left(-\frac{(\theta - \theta')^2}{4\rho^2}\right). \tag{8.24}$$

which is the common squared exponential kernel in one dimension. Thus we have managed to analytically marginalize over all basis functions and are left with a simple expression that can be used to evaluate the covariance between any two given points that implicitly uses an uncountably infinite number of basis functions.

More generally, the Moore-Aronszajin theorem [Aronszajn, 1950] states that

Theorem 8.1. *Any symmetric, positive definite kernel $k\colon \vartheta \times \vartheta \to \mathbb{R}$ has a unique RKHS for which k is a reproducing kernel.*

In other words, if we define a kernel function that is symmetric $k(\theta, \theta') = k(\theta', \theta)$ and positive definite as per (8.20), then at least one corresponding feature map $\Psi\colon \vartheta \to \mathcal{H}$ must exist. Therefore a corresponding $Z\colon \vartheta \to \mathcal{H}$ and C must also exist (for example we can trivially take C as the identity matrix and $Z = \Psi$), and if we further ensure that k is bounded $k(\theta, \theta') < \infty,\ \forall \theta, \theta' \in \vartheta$, then finite realizations $[f(\theta_1), \ldots, f(\theta_t)]^T$ of equation (8.15) must be distributed to a finite multivariate Gaussian distribution with mean $[\mu(\theta_1), \ldots, \mu(\theta_t)]^T$ and covariance $k(\theta_i, \theta_j)\ \ \forall i, j \in 1, \ldots, t$.

We can, therefore, think of GP regression as Bayesian linear regression using a feature mapping to an RKHS. This shows the power of a GP – for appropriate choices of kernel it uses an uncountable number of features – but also highlights their assumptions: the co-Gaussianity of the weights and reliance of the target function to fall in RKHS represented by the covariance function.

8.3 Bayesian Optimization

Bayesian optimization (BO) is a global optimization scheme that requires only that the target function can be evaluated (noisily) at any given point [Mockus, 1975; Jones et al., 1998; Osborne

et al., 2009; Brochu et al., 2010; Shahriari et al., 2016b]. It does not require derivatives,[3] naturally incorporates noisy evaluations, and is typically highly efficient in the number of function evaluations. It is therefore suited to problems where the target function corresponds to the output of a simulator, estimation scheme, algorithm performance evaluation, or other cases where the target is not known in closed form. It remains a fast growing area of active research and has been successfully applied to a wide range of applications such as hyperparameter tuning [Snoek et al., 2012], robotics [Calandra et al., 2016], and sensor networks [Garnett et al., 2010].

The key idea of BO is to place a prior on f, most commonly a GP, that expresses beliefs about the space of functions within which f might live. When the function is evaluated, the resultant information is incorporated by conditioning upon the observed data to give a posterior over functions. This allows estimation of the expected value and uncertainty in $f(\theta)$ for all $\theta \in \vartheta$. From this, an acquisition function $\zeta : \vartheta \to \mathbb{R}$ is defined, which assigns an expected utility to evaluating f at particular θ, based on the trade-off between exploration and exploitation in finding the maximum. When direct evaluation of f is expensive, the acquisition function constitutes a cheaper to evaluate substitute, which is optimized to ascertain the next point at which the target function should be evaluated in a sequential fashion. By interleaving optimization of the acquisition function, evaluating f at the suggested point, and updating the surrogate, BO forms a global optimization algorithm that is typically very efficient in the required number of function evaluations, whilst naturally dealing with noise in the outputs.

Figure 8.3 shows BO being used to maximize the following simple one-dimensional function

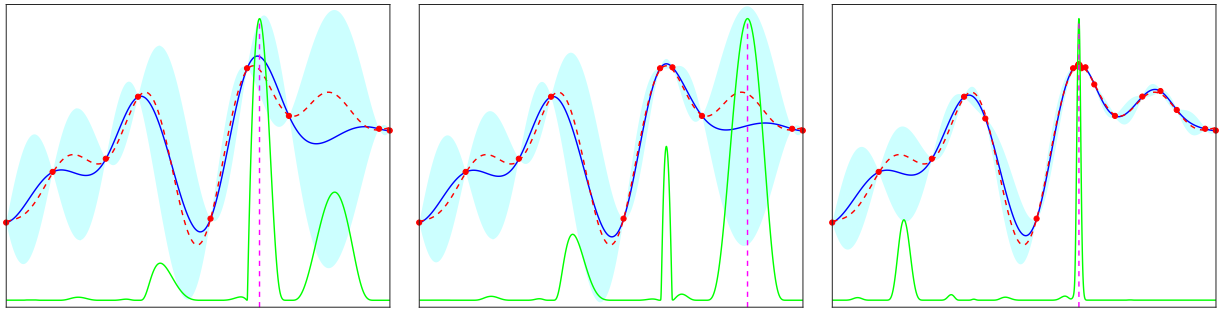
$$f(\theta) = \frac{\theta}{15} - \frac{\theta^2}{50} - \frac{\sin\theta}{\theta}, \quad -5 \leq \theta \leq 5 \tag{8.25}$$

with noisy observations $v \sim \mathcal{N}(f(\theta), 0.01^2)$. As we explained before, BO employs only pointwise function evaluations and so the optimization problem boils down to deciding

1. What input point should be evaluated at each iteration?
2. Where do we think the optimum is given our evaluations?

Both of these are done using a GP regressed to the existing evaluations, which therefore forms the first step of each BO iteration. As shown in Figure 8.3, this gives an estimated value and uncertainty for the function at every point in the form of the GP posterior mean $\mu_{\text{post}}(\theta)$ and

[3] Nonetheless, derivative measurements, if available, can still often be exploited to improve the BO procedure. For example, when using a GP surrogate, we can exploit the linearity of differentiation to update our surrogate with evaluations of the function derivative in an analogous manner as updating with function evaluations. See, for example, [Osborne, 2010, Section 4.7] for more details.



(a) 10 iterations **(b)** 11 Iterations **(c)** 20 Iterations

Figure 8.3: Illustration of using Bayesian optimization to optimize the toy one dimensional problem given in (8.25). Red dots show the function evaluations, the dotted red line shows the true function, the solid blue line is the GP mean, the shaded region is the GP mean ± 2 standard deviations, the green line is the acquisition function, and the dotted purple line is the location of the maximum of the acquisition function. See main text for further description.

marginal standard deviation $\sigma_{\text{post}}(\theta) = \sqrt{k_{\text{post}}(\theta, \theta)}$, where μ_{post} and k_{post} are as per (8.11). We will drop the $_{\text{post}}$ subscript in the rest of this chapter to avoid clutter. The second decision – deciding the location of the optimum – is now straight forward: our estimate for the most optimal point of those evaluated is simply the point with the highest mean, or if we allow our estimate to be a non-evaluated point, we can the instead take the maximum of the mean function.

To solve the first decision – where to evaluate next – we specify a so-called *acquisition function* $\zeta : \vartheta \to \mathbb{R}$ that encodes the relative utility of evaluating a new point. The optimum of this acquisition function is then the optimal point to evaluate at the next iteration. At a high level, the utility of a point is based on a trade-off between *exploration*, namely the desire to evaluate points where our uncertainty is high to reduce the uncertainty in those regions, and *exploitation*, namely the desire to evaluate points where the expected function value is high so that we get a good characterization of the function in promising regions. One could alternatively think about this as wanting to refine the estimates of promising local optima and sample in places that are likely to contain a missing mode. We will return to discuss specific acquisition functions in depth in the next section and for now we simply note that the most desirable points to evaluate will be those that provide both exploration and exploitation. For example, consider the expected improvement acquisition function (see (8.29)) after 10 iterations shown in green in Figure 8.3a. This is reasonably large in the two regions of highest uncertainty, but it is highest in the region near the true global optima where both the expected value and uncertainty are high. After 11 iterations, on the other hand, the uncertainty in this region has dropped dramatically and the point with the highest uncertainty now has the largest value for the acquisition function. Note

that the scales for the acquisition functions are different for the different iterations and that the acquisition function typically decreases as more points are observed.

At first, it might seem strange to replace our original optimization problem with another in the form of optimizing the acquisition function to choose the point to evaluate next. This new optimization problem is itself a global optimization of a typically highly multi-modal function and we need to carry it out at each iteration of the BO algorithm. Indeed if the target function f is cheap to evaluate, taking such an approach would be rather foolish. The key difference is that the surrogate optimization problem can be solved without the need to evaluate the original target function. Therefore, if f is expensive to evaluate, this additional computation can be justified if it keeps the number of required evaluations of f to a minimum. There are also a number of other features that mean the problem of optimizing the acquisition function is typically much easier than the original target – it can be evaluated exactly, derivatives are often available, and provided an appropriate kernel is used then it is guaranteed to be smooth. It is also not typically necessary to ensure that the acquisition function is optimized exactly, as evaluating a point that is sub-optimal under the acquisition function simply means that a point expected to be less helpful is evaluated at the next iteration. This can be useful when the time for a BO iteration is comparable to a function evaluation, as the amount of computational effort undertaken by the BO can be tuned to the problem at hand. It is also worth noting that the computational complexity for optimizing the acquisition function is theoretically less (in terms of BO iterations) than the GP training ($O(N^2)$ instead of $O(N^3)$), though the two are still often comparable for the modest number of iterations BO is generally run for.

Going back to Figure 8.3, we see that BO works by interleaving regressing a GP to the evaluations, optimizing the acquisition function to find the next point to evaluate, evaluating the target function at that point, and the updating the GP regression again. After 20 iterations for our simple problem, the GP regression around the optimum has become very accurate, while it is less accurate elsewhere. This highlights the sample efficiency of BO as it shows how computational resources are focused on where they are required.

Though we have thus far assumed that a GP is used for the surrogate, this is far from the only possible choice and one can, for example, use random forests [Bergstra et al., 2011; Hutter et al., 2011] or neural networks [Snoek et al., 2015]. There are a number of characteristics that make GPs suitable for the surrogate model. For example, they are very powerful regressors that can accurately represent the function from relatively few evaluations, especially for low-dimensional smooth functions. They also naturally produce uncertainty estimates that are typically more

accurate than those produced by alternatives which often have to resort to post-processing or heuristics to estimate uncertainty. This also leads to simple and tractable acquisition functions. Similarly, their ability to incorporate noisy observations is often helpful, though because GP inference is only analytic for Gaussian likelihoods, it is often impractical to incorporate non-Gaussian noise. This problem can be particularly manifest when the observations are bounded, such as when the target is a probability and thus always positive. A common way of dealing with this is to optimize a mapping of the original function, for example optimizing the log probability in (M)MAP problems [Osborne, 2010].

When prior information is known about the function, the flexibility in choosing the covariance kernel can also be very helpful, for example allowing known smoothness to be incorporated. However, this can also be a curse as an inappropriate choice of kernel can severely hamper the performance. In particular, it will typically be necessary to do inference over, or at least optimize, the GP hyperparameters. Another drawback to using GPs is poor scaling in the number of iterations – training is a $O(N^3)$ operation due to the required matrix inversion. This typically restricts BO using GPs to using on the order of hundreds of iterations unless appropriate approximations are made [Snelson and Ghahramani, 2006; Hensman et al., 2013]. Another drawback can be poor scaling in the dimensionality of the inputs – most common kernels only encode information about the inputs through the (scaled) Euclidean distances between points, which can break down in high dimensions because all points are typically "far away" from one another [Bengio et al., 2006].

8.3.1 Acquisition Functions

As we previously explained, the GP posterior provides convenient analytic representations for the expected value and uncertainty in the function in the form of the GP posterior mean $\mu(\theta)$ and marginal standard deviation $\sigma(\theta)$ respectively. We now introduce a number of common acquisition functions, many, but not all, of which will use these representations directly. The choice of acquisition function can be critical to the performance of Bayesian optimization and its choice forms a basis for a significant proportion of the Bayesian optimization literature [Shahriari et al., 2016b]. In particular, some acquisition functions have pathologies, for example probability of improvement has a tendency to insufficiently explore, which can severely impede the performance of BO. This has lead to the development of some very powerful, but computationally intensive and algorithmically complicated, acquisition strategies (e.g. Hernández-Lobato et al. [2014]), such that there is often a speed/simplicity vs per iteration performance trade-off in their selection.

8.3.1.1 Probability of Improvement

One of the conceptually simplest acquisition functions is the probability of improvement (PI, Kushner [1964]), namely the probability that the function value at a point is higher than the current estimated optimum. As shown by, for example, Brochu et al. [2010], the probability of improvement has a simple analytic form because the marginal distributions for function evaluations are Gaussian. Specifically, let

$$\mu^{+} = \max_{j \in \{1,\ldots,m\}} \mu\left(\theta_j\right) \tag{8.26}$$

denote the point with the highest expected value and define

$$\gamma\left(\theta\right) = \frac{\mu\left(\theta\right) - \mu^{+} - \xi}{\sigma\left(\theta\right)}, \tag{8.27}$$

where $\xi \geq 0$ is a user set parameter. The probability of improvement is then defined as the probability of improving the objective function by at least an amount ξ

$$\mathrm{PI}\left(\theta\right) = p\left(f\left(\theta\right) \geq \mu^{+} + \xi\right) = \Phi\left(\gamma\left(\theta\right)\right) \tag{8.28}$$

where $\Phi\left(\cdot\right)$ represents the unit normal cumulative distribution function. Note that $\xi = 0$ corresponds to pure exploitation and $\xi \to \infty$ corresponds to pure exploration. Typically for PI, one will employ a cooling schedule for ξ to incorporate the intuitive idea that one should first explore and then exploit. However, as shown by for example Jones [2001], the performance of the PI acquisition function is sensitive to the choice of ξ and therefore the cooling strategy requires careful tuning. Doing this effectively in a general purpose way is very challenging, meaning that PI is rarely used in practice.

8.3.1.2 Expected Improvement

A method less sensitive to tuning is the expected improvement [Mockus, 1975]. It is conceptually similar to PI, but incorporates the idea that large improvements are more beneficial to small improvements. It, therefore, better represents the importance of exploration and is less prone to over-exploitation. It can also be calculated analytically for a GP (see e.g. Brochu et al. [2010]) leading to the following definition

$$\begin{aligned}
\mathrm{EI}\left(\theta\right) &= \int_{\mu^{+}+\xi}^{\infty} p\left(f\left(\theta\right)|\mu\left(\theta\right), \sigma\left(\theta\right)\right)\left(f\left(\theta\right) - \mu^{+} - \xi\right) df\left(\theta\right) \\
&= \begin{cases} \left(\mu\left(\theta\right) - \mu^{+} - \xi\right)\Phi\left(\gamma\left(\theta\right)\right) + \sigma\left(\theta\right)\phi\left(\gamma\left(\theta\right)\right), & \sigma\left(\theta\right) > 0 \\ 0, & \sigma\left(\theta\right) = 0 \end{cases}
\end{aligned} \tag{8.29}$$

where $\phi\left(\cdot\right)$ represents the probability density function of the unit normal distribution. The parameter ξ plays a similar role as for the PI, but in this case $\xi = 0$ no longer corresponds to

pure exploitation. Lizotte [2008] suggests that cooling schedules on ξ are not helpful for EI and that $\xi = 0.01\mathbb{E}\left[\sigma\left(\theta\right)\right]$ where $\mathbb{E}\left[\sigma\left(\theta\right)\right]$ is signal standard deviation works well in almost all cases. Simply setting $\xi = 0$ is also a common choice, but it can be prone to doing insufficient exploration. Convergence rates for EI have been demonstrated by Bull [2011].

8.3.1.3 Upper Confidence Bounding

Another possible acquisition strategy is to maximize the upper confidence bound (UCB) for the function value [Lai and Robbins, 1985; Srinivas et al., 2009], namely

$$\mathrm{UCB}\left(\theta\right) = \mu\left(\theta\right) + \kappa\sigma\left(\theta\right) \tag{8.30}$$

where $\kappa \geq 0$ is again a parameter that will control the exploration/exploitation trade-off. This bound represents the point at which the probability the function is less than this value is $\Phi(\kappa)$. A particularly nice feature of the UCB acquisition function is that given certain conditions on ϑ and k, κ can be chosen in a manner such that acquisition has bounded cumulative regret with high probability, this is known as GP-UCB [Srinivas et al., 2009]. However, the need to (often adaptively) set κ can be a noticeable drawback.

8.3.1.4 Information-Based Policies

At the end of the day, the aim of BO is to find the location of the optimum. The true utility of evaluating a new point is, therefore, in the information it provides about the location of the true maximum θ^* and the assistance it provides in guiding future evaluations. Though the latter of these is difficult to actively incorporate, the former is something that can be targeted in a principled manner by considering the distribution on the location of the maximum $p(\theta^*|\Theta, V)$ where Θ and V are the previously evaluated input and output points as per Section 8.2.

The simplest way this can be done is using Thompson sampling [Thompson, 1933], where instead of optimizing an acquisition function directly, one looks to sample the next point to evaluate from $p(\theta^*|\Theta, V)$ [Shahriari et al., 2014; Bijl et al., 2016; Kandasamy et al., 2017]. In the GP setting, this is equivalent to sampling a particular function realization f and then taking θ_{next} as the optimum of this particular realization. This can prove surprisingly expensive, as sampling a joint realization of a GP at M points is an $O(M^3)$ operation.

A more complicated, but theoretically more powerful approach, is to try and directly optimize for the point that will, in expectation, most reduce the uncertainty about the location of the maximum. These so-called entropy search (ES) methods [Villemonteix et al., 2009; Hennig and Schuler, 2012; Hernández-Lobato et al., 2014] use as an acquisition function the expected

gain in Shannon information about the location of the optimum, or equivalently the reduction in entropy of the location of the maximum. Namely they take

$$\zeta(\theta) = H\left[p(\theta^*|\Theta, V, \theta)\right] - \mathbb{E}_{p(v|\Theta,V,\theta)}\left[H\left[p(\theta^*|\Theta, V, \theta, v)\right]\right] \tag{8.31}$$

where $H[p(x)] = -\int p(x)\log p(x)dx$ is the differential entropy of its argument and $p(v|\Theta, V, \theta)$ is the predictive distribution of the GP such that $v \sim \mathcal{N}\left(\mu(\theta), \sigma(\theta)^2 + \sigma_n^2\right)$ as explained in Section 8.2.1. Entropy search methods are closely linked to the idea of Bayesian experimental design, in which an equivalent target has been used for some time Chaloner and Verdinelli [1995]. We return to Bayesian experimental design in detail in Chapter 11.

Unfortunately, (8.31) is not analytically tractable and so entropy search methods must rely on methods for approximately solving (8.31). This can be particularly difficult as (8.31) represents a nested estimation problem that cannot be solved simply by, for example, Monte Carlo (see Chapter 10). An important advancement in this regard was the development of an estimation scheme by Hernández-Lobato et al. [2014] that uses a combination of analytic approximations and Monte Carlo sampling. Despite these approximations, the resulting predictive ES algorithm provides excellent empirical performance in terms of the number of function evaluations and remains arguably the state-of-the-art acquisition function for BO.

8.3.1.5 Other Acquisition Functions

Many more acquisition strategies have been developed than we have space to cover here. However, some of other strategies of particular note include

- Contal et al. [2014] use a combination of the GP mean and an approximation of the mutual information between the function and the noisy evaluations at the chosen input points, providing impressive theoretical and empirical results.
- Osborne et al. [2009] and González et al. [2016b] develop non-myopic methods that incorporate the effect of future hypothetical evaluations on the relative optimality of evaluating a point at the current iteration. Such lookahead methods can improve performance, but at the expense of substantially increased computational cost.
- Wang et al. [2016] introduce a strategy explicitly estimating, then using, the value of $f(\theta^*)$ and establish connections between their approach, UCB, and PI.
- Swersky et al. [2013]; Shah and Ghahramani [2016]; Hernández-Lobato et al. [2016a]; Feliot et al. [2017] introduce acquisition functions for dealing with multi-objective problems,

for which we wish to optimize many functions simultaneously, e.g. by approximating the so-called Pareto front, constituting the set of non-dominated solutions.

8.3.2 GP Hyperparameters

Generally, the performance of BO will be strongly dependent on the choice of GP hyperparameters α. Rather than optimizing for α it is natural, when tractable, to take a fully Bayesian view and marginalize over α [Osborne et al., 2009], leading to mixture of GPs posterior. This was shown by Snoek et al. [2012] to substantially improve the performance of BO. An easy way to do this for simple acquisition functions is to consider an integrated acquisition function corresponding to the expectation of the acquisition function over the GP hyperparameters [Snoek et al., 2012]

$$\bar{\zeta}(\theta) = \mathbb{E}_{p(\alpha|\Theta,V)}\left[\zeta(\theta;\alpha)\right] = \frac{1}{p(V|\Theta)}\mathbb{E}_{p(\alpha)}\left[\zeta(\theta;\alpha)\, p(V|\Theta,\alpha)\right] \tag{8.32}$$

where $p(\alpha)$ is a prior on the hyperparameters, $p(V|\Theta)$ is a constant that can be ignored as it does not affect the position of the optimum, and $p(V|\Theta,\alpha)$ is the GP marginal likelihood as per (8.12). The expectation can be approximated using Monte Carlo inference, with slice sampling [Murray and Adams, 2010] and Hamiltonian Monte Carlo [Hensman et al., 2015] being common choices.

8.3.3 Parallelization

In recent years there has been increased interest in performing Bayesian optimization in the setting where function evaluations can be performed in parallel [Contal et al., 2013; Desautels et al., 2014; González et al., 2016a; Kathuria et al., 2016]. Such methods usually predict a batch of evaluation points that are then evaluated simultaneously. It is desirable for the batch to both include points that have high acquisition functions values, but which are well separated to minimize the correlation between their outputs. Recent work by Kandasamy et al. [2017] has also looked at the case where evaluation times can vary greatly, such that is desirable to carry out evaluations asynchronously in parallel, rather than in batches.

8.3.4 Constraints

Bayesian optimization approaches mostly assume that the target function is constrained by a bounding box, i.e. that each input is independently constrained. In practice, this assumption is rarely fulfilled, though for technically unbounded problems then it is not uncommon for the user to be able to specify a box conveying a region where it is reasonable to assume the true optimum falls within the box. To go beyond this assumption, it is necessary to design strategies that can deal with constraints and that deal with unbounded problems. Dealing with the former when using a GP

surrogate can be particularly challenging, as simply setting the points that violate the constraints to have very low objective function evaluations will severely undermine the GP regression.

Known inequality constraints with simple closed forms can easily be incorporated into most BO methods by adapting the acquisition function to be arbitrarily bad when the constraints are violated [Gramacy and Lee, 2010]. In other words, the constraints can be incorporated by optimizing the acquisition function as a constrained optimization problem. If the constraints are not known a-priori and expensive to evaluate, then is typically also necessary to model the constraints as well as the target function. Gardner et al. [2014] introduced a method for carrying out BO in the scenario where the constraint function is evaluated concurrently to the target function. Their approach learns the constraint function in a similar way as the target function using a Gaussian process and they then adapt the EI acquisition function to incorporate this constraint information. In independently developed work, Gelbart et al. [2014] introduce a similar, but more general, approach where the constraints can be evaluated independently and the constraint evaluations may be noisy. Hernández-Lobato et al. [2016b] take this further by showing that the trade-off between improving the models of the target and constraints respectively can be dealt with naturally by ES based acquisition strategies.

A weakness of all these approaches is that they do not allow for dealing with equality constraints and will similarly perform poorly for constraint functions that are not well modeled by a GP. They also do not guarantee that the target function with only be evaluated at places where the constraints are satisfied. This can be inefficient as evaluations are wasted and makes the method inappropriate for scenarios where it is essential that the constraints are never violated. In the next Chapter, we show how probabilistic programming can be used to overcome both these problems, creating a BO strategy that automatically satisfies the problem constraints at every evaluation, even when these constraints are equality constraints. Our method also introduces an approach for dealing with unconstrained BO problems.

9

Automating Learning – Bayesian Optimization for Probabilistic Programs

So far in this thesis, and in the literature more generally, the focus has been on automating inference for probabilistic programs. However, as we explained in Section 8.1, optimization is also a very important tool in the machine learning arsenal. Further, many problems require coincident inference and optimization in the form of marginal maximum a posteriori (MMAP) estimation, maximum marginal likelihood (MML) estimation, or risk minimization. In this chapter, we develop a probabilistic programming framework for automating the solution of these mixed inference-optimization problems. We will focus for the most part on MMAP estimation, highlighting differences for MML estimation and risk minimization when necessary.

MMAP estimation is challenging as it corresponds to the optimization of an intractable integral, such that the optimization target is expensive to evaluate and gives noisy results. Current PPS inference engines are typically unsuited to such settings. We, therefore, introduce *BOPP*[1] (Bayesian optimization for probabilistic programs) [Rainforth et al., 2015, 2016b], which uses a series of code transformations, existing inference algorithms, and a bespoke Gaussian process (GP) based Bayesian optimization (BO) package, which we call *Deodorant*[2], to optimize the evidence of a query with respect to an arbitrary subset of its internally sampled variables.[3]

BOPP can be viewed from two different perspectives, providing distinct contributions to both the probabilistic programming and Bayesian optimization literatures. From a probabilistic programming perspective, we can view BOPP as extending PPSs beyond their typical inference setting to a more general mixed inference-optimization framework. It allows users to specify a model in the same manner as existing systems, but then select some subset of the sampled variables in the query to be optimized, with the rest marginalized out using existing inference algorithms. Though the exact code transformations we introduce are inevitably language specific and have been implemented for Anglican, the concepts we introduce apply more widely and

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

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

[3] There is no need to provide separate consideration for optimizing with respect to the inputs of the query as, if required, it is trivial to include variables internally to the program and then do MML estimation.

Deodorant, the BO package itself, is not PPL specific and is provided as a stand-alone package. More generally, the *optimization query* we introduce can be implemented and utilized in any PPL that supports an inference method returning a partition function estimate. This framework increases the scope of models that can be expressed in PPSs and gives additional flexibility in the outputs a user can request from a program.

From a BO perspective, we can view BOPP as the first package to directly exploit the source code of its target. This leads to a number of novel contributions to the BO literature, such as innovations in problem-independent hyperpriors, unbounded optimization, and implicit constraint satisfaction. The ability to do these is rooted in the ability provided by PPLs to manipulate the target source code and return artifacts suitable for carrying out certain tasks on a model. For example, we use a code transformation to produce a problem-specific optimizer for the acquisition function that ensures the implicit constraints specified by the query, including equality constraints, are automatically and exactly satisfied. As a consequence, BOPP provides significant novel features and performance improvements even when used simply as a conventional BO package.

9.1 Motivation

To demonstrate the functionality provided by BOPP, we consider an example application of engineering design. Engineering design relies extensively on simulations which typically have two things in common: the desire of the user to find a single best design and an uncertainty in the environment in which the designed component will live. Even when these simulations are deterministic, this is an approximation to a truly stochastic world. By expressing the utility of a particular design-environment combination using an approximate Bayesian computation (ABC) likelihood [Csilléry et al., 2010], one can pose this as a MMAP problem, optimizing the design while marginalizing out the environmental uncertainty.[4] Figure 9.1 illustrates how BOPP can be applied to engineering design, taking the example of optimizing the distribution of power between radiators in a house so as to homogenize the temperature, while marginalizing out possible weather conditions and subject to a total energy budget. The Anglican program shown in Figure 9.2 allows us to define a prior over the uncertain weather, while conditioning on the output of a deterministic simulator (here Energy2D [Xie, 2012] – a finite element package for heat transfer) using an ABC likelihood. BOPP now allows the required coincident inference and optimization to be carried out automatically, directly returning increasingly optimal configurations.

[4]For the reasons explained in Section 8.1, it will often be more convenient to use the risk minimization framework for engineering design because one generally wishes to place emphasis on rare catastrophic events.

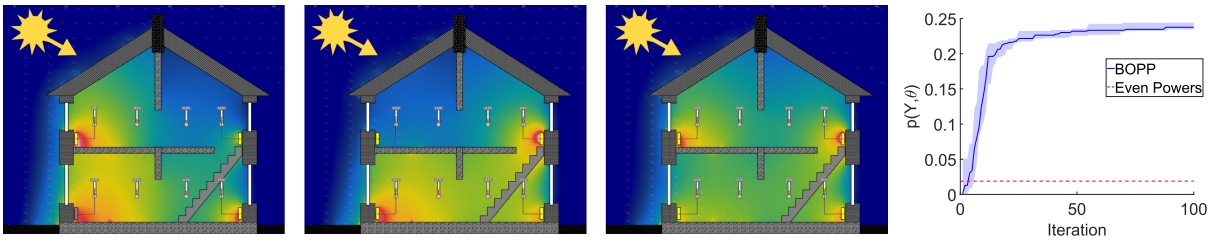


Figure 9.1: Simulation-based optimization of radiator powers subject to varying solar intensity. Shown are output heat maps from Energy2D [Xie, 2012] simulations at one intensity, corresponding from left to right to setting all the radiators to the same power, the best result from a set of randomly chosen powers, and the best setup found after 100 iterations of BOPP. The far right plot shows convergence of the evidence of the respective model, giving the median and 25/75% quartiles.

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

Figure 9.2: BOPP query for optimizing the power allocation to radiators in a house. Here `weather-prior` is a distribution over the solar intensity and a uniform Dirichlet prior with concentration `alpha` is placed over the powers. Calling **`simulate`** performs an Energy2D simulation of house temperatures. The utility of the resulting output is incorporated using **`abc-likelihood`**, which measures a discrepancy from the `target-temps`. Calling **`doopt`** on this query invokes BOPP to perform MMAP estimation, with the second input `powers` indicating the variable(s) to be optimized. Full details on the model and experiment are provided in Rainforth et al. [2017b].

BO is an attractive choice for the required optimization in MMAP as it is typically efficient in the number of target evaluations, operates on non-differentiable targets, and incorporates noise in the target function evaluations (see Section 8.3). However, applying BO to probabilistic programs presents challenges, such as the need to give robust performance on a wide range of problems with varying scaling and potentially unbounded support. Furthermore, the target program may contain unknown constraints, implicitly defined by the generative model, and variables whose type is unknown (i.e. they may be continuous or discrete). On the other hand, the availability of the target source code in a PPS presents opportunities to overcome these issues and go beyond what can be done with existing BO packages. BOPP exploits the source code in a number of ways, such as optimizing the acquisition function using the original generative model to ensure the solution satisfies the implicit constraints, performing adaptive domain scaling to ensure that GP kernel hyperparameters can be set according to problem-independent hyperpriors, and defining an adaptive non-stationary mean function to support unbounded BO.

Together, these innovations mean that BOPP can be run in a manner that is fully black-box from the user's perspective, requiring only the identification of the target variables relative to

current syntax for operating on arbitrary programs. We further show that BOPP is competitive with existing BO engines for direct optimization on common benchmarks problems that do not require marginalization.

9.2 Related Work

Reasonable consideration has been given before to solving maximum likelihood and marginal a posteriori (MAP) problems with PPSs and many systems provide some form of appropriate estimation scheme (see e.g. [Goodman and Stuhlmüller, 2014; Carpenter et al., 2015; Salvatier et al., 2016]). One simple approach is to apply an annealing to the `observe` densities (for maximum likelihood estimation) or both the `sample` and `observe` densities (for MAP estimation) in a conventional MCMC sampler such as LMH (see Section 7.4.1). This results in a simulated annealing algorithm [Aarts and Korst, 1988] for general purpose programs. An alternative approach, introduced by Tolpin and Wood [2015] instead uses ideas from Monte Carlo tree search to construct a general purpose MAP estimator. However, all of these approaches do not permit the more challenging scenario where the target is to optimize a marginal distribution. They are, therefore, not suitable for combined inference and optimization problems.

One approach that does consider optimization of marginal probabilities of probabilistic programs is given by van de Meent et al. [2016], which thus perhaps presents the closest approach to our own work. The aim of van de Meent et al. [2016] is, on the surface, somewhat different to our own as they look to automate policy search problems [Deisenroth et al., 2013] using probabilistic programs. However, policy search is a particular instance of MML estimation and so falls within our general problem class. Moreover, their approach of maximizing an evidence lower bound [Blei et al., 2016] using stochastic gradient ascent [Robbins and Monro, 1951] is, in principle, substantially more general than policy search problems and constitutes a general MML estimation scheme in its own right. However, the approach has a number of restrictions and assumptions. Perhaps the most critical for our purposes is that gradients are estimated using importance sampling and thus will generally become increasingly noisy as the dimensionality of the nuisance variables (i.e. those we wish to marginalize over) increases. The approach also requires mean field approximations to be made, the appropriateness of which will vary from problem to problem, while, as with other stochastic gradient ascent approaches, a large number of optimization iterations is typically required to reach convergence, which can be prohibitive for some problems. BOPP, on the other hand, requires very few assumptions to be made and is free to use more advanced inference approaches, for example SMC, for

estimating the marginal likelihood. One consequence of this is that it can scale to far higher dimensions in the nuisance variables. For example, our HMM model in Section 9.5 marginalizes over a roughly 5000-dimensional space.

Another interesting alternative approach, developed since the publication of BOPP, involves taking derivatives *through* an SMC sweep [Le et al., 2017c; Naesseth et al., 2017; Maddison et al., 2017]. More precisely, these methods allow the derivative of the marginal likelihood estimate (or more typically its logarithm) to be calculated during an SMC sweep, for example by using automatic differentiation on the calculation of the original estimate [Le et al., 2017c]. This can be used for MML or MMAP estimation of global parameters by using these gradients as input to a stochastic gradient ascent scheme. A key difference of this approach, to say van de Meent et al. [2016], is that using SMC instead of importance sampling means that substantially lower variance gradient estimates can be achieved. Though, to the best of our knowledge, no such approach is currently implemented in a PPS, doing so is in theory perfectly possible given a system supporting automatic differentiation. In Le et al. [2017c], we also show how this approach can be extended to perform simultaneous model learning and proposal adaptation and further to an amortized inference setting, whereby we learn an inference artifact that returns a proposal at run time. This means that, when the model of interest comprises of a deep neural network, then the method can be viewed as extending so-called auto-encoding methods [Kingma and Welling, 2014; Burda et al., 2015] from their limited importance sampling setting, to a more powerful SMC framework.

9.3 Problem Formulation

As we explained in Section 4.3.2, probabilistic program queries define unnormalized distributions $\gamma(x_{1:n_x}, \lambda)$ on program traces as defined in (4.3). At a high level, our aim is to optimize with respect to some of the $x_{1:n_x}$ variables while marginalizing out the rest, as per MMAP estimation, MML estimation, and risk minimization introduced in Section 8.1. However, formalizing what we mean by "some variables" is less trivial than it would first appear and will require specialist syntax for specifying models. To this end, we introduce a new query macro `defopt`. The syntax of `defopt` is identical to `defquery` except that it has an additional input identifying the variables to be optimized. As we will explain later, `defopt` invokes a series of code transformations to produce a number of different Anglican queries, each of which is compiled to a CPS-style Clojure function using `query` and that is then used by the BOPP back end.

A possible naïve strategy for identifying the target variables would be to simply predefine some subset of the `sample` indices $c \subset \mathbb{N}^+$ and optimize for $\{x_j\}_{j \in c}$. However, this is clearly

not satisfactory because it is generally an awkward method of identifying the target variables and the variables we wish to optimize may not always be sampled at the same position in our trace (i.e. we do not want superfluous **sample** statements to effect which variables constitute our target variables). Unfortunately though, the more natural choice of specifying any variable in the program, including those which are deterministic functions of other variables, is not possible in general, because of complications originating from changes of variables, namely that nonlinear deterministic mappings change the density function in ways that it might not be possible to track. Instead, we will still specify our target variables $\theta = \phi_{1:L}$ by name (i.e. using the lexical name of the variable as bound by a **let** block in the raw program code), but we will place some restrictions, enforced at run time (see Rainforth et al. [2017b]) to ensure validity of the model. First, each optimization variable ϕ_ℓ must be bound to a value directly by a **sample** statement to avoid change of variable complications. Second, in order for the optimization to be well defined, the program must be written in such a way that any possible execution trace binds each optimization variable ϕ_ℓ exactly once. By proxy, this also ensures that the number of variables to be optimized, L, remains fixed. Finally, although any ϕ_ℓ may be lexically multiply bound, it must have the same reference measure in all possible execution traces, because, for instance, if the reference measure of a ϕ_ℓ were to change between Lebesgue to counting, the notion of optimality would no longer admit a conventional interpretation. Note that we impose no restrictions on the latent variables which will be marginalized over.

From a developer's perspective, these minor restrictions mean that all **sample** statements are either associated with a particular target variable ϕ_ℓ or never associated with any target variable. Further, **sample** statements associated with the target variables will never be evaluated more than once (but might never be evaluated if there are multiple possible **sample** statements associated with a particular ϕ_ℓ) and the total number of **sample** statements associated with the target variables is always L. Therefore building on the notation from Section 4.3.2, we will redefine $x_{1:n_x}$ as the variables that are to be marginalized over, with all associated terms similarly redefined (except γ). We next denote the m_ℓ lexical **sample** statements associated with each ϕ_ℓ as $h_{\ell,1}, \ldots, h_{\ell,m_\ell}$ with associated density functions $h_{\ell,i}(\phi_\ell|\xi_\ell)$ where ξ_ℓ is the provided distribution object (which can be a random variable but its reference measure must be deterministic). We further denote $c_\ell \in \{1, \ldots, m_\ell\},\ \forall \ell \in \{1, \ldots, L\}$ as the (potentially random) variable used to index which lexical **sample** statement ϕ_ℓ is drawn from in a particular

execution. We now have that the conditional distribution on the trace $\mathcal{T}$ implied by the query is $p(\mathcal{T} = \{\phi_{1:L}, x_{1:n_x}\}|\lambda) \propto \gamma(\theta, x_{1:n_x}, \lambda)$ where

$$\gamma(\theta, x_{1:n_x}, \lambda) = \begin{cases} \prod_{\ell=1}^{L} h_{\ell,c_\ell}(\phi_\ell|\xi_\ell) \prod_{j=1}^{n_x} f_{a_j}(x_j|\eta_j) \prod_{k=1}^{n_y} g_{b_k}(y_k|\psi_k) & \text{if} \quad \mathcal{B}(\theta, x_{1:n_x}, \lambda) = 1 \\ 0 \quad \text{otherwise} \end{cases} \tag{9.1}$$

and we have redefined the trace validity function $\mathcal{B}(\phi_{1:L}, x_{1:n_x}, \lambda)$ appropriately. As before, many terms in our trace probability might be random variables, but all are deterministic functions of $\{\phi_{1:L}, x_{1:n_x}\}$. Note that the relative ordering of the $\phi_{1:L}$ to the $x_{1:n_x}$ does not affect the validity of the trace or the probability, as the **`sample`** statements associated with each are mutually exclusive.

We can now use (9.1) to define the MMAP estimate targeted by a **`defopt`** query as

$$\begin{aligned} \theta^*(\lambda) &= \arg\max_{\theta \in \vartheta(\lambda)} \mathbb{E}\left[p(\mathcal{T} = \{\phi_{1:L}, x_{1:n_x}\}|\lambda)|\theta\right] = \arg\max_{\theta \in \vartheta(\lambda)} \mathbb{E}\left[\gamma(\theta, x_{1:n_x}, \lambda)|\theta\right] \\ &= \arg\max_{\theta \in \vartheta(\lambda)} \int_{x_{1:n_x} \in \{X : \mathcal{B}(\theta, X, \lambda) = 1\}} \prod_{\ell=1}^{L} h_{\ell,c_\ell}(\phi_\ell|\xi_\ell) \prod_{j=1}^{n_x} f_{a_j}(x_j|\eta_j) \prod_{k=1}^{n_y} g_{b_k}(y_k|\psi_k) dx_{1:n_x} \end{aligned} \tag{9.2}$$

where $\vartheta(\lambda) := \{\theta : \{\exists x_{1:n_x} : B(\theta, x_{1:n_x}, \lambda) = 1\}\}$ is the support of θ given λ. The target for the optimization is effectively the partition function of the program that results from "clamping" θ to a particular value. Though, in theory, one can conduct GP-based BO for non-numerical inputs by defining an appropriate kernel, we will not consider this case here for simplicity. However, we will allow for inputs of unknown type (i.e. discrete or continuous) as well as unknown constraints. We can therefore alternatively express our MMAP problem as the following constrained optimization

$$\theta^*(\lambda) = \arg\max_{\theta \in \mathbb{R}^D} \mathbb{E}\left[\gamma(\theta, x_{1:n_x}, \lambda)|\theta\right] \quad \text{s.t. } \mathbb{I}(\exists x_{1:n_x} : B(\theta, x_{1:n_x}, \lambda) = 1) = 1 \tag{9.3}$$

where we have applied the constraint that there is at least one valid trace associated with θ. For example, in Figure 9.2 the use of a Dirichlet distribution imposes the constraint $\sum_{\ell=1}^{L} \phi_\ell = 1$.

For MML estimation, the $\prod_{\ell=1}^{L} h_{\ell,c_\ell}(\phi_\ell|\xi_\ell)$ term is removed from the target formulation, thereby removing the terms from the generative model on θ, analogously to removing the prior in conventional frameworks.[5] The generative model on θ is still important here, specifying the constraints on θ. Thus, for example, doing MML estimation for the model defined in Figure 9.2 would retain the constraint $\sum_{\ell=1}^{L} \phi_\ell = 1$. Almost all BO problems are constrained in some way: if nothing else, one almost always needs to define some sort of bounding box for GP based BO, which can be encoded using uniform distributions in BOPP. Therefore, in addition to providing the first MML estimation scheme for PPS, BOPP forms a highly convenient means of specifying,

[5] Because the y_k could still depend on θ, this definition does not necessarily always correspond to the conventional notion of an MML solution. However, this is a necessary design decision and it does not place any restrictions on the problems that can be encoded as one can always manually remove the relevant **`observe`** statements if desired.

and automating the solution for, conventional constrained BO problems. In particular, BOPP is, to the best of our knowledge, the first system to support *equality* constraints.

The only thing that needs to change from the MML definition for the risk minimization case is that the $\arg\max$ is replaced with an $\arg\min$. As we presume that λ is fixed, this is analogous to solving (8.6) with a fixed Y such that the expectation is only over X.

For simplicity and notational consistency with our original paper [Rainforth et al., 2016b], we will now drop the dependency on λ and switch to the standard graphical model notation given in (8.5) with Y representing data, X are variables marginalized over, and θ the target variables (e.g. we express the MMAP problem as $\theta^* = \arg\max_{\theta \in \vartheta} p(Y, \theta)$). We note though that is not always completely accurate as per Section 4.3.2 and (9.2).

To carry out the interleaving of inference and optimization required by BOPP, we introduce **`doopt`**, which, analogous to **`doquery`**, takes a compiled output from **`defopt`** and an estimation type to carry out (i.e. MMAP, MML, or risk minimization), and returns a lazy sequence $\{\hat{\theta}^*_m, \hat{\Omega}^*_m, \hat{u}^*_m\}_{m=1,\ldots}$ where $\hat{\Omega}^*_m \subseteq X$ are the program outputs associated with $\theta = \hat{\theta}^*_m$ and each $\hat{u}^*_m \in \mathbb{R}^+$ is an estimate of the corresponding log partition function $\log p(Y, \hat{\theta}^*_m)$ (see Section 9.4.2). The sequence is defined such that, at any time, $\hat{\theta}^*_m$ corresponds to the point expected to be most optimal of those evaluated so far (using the GP surrogate) and allows both inference and optimization to be carried out online.

9.4 Bayesian Program Optimization

On top of the syntax introduced in the previous section, there are five main components to BOPP:

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

Together these allow BOPP to perform online MMAP estimation for arbitrary programs in a manner that is black-box from the user's perspective – requiring only the definition of the target program in the same way as existing PPS and identifying which variables to optimize. The

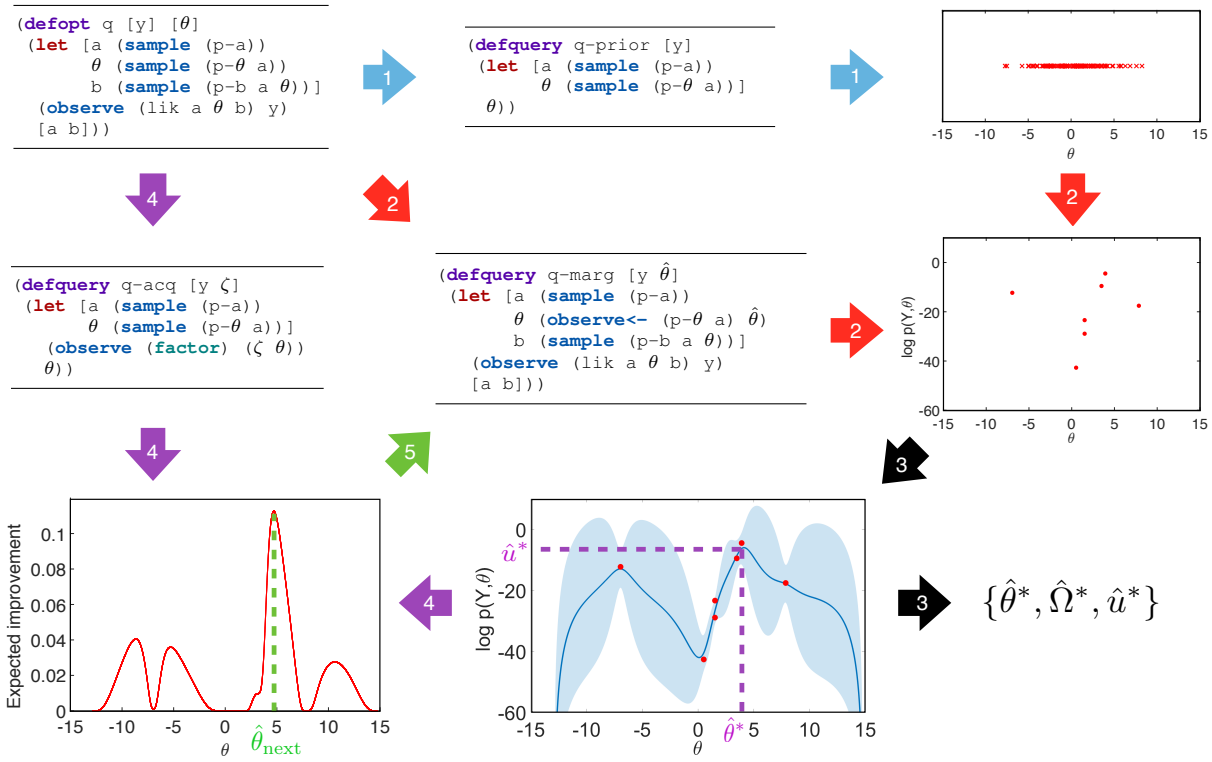


Figure 9.3: Overview of the BOPP algorithm, description given in main text. `p-a`, `p-θ`, `p-b` and `lik` all represent distribution object constructors. **`observe<-`** is identical to **`observe`** except it returns the observation. **`factor`** is a special distribution constructor that here factors the trace probability by $\zeta(\theta)$.

BO component of BOPP is both probabilistic programming and language independent, and is provided as a stand-alone package. It requires as input only a target function, a sampler to establish rough input scaling, and a problem specific optimizer for the acquisition function that imposes the problem constraints.

Figure 9.3 provides a high level overview of the algorithm invoked when **`doopt`** is called on a query `q` that defines a distribution $p\,(Y, a, \theta, b)$. We wish to optimize θ whilst marginalizing out a and b, as indicated by the the second input to `q`. In summary, BOPP performs iterative optimization in 5 steps

Step 1 (*blue arrows*) generates exact samples from the prior program `q-prior` (*top center*), constructed by removing all conditioning. This initializes the domain scaling for θ.

Step 2 (*red arrows*) evaluates the marginal $p(Y, \theta)$ at a small number of the generated $\hat{\theta}$ by performing inference on the marginal program `q-marg` (*middle center*), which returns samples from the distribution $p\,(a, b|Y, \theta)$ along with an estimate of $\log p(Y, \theta)$. The evaluated points (*middle right*) provide an initial domain scaling of the outputs and starting points for the BO surrogate.

Step 3 (*black arrow*) fits a mixture of GPs posterior to the scaled data (*bottom centre*) using a problem independent hyperprior. The solid blue line and shaded area show the posterior mean and ± 2 standard deviations respectively. The new estimate of the optimum $\hat{\theta}^*$ is the value for which the mean estimate is largest, with $\hat{u}^*$ equal to the corresponding mean.

Step 4 (*purple arrows*) constructs an acquisition function $\zeta : \vartheta \to \mathbb{R}^+$ (*bottom left*) using the GPs. This is optimized, giving the next point to evaluate $\hat{\theta}_{\text{next}}$, using simulated annealing on a transformed program `q-acq` (*middle left*) in which all **`observe`** statements are removed and replaced with a single **`observe`** adding a $\zeta(\theta)$ factor to the trace probability.

Step 5 (*green arrow*) evaluates $\hat{\theta}_{\text{next}}$ using `q-marg` and continues to step 3.

9.4.1 Program Transformation to Generate the Target

Consider the **`defopt`** query `q` in Figure 9.3, the body of which defines the joint distribution $p(Y, a, \theta, b)$. Calculating (8.5) (defining $X = \{a, b\}$) using a standard optimization scheme presents two issues: θ is a random variable within the program rather than something we control and its probability distribution is only defined conditioned on a.

We deal with both these issues simultaneously using a program transformation similar to the disintegration transformation in Hakaru [Zinkov and Shan, 2016]. Our *marginal* transformation can be thought of generating a new query, `q-marg` as shown in Figure 9.3, that defines the same unnormalized joint distribution on program variables and inputs (i.e. $\gamma(\theta, x_{1:n_x}, \lambda)$ is unchanged), but now accepts the value for θ as an input (i.e. the $\phi_{1:L}$ become terms in λ rather than being random variables). As such, the partition function of the program is now a function of θ and therefore can be optimized using an external algorithm. The transformation itself replaces all **`sample`** statements associated with each ϕ_ℓ with an equivalent **`observe<-`** statement, taking ϕ_ℓ as the observed value, where **`observe<-`** is identical to **`observe`** except that it returns the observed value instead of **`nil`**. As both **`sample`** and **`observe`** operate on the same variable type – a distribution object – this transformation can always be made, while the identical returns of **`sample`** and **`observe<-`** trivially ensures the validity of the transformed program. The transformation used for MML and risk minimization is equivalent except that the **`observe`** statements are replaced by an identity function (rather than **`observe<-`**), such that the transformation effectively removes the original **`sample`** statements.

In truth, the transformations used by BOPP are not exactly as shown in 9.3 and as described above. This is because, although for simple programs, such as the given example, these transformations can be easily expressed as static transformations, for more complicated programs

it would be difficult to actually implement these as purely static generic transformations in a higher-order language. Therefore, even though all the transformations dynamically execute as shown at runtime, in truth, the generated source code for the prior and acquisition transformations varies from what is shown and has been presented this way in the interest of exposition. Our true transformations exploit the existing Anglican special forms `store` and `retrieve` and two new special forms we introduce called `catch` and `throw`, in order to generate programs that dynamically execute in the same way at run time as the static examples shown, but whose actual source code varies significantly. Full details are given in Rainforth et al. [2017b].

9.4.2 Bayesian Optimization of the Marginal

The target function for our BO scheme is $\log p(Y, \theta)$, noting $\arg\max f(\theta) = \arg\max \log f(\theta)$ for any $f : \vartheta \rightarrow \mathbb{R}^+$. The log is taken because GPs have unbounded support, while $p(Y, \theta)$ is always positive, and because we expect variations over many orders of magnitude. PPS with importance sampling based inference engines, e.g. SMC or the particle cascade (see Section 7.4), can return noisy estimates of this target given the transformed program `q-marg`. Full details on our BO scheme can be found in Rainforth et al. [2017b], a summary of which is provided below.

Our BO scheme uses a GP prior and a Gaussian likelihood. Though the rationale for the latter is predominantly computational, giving an analytic posterior, there are also theoretical results suggesting that this choice is appropriate Bérard et al. [2014]. We use as a default covariance function a combination of a Matérn-3/2 and Matérn-5/2 kernel (see Section 8.2). By using automatic domain scaling as described in the next section, problem independent priors are placed over the GP hyperparameters such as the length scales and observation noise. Inference over hyperparameters is performed using Hamiltonian Monte Carlo (HMC) [Duane et al., 1987], giving an unweighted mixture of GPs. Each term in this mixture has an analytic distribution fully specified by its mean function $\mu_m^i : \vartheta \rightarrow \mathbb{R}$ and covariance function $k_m^i : \vartheta \times \vartheta \rightarrow \mathbb{R}$, where m indexes the BO iteration and i the hyperparameter sample.

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

et al. [2012], defining $\sigma_m^i(\theta) = \sqrt{k_m^i(\theta,\theta)}$ and $\gamma_m^i(\theta) = \frac{\mu_m^i(\theta)-\hat{u}_m^*}{\sigma_m^i(\theta)}$,

$$\zeta(\theta) = \sum_{i=1}^{N} \left(\mu_m^i(\theta) - \hat{u}_m^*\right) \Phi\left(\gamma_m^i(\theta)\right) + \sigma_m^i(\theta)\,\phi\left(\gamma_m^i(\theta)\right) \tag{9.4}$$

where ϕ and Φ represent the pdf and cdf of a unit normal distribution respectively. We note that more powerful, but more involved, acquisition functions, e.g. Hernández-Lobato et al. [2014], could be used instead.

9.4.3 Automatic and Adaptive Domain Scaling

Domain scaling, by mapping to a common space, is crucial for BOPP to operate in the required black-box fashion as it allows a general purpose and problem independent hyperprior to be placed on the GP hyperparameters. BOPP, therefore, employs an affine scaling to a $[-1,1]$ hypercube for both the inputs and outputs of the GPs. To initialize scaling for the input variables, we sample directly from the generative model defined by the program. This is achieved using a second transformed program, `q-prior`, which removes all conditioning, i.e. **`observe`** statements, and returns θ. This transformation also introduces code to terminate execution of the query once all θ are sampled, in order to avoid unnecessary computation. As **`observe`** statements return **`nil`**, this transformation trivially preserves the generative model of the program, but the probability of the execution changes. Specifically, if we denote n_θ as the number of non-target **`sample`** statements that have been invoked by the time all $\phi_{1:L}$ are sampled, then `q-prior` more formally defines the *unconditional* distribution $p_\lambda(\mathcal{T} = \{\phi_{1:L}, x_{1:n_\theta}\}) \propto \gamma_{\text{prior}}(\theta, x_{1:n_\theta}, \lambda)$ where

$$\gamma_{\text{prior}}(\theta, x_{1:n_\theta}, \lambda) = \begin{cases} \prod_{\ell=1}^{L} h_{\ell,c_\ell}(\phi_\ell|\xi_\ell) \prod_{j=1}^{n_\theta} f_{a_j}(x_j|\eta_j) & \text{if} \quad \mathcal{B}(\theta, x_{1:n_\theta}, \lambda) = 1 \\ 0 \quad \text{otherwise} \end{cases} \tag{9.5}$$

and the trace validity function $\mathcal{B}(\theta, x_{1:n_\theta}, \lambda)$ is redefined appropriately. Because (9.5) is an unconditional distribution, it can be sampled from directly by running the program forwards, returning exact samples from the corresponding marginal distribution on θ. This is computationally inexpensive, as it does not require inference or calling potentially expensive likelihood functions. It can thus be cheaply sampled from a number of times to initialize the input scaling. By further running inference on `q-marg` given a small number of these samples as arguments, a rough initial characterization of output scaling can also be achieved.

If points are later observed that fall outside the hypercube under this initial scaling, the domain scaling is appropriately updated so that the target for the GP remains the $[-1,1]$ hypercube. An important exception to this is that the output mapping to the bottom of the hypercube remains fixed and any points with partition function estimates lower than this are not incorporated into

the scaling in any way, i.e. the input scaling is not updated to incorporate these points either. For MMAP estimation, this ensures stability for unbounded problems as there can only be a finite region of the input space where the true value of the partition function is above any given value because its integral over θ must be finite. Similarly, the maximum possible estimate the inference algorithm might return will be bounded given some weak assumptions (roughly that $p(Y, X, \theta)$ is itself bounded). Consequently, the fixed base of the hypercube ensures that the there is a maximum possible size the hypercube can reach. For risk minimization (where our target is $-\log p(Y|\theta)$) and MML estimation (where our target is $\log p(Y|\theta)$) problems then we have no such guarantee that the adaptation will eventually cease. However, this is somewhat inherent to unbounded global optimization problems, rather than being a specific issue of BOPP.

9.4.4 Unbounded Bayesian Optimization via Mean Function Adaptation

Unlike standard BO implementations, BOPP is not provided with external constraints and we, therefore, develop a scheme for operating on targets with potentially unbounded support. For MMAP estimation, the target function is an unnormalized density, implying that the area that must be searched in practice to find the optimum is finite. For MML estimation and risk minimization this assumption is still reasonable in practice as if it is not true, we are effectively doomed to fail anyway. We, therefore, exploit this assumption by defining a non-stationary prior mean function. This takes the form of a bump function that is constant within a region of interest, but decays rapidly outside. Specifically, we define this bump function in the transformed space as

$$\mu_{\text{prior}}\left(r; r_e, r_\infty\right) = \begin{cases} 0 & \text{if } r \leq r_{\text{e}} \\ \log\left(\frac{r-r_{\text{e}}}{r_\infty - r_{\text{e}}}\right) + \frac{r-r_{\text{e}}}{r_\infty - r_{\text{e}}} & \text{otherwise} \end{cases} \tag{9.6}$$

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

An important consequence of this approach is that BOPP is not always an entirely global optimizer as the adaptation can, at least in theory, become stuck around a single mode if there

is extreme prior-target mismatch. Specifically, because "bad" points are not incorporated into the rescaling as described in the last section, we might have a "bad" region blocking expansion to another mode. In practice, such occurrences should be extremely rare (at least for MMAP estimation) as the initial scaling is approximately set to the region where the generative model has significant density, such that the problem would need to be both multi-modal and have extreme prior-target mismatch for BOPP to get stuck. One could, in theory, refine our method to provide better guarantees against such occurrences, but given the inherent difficulty of such problems and the fact that BOPP, like other GP-based BO methods, is heavily restricted in the number of iterations before the GP training cost becomes prohibitive (usually in the hundreds of iterations), doing so seems more likely to do harm than good in practice.

9.4.5 Optimizing the Acquisition Function

As we previously alluded to, BOPP presents the issue that the query contains implicit constraints that are unknown to the surrogate function. As we discussed in Section 8.3.4, constraints have previously been considered in the BO literature using a second surrogate model or an appropriate adaptation to the existing surrogate. Along with the potentially significant expense such a method incurs, this approach is inappropriate for *equality* constraints or when the target variables are potentially discrete. We, therefore, take an alternative approach based on directly using the program to optimize the acquisition function. To do so we consider a transformed program `q-acq` that is identical to `q-prior` (see Section 9.4.3), but adds an additional **`observe`** statement that assigns a weight $\zeta(\theta)$ to the execution such that the unnormalized joint distribution is now

$$\gamma_{\text{acq}}(\theta, x_{1:n_\theta}, \lambda) = \gamma_{\text{prior}}(\theta, x_{1:n_\theta}, \lambda)\zeta(\theta) \tag{9.7}$$

By setting $\zeta(\theta)$ to the acquisition function, the maximum likelihood solution of $\gamma_{\text{acq}}(\theta, x_{1:n_\theta}, \lambda)$ corresponds to the optimum of the acquisition function subject to the implicit program constraints. More precisely, we maximize $\zeta(\theta)$ subject to the constraint that

$$\theta \in \vartheta(\lambda) := \{\theta : \{\exists x_{1:n_\theta} : \gamma_{\text{prior}}(\theta, x_{1:n_\theta}, \lambda) > 0\}\}.$$

Such an estimate is achieved by running on `q-acq` a variant of the simulated annealing algorithm discussed in Section 9.2 in which RMH (see Section 7.4.1) is used for the transition kernel. Using an RMH, rather than LMH, transition kernel here is important as it will often be the case that the optimum of the transition function is in a position of very low prior mass (see e.g. Figure 9.4), such that LMH could take an unreasonably long time to propose an appropriate set of parameters.

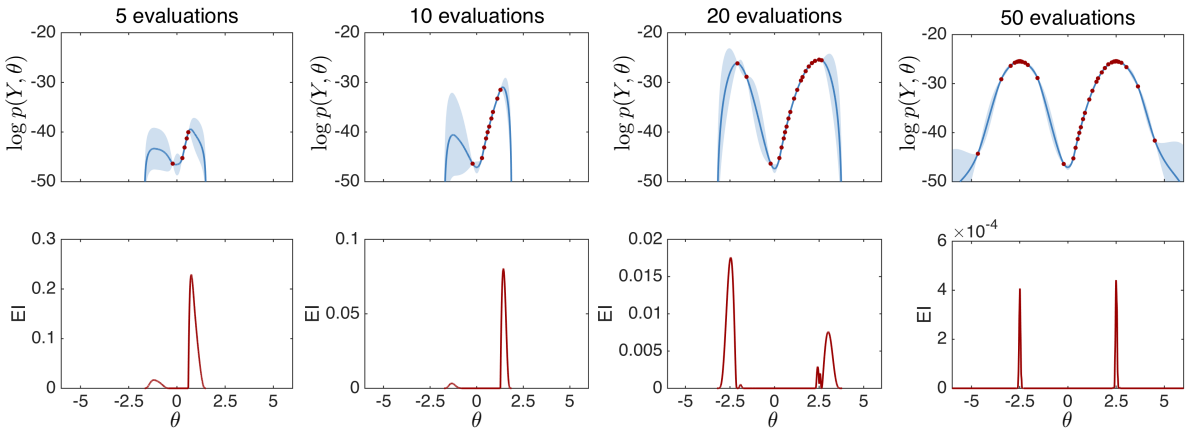


Figure 9.4: Convergence on an unconstrained bimodal problem with $p(\theta) = \text{Normal}(0, 0.5)$ and $p(Y|\theta) = \text{Normal}(5 - |\theta|, 0.5)$ giving significant prior misspecification. The top plots show a regressed GP, with the solid line corresponding to the mean and the shading shows $\pm$ 2 standard deviations. The bottom plots show the corresponding acquisition functions.

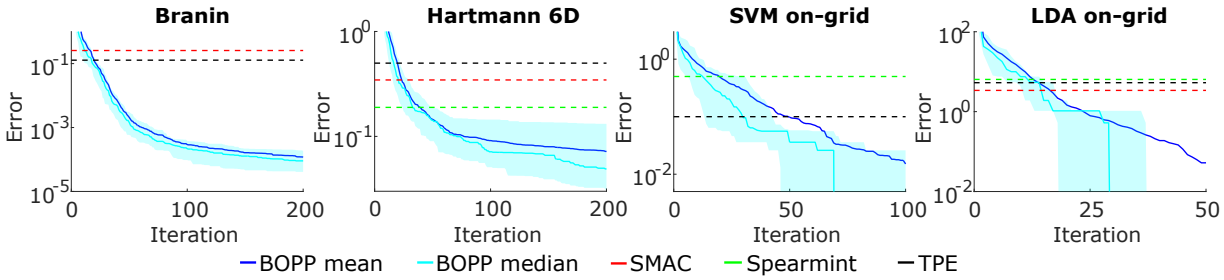


Figure 9.5: Comparison of BOPP used as an optimizer to prominent BO packages on common benchmark problems. The dashed lines shows the final mean error of SMAC (red), Spearmint (green) and TPE (black) as quoted by Eggensperger et al. [2013]. The dark blue line shows the mean error for BOPP averaged over 100 runs, whilst the median and 25/75% percentiles are shown in cyan. Results for Spearmint on Branin and SMAC on SVM on-grid are omitted because both BOPP and the respective algorithms averaged zero error to the provided number of significant figures in Eggensperger et al. [2013].

9.5 Experiments

We first demonstrate the ability of BOPP to carry out unbounded optimization using a 1D problem with a significant prior-posterior mismatch as shown in Figure 9.4. It shows BOPP adapting to the target and effectively establishing a maxima in the presence of multiple modes. After 20 evaluations the acquisitions begin to explore the right mode, after 50 both modes have been fully uncovered.

9.5.1 Classic Optimization Benchmarks

Next we compare BOPP to the prominent BO packages SMAC Hutter et al. [2011], Spearmint Snoek et al. [2012] and TPE Bergstra et al. [2011] on a number of classical benchmarks as shown in Figure 9.5. These results demonstrate that BOPP provides substantial advantages over these systems when used simply as an optimizer on both continuous and discrete optimization problems.

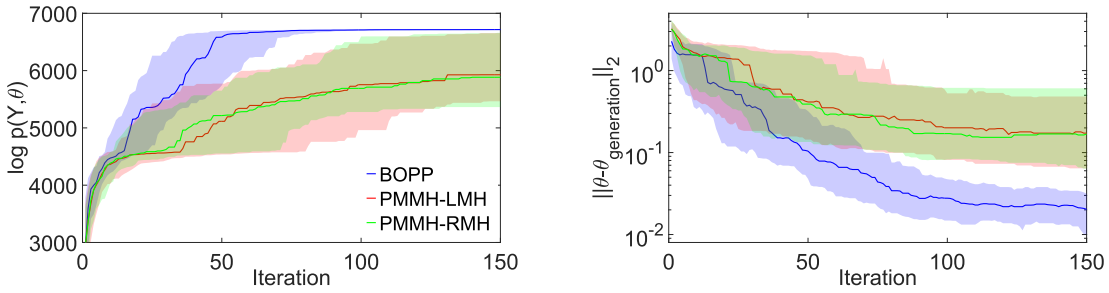
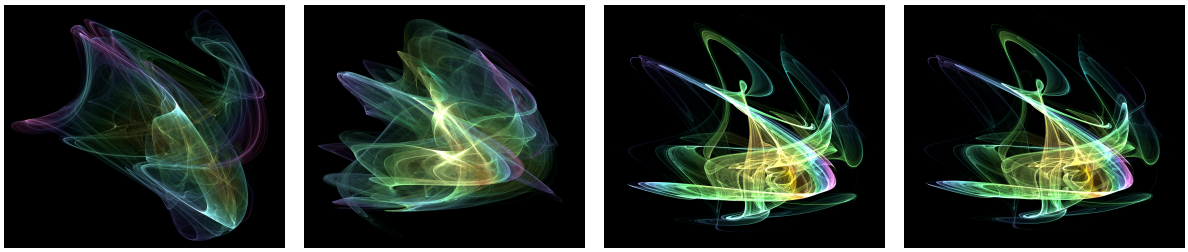


Figure 9.6: Convergence for transition dynamics parameters of the pickover attractor in terms of the cumulative best $\log p(Y, \theta)$ (*left*) and distance to the "true" θ used in generating the data (*right*). Solid line shows median over 100 runs, whilst the shaded region the 25/75% quantiles.



(a) 1 iteration **(b)** 20 iterations **(c)** 100 iterations **(d)** Ground truth

Figure 9.7: A series of trajectories for different parameters, demonstrating convergence to the true attractor. The colormap is based on the speed and curvature of the trajectory, with rendering done using the program Chaoscope (available at http://www.chaoscope.org/).

In particular, it offers a large advantage over SMAC and TPE on the continuous problems (Branin and Hartmann), due to using a more powerful surrogate, and over Spearmint on the others due to not needing to make approximations to deal with discrete problems.

9.5.2 Marginal Maximum a Posteriori Estimation Problems

We now demonstrate application of BOPP on a number of MMAP problems. Comparisons here are more difficult due to the dearth of existing alternatives for PPS. In particular, simply running inference on the original query does not return estimates for $p(Y, \theta)$. We consider the possible alternative of using our conditional code transformation to design a particle marginal Metropolis Hastings (PMMH, see Section 6.2.4) sampler which operates in a similar fashion to BOPP except that new θ are chosen using either an LMH or RMH step instead of actively sampling with BO. For these experiments we will only provide summaries here and refer the reader to Rainforth et al. [2017b] for full details and an additional example.

9.5.2.1 Extended Kalman Filter for the Pickover Chaotic Attractor

Our first MMAP example considers the case of learning the dynamics parameters of a chaotic attractor. Specifically, we will aim to optimize the parameters of a Kalman smoother, where

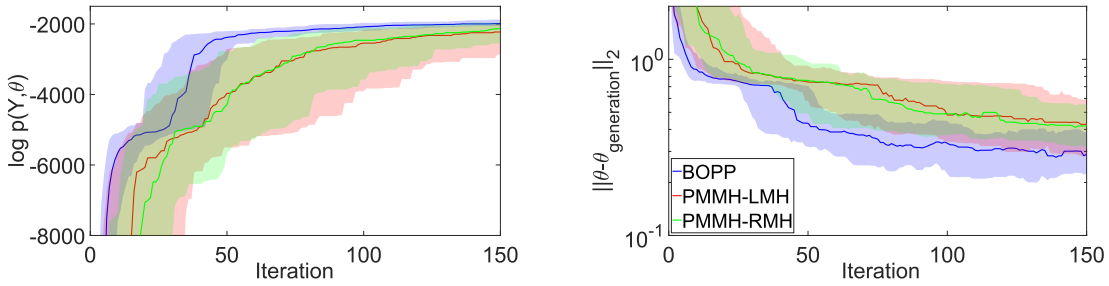


Figure 9.8: Convergence for HMM in terms of the cumulative best $\log p(Y, \theta)$ (*left*) and distance to the "true" θ used in generating the data (*right*). Solid line shows median over 100 runs, whilst the shaded region the 25/75% quantiles. Note that for the distance to true θ was calculated by selecting the three states (out of the 5 generated) that were closest to the true parameters.

we observe a noisy signal $y_t \in \mathbb{R}^K,\ t = 1, 2, \ldots, T$ in some K dimensional observation space and each observation has a lower dimensional latent parameter $x_t \in \mathbb{R}^D,\ t = 1, 2, \ldots, T$. In addition to an initial distribution $x_1 \sim \mathcal{N}(\mu_1, \sigma_1 I)$, our model is specified by

$$x_t = A(x_{t-1}, \theta) + \delta_{t-1}, \qquad \delta_{t-1} \sim \mathcal{N}(0, \sigma_q I) \tag{9.8a}$$

$$y_t = C x_t + \varepsilon_t, \qquad \varepsilon_t \sim \mathcal{N}(0, \sigma_y I) \tag{9.8b}$$

where I is the identity matrix, C is a known $K \times D$ matrix, and $\mu_1, \sigma_1, \sigma_q$ and σ_y are all known scalars. Our aim is to learn the dynamics parameters θ given $y_{1:T}$, marginalizing over the latent variables $x_{1:T}$. We use a uniform prior on the parameters θ, which means that the MMAP values for θ coincide with their (constrained) MML values. A synthetic dataset was generated with $T = 500$ and $K = 20$ (see Rainforth et al. [2017b]). Inference on `q-marg` was carried out using SMC with 500 particles. Convergence results are given in Figure 9.6 showing that BOPP comfortably outperforms the PMMH variants, while Figure 9.7 shows the simulated attractors generated from the dynamics parameters output by various iterations of a particular run of BOPP.

9.5.2.2 Hidden Markov Model with Unknown Number of States

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

Our model places a uniform prior on the number of states $K \sim \textsc{Discrete}\{1, 2, 3, 4, 5\}$. Each emission distribution corresponds to a Gaussian with unknown mean (which we wish

to optimize) and known variance. We marginalize over the transition distribution parameters and the HMM latent states. Rather than constructing a model where the emission distribution parameters only exist conditioned on the value of K, we generate variables for all 5 emission distributions that might be required, with only the first K actually then used. A synthetic dataset was generated using $K = 3$ and $T = 1000$ observations. Results of running BOPP are given in Figure 9.8, again showing that BOPP outperforms these PMMH alternatives. See Rainforth et al. [2017b] for further details.

9.6 Discussion

We have introduced a new method for carrying out MMAP estimation, MML estimation, and risk minimization of probabilistic program variables using Bayesian optimization, representing the first unified framework for optimization and inference of probabilistic programs. By using a series of code transformations, our method allows an arbitrary program to be optimized with respect to an arbitrary defined subset of its sampled variables, whilst marginalizing out the rest. To carry out the required optimization, we have introduced a new GP-based BO package that exploits the availability of the target source code to provide a number of novel features, such as automatic domain scaling and constraint satisfaction. The concepts we introduce lead directly to a number of extensions of interest, including but not restricted to smart initialization of inference algorithms, adaptive proposals, and nested optimization.

The main drawbacks of BOPP are inherited from GP-based BO more generally, most notably poor performance scaling in the dimensionality of the target variables θ and poor computational scaling in the number of iterations used (see Section 8.3). The former is arguably an almost inevitable consequence of the inherent difficultly of global optimization, though is perhaps more pronounced for GP-based BO than alternatives. It can be alleviated by making alternative assumptions, for example using a local optimization method (e.g. using stochastic gradient ascent), the suitability of which will be problem dependent. The latter drawback can be substantially alleviated by using a more lightweight surrogate regressor, e.g a random forest [Hutter et al., 2011], at the expense of typically reducing per-sample performance. From a practical usage perspective, there would be clear utility of such a system as an alternative or addition to BOPP, e.g. by automatically making approximations to the GP once a certain number of iterations has passed.

10
Nested Estimation

The convergence of Monte Carlo estimators has been considered extensively in the literature (see Chapter 5). However, the theoretical implications arising from the *nesting* of Monte Carlo estimators, where terms in the integrand depend on the result of separate, nested, Monte Carlo estimators, is generally less well known. This chapter examines the convergence of such nested Monte Carlo (NMC) methods [Rainforth et al., 2016a, 2017a]. We demonstrate that NMC can provide consistent estimates of nested expectations, including cases of repeated nesting, under mild conditions; establish corresponding rates of convergence; and provide empirical evidence that suggests these rates are observed in practice. We further establish a number of pitfalls that can arise from naïve nesting of Monte Carlo estimators and provide guidelines about how they can be avoided. Our results show that whenever an outer estimator depends nonlinearly on an inner estimator, then the number of samples used in *both* the inner and outer estimators must, in general, be driven to infinity for convergence. In other words, an infinite number of samples need to be used for each term in the outer estimator.

A key motivation behind this work is the ability of some PPSs to allow arbitrary nesting of models [Mantadelis and Janssens, 2011; Stuhlmüller and Goodman, 2014] (for example using the `conditional` construct in Anglican) such that it is easy to define and run nested inference problems, such as done by [Ouyang et al., 2016; Le et al., 2016]. These nested inference problems fall outside the scope of the conventional proof of the convergence of Monte Carlo estimation. Thus additional work is required to prove the validity of the corresponding back-end inference engines. However, the prevalence of NMC goes far beyond PPSs and so we will introduce it in a more general context, before returning to assess the implications for PPSs in Section 10.7.

We note that NMC should not be confused with a number of similarly named methods such as nested sampling [Skilling, 2004], nested SMC [Naesseth et al., 2015], or multilevel Monte Carlo [Heinrich, 2001], none of which target explicitly nested estimation problems. Of these, only nested SMC "nests" Monte Carlo methods as per our definition, but, as we will show later, it is based on one of our special cases where the problem can be rearranged to a single, non-nested,

estimation problem. *Nested simulation*, on the other hand, is an alternative term for NMC used in, for example, the financial statistics literature [Gordy and Juneja, 2010].

10.1 Background

There are various problems involving nested expectations that require the use of nested estimation schemes such as NMC. For example, the expected information gain used in Bayesian experimental design requires the calculation of an entropy of a marginal distribution (see Chapter 11), and therefore includes the expectation of the logarithm of an expectation. By extension, any Kullback-Leibler divergence where one of the terms is a marginal distribution also involves a nested expectation. Hence, our results have important implications for relaxing mean field assumptions in variational inference [Hoffman and Blei, 2015; Naesseth et al., 2017; Maddison et al., 2017] and deep generative models [Burda et al., 2015; Maaløe et al., 2016; Le et al., 2017c]. Another common nested estimation scenario is calculating expectations with respect to so-called doubly-intractable distributions [Møller et al., 2006; Murray et al., 2006; Liang, 2010], whereby the target distribution is only known up to a parameter-dependent normalizing constant. Here the normalization constant itself represents an intractable expectation nested within the overall expectation. NMC can also arise in contexts demanding the use of direct Monte Carlo simulations, for example in portfolio risk management [Gordy and Juneja, 2010] and stochastic control [Belomestny et al., 2010]. In particular, simulations of agents that reason about decisions of other agents or which reason about sequential decision making processes tend to include nested expectations. Thus problems like planning in games and "theory-of-mind" type meta-reasoning tend to require some form of nested estimation [Stuhlmüller and Goodman, 2014; Evans et al., 2017].

Certain nested estimation problems can be tackled by so-called pseudo-marginal methods [Beaumont, 2003; Andrieu and Roberts, 2009; Andrieu et al., 2010, 2015; Naesseth et al., 2015]. These consider cases of Bayesian inference where the likelihood is intractable, but can be estimated unbiasedly. From a theoretical perspective, they involve reformulating the problem in an extended space with auxiliary variables that are used to represent the stochasticity in the likelihood computation. This then enables the problem to be expressed as a single expectation. Our work goes beyond this by also considering cases in which a non-linear mapping is applied to the output of the inner expectation (such as the logarithm in the experimental design case or the inverse for general doubly-intractable distributions), so that this reformulation to a single expectation is no longer possible.

NMC with non-linear mappings of the inner expectation has been previously considered in the financial statistics literature, for example in the pricing of American options [Longstaff and Schwartz, 2001]. Though most of this literature focuses on particular application-specific non-linear mappings [Broadie et al., 2011; Gordy and Juneja, 2010], convergence bounds for a more general class of models has been shown by Hong and Juneja [2009]. We build on these results and outline the opportunities and pitfalls of nesting Monte Carlo estimators in a machine learning context. Our proofs apply to a more general class of problems than existing approaches. For example, we provide the first convergence bounds for cases of multiple levels of estimator nesting as might occur in a probabilistic programming system. We further lay out novel methods for reformulating certain classes of nested expectation problems into a single expectation, allowing the use of conventional Monte Carlo estimation schemes with superior convergence rates than the naïve use of NMC, one of which we exploit in Chapter 11 to derive an improved estimator for discrete Bayesian experimental design problems. Meanwhile, we provide theoretical results showing that any Monte Carlo estimator using imperfect nested estimates is, in general, biased. Finally, we provide numerical experiments, corroborating our theoretical results.

10.2 Problem Formulation

To recap from Chapter 5, the key idea of Monte Carlo is that the expectation of an arbitrary function $\lambda\colon \mathcal{Y} \to \mathcal{F} \subseteq \mathbb{R}$ under a probability distribution $p(y)$ for its input $y \in \mathcal{Y}$ can be approximately calculated using:

$$I = \mathbb{E}_{p(y)}\left[\lambda(y)\right] \approx \frac{1}{N}\sum_{n=1}^{N} \lambda(y_n) \quad \text{where} \quad y_n \overset{i.i.d.}{\sim} p(y). \tag{10.1}$$

In this chapter, we consider the case that λ is itself intractable, defined only in terms of a functional mapping of an expectation. Specifically, $\lambda(y) = f(y, \gamma(y))$ where we can evaluate $f\colon \mathcal{Y} \times \Phi \to \mathcal{F}$ exactly for a given y and $\gamma(y)$, but $\gamma(y)$ is the output of the following intractable expectation of another variable $z \in \mathcal{Z}$:

$$\text{either} \quad \gamma(y) = \mathbb{E}_{p(z|y)}\left[\phi(y, z)\right] \tag{10.2a}$$

$$\text{or} \quad \gamma(y) = \mathbb{E}_{p(z)}\left[\phi(y, z)\right] \tag{10.2b}$$

depending on the problem, with $\phi\colon \mathcal{Y} \times \mathcal{Z} \to \Phi \subseteq \mathbb{R}^{D_\phi}$. All our results apply to both cases, but we will focus on (10.2a) for clarity. Estimating I involves computing two integrals, one for y and the other for z. We refer to the approach of tackling both integrations using Monte

Carlo as *nested Monte Carlo* (NMC):

$$I \approx I_{N,M} = \frac{1}{N}\sum_{n=1}^{N} f(y_n, (\hat{\gamma}_M)_n) \qquad \text{where} \quad y_n \overset{i.i.d.}{\sim} p(y) \quad \text{and} \tag{10.3a}$$

$$(\hat{\gamma}_M)_n = \frac{1}{M}\sum_{m=1}^{M} \phi(y_n, z_{n,m}) \qquad \text{where each} \quad z_{n,m} \sim p(z|y_n) \quad \text{independently} . \tag{10.3b}$$

In Section 10.4 we will build on this further by considering cases with multiple levels of nesting, where computing $\phi(y, z)$ requires the computation of an intractable (nested) expectation.

10.3 Special Cases

We begin by discussing some special cases where it is possible to achieve a convergence rate of $O(1/N)$ in the mean square error (MSE) as per conventional Monte Carlo estimation [Robert, 2004]. Establishing these cases is important because it identifies what problems we can use existing results for, when we can achieve an improved convergence rate, and what precautions we must take to ensure this. The first case is that the top-level integrand f in our estimation problem is linear. Section 10.3.1 summarizes the well-known result in this case, which forms the basis for pseudo-marginal, nested SMC [Naesseth et al., 2015], and ABC methods. The next is that y has finite possible realizations. The result for this case is given in Section 10.3.2. Though intuitively straight-forward, it requires significant care to prove. The third case in Section 10.3.3 is concerned with the product of expectations. This result applies to many latent variable models and a number of probabilistic program examples, e.g. when the probability of a program trace is weighted by marginal likelihood estimates from other programs. The last case is that the integrand f is a polynomial (Section 10.3.4). It covers cases such as moment estimation and opens up interesting possibilities in using tractable approximations to the integrand f.

10.3.1 Linear f

Suppose that f is integrable and linear in its second argument, i.e. $f(y, \alpha v + \beta w) = \alpha f(y, v) + \beta f(y, w)$. In this case, we can rearrange the problem to a single expectation:

$$I = \mathbb{E}\left[f\left(y, \mathbb{E}\left[\phi(y, z)|y\right]\right)\right] = \mathbb{E}\left[\mathbb{E}\left[f(y, \phi(y, z))|y\right]\right] = \mathbb{E}\left[f(y, \phi(y, z))\right] \approx \frac{1}{N}\sum_{n=1}^{N} f(y_n, \phi(y_n, z_n))$$

where $(y_n, z_n) \sim p(y)p(z|y)$ if $\gamma(y)$ is of the form of (10.2a) and $y_n \sim p(y)$ and $z_n \sim p(z)$ are independently drawn if $\gamma(y)$ is of the form of (10.2b).

10.3.2 Finite Possible Realizations of y

If y must take one of finitely many values $y_1, \cdots, y_C$, then it is possible to use another approach to ensure the same convergence rate as standard Monte Carlo. The key observation is to note that in this case we can convert the nested problem (10.1) into C separate non-nested problems

$$I = \sum_{c=1}^{C} P(y = y_c)\, f(y_c, \gamma(y_c)) \tag{10.4}$$

which can then be estimated using

$$I_N = \sum_{c=1}^{C} (\hat{P}_N)_c\, (\hat{f}_N)_c \quad \text{where} \tag{10.5}$$

$$P(y = y_c) \approx (\hat{P}_N)_c = \frac{1}{N} \sum_{n=1}^{N} \mathbb{1}(y_n = y_c) \tag{10.6}$$

$$f(y_c, \gamma(y_c)) \approx (\hat{f}_N)_c = f\left(y_c, \frac{1}{N} \sum_{n=1}^{N} \phi(y_c, z_{n,c})\right), \tag{10.7}$$

with $y_n \overset{i.i.d.}{\sim} p(y)$ and $z_{n,c} \sim p(z|y_c)$ or $z_{n,c} \sim p(z)$ depending on whether our formulation uses (10.2a) or (10.2b). We can now show the following result, noting that as the same set of y_n's is used for every $(\hat{P}_N)_c$, calculating I_N requires N samples of y, CN samples of z (each drawn independently to the samples of y), CN evaluations of ϕ, and C evaluations of f.

Theorem 10.1. *If f is Lipschitz continuous, then the mean squared error of $I_N = \sum_{c=1}^{C} (\hat{P}_N)_c\, (\hat{f}_N)_c$ as an estimator for $I = \sum_{c=1}^{C} P(y = y_c)\, f(y_c, \gamma(y_c))$ converges at rate $O(1/N)$.*

Proof. See [Rainforth et al., 2017a]. □

10.3.3 Products of Expectations

In this section we consider the scenario where $\gamma(y)$ is equal to the product of multiple expectations, rather than just a single expectation as per (10.2a), namely

$$I = \mathbb{E}_{p(y)}\left[f\left(y, \prod_{\ell=1}^{L} \mathbb{E}_{p(z_\ell|y)}\left[\psi_\ell(y, z_\ell)\right]\right)\right]. \tag{10.8}$$

We now show that the product of expectations $\prod_{\ell=1}^{L} \mathbb{E}_{p(z_\ell|y)}\left[\psi_\ell(y, z_\ell)\right]$ can be rearranged to a single expectation and so the estimation of I corresponds to the formulation laid out in Section 10.2. Consider the set of random variables $\{z'_\ell\}_{\ell=1:L}$ such that each $z'_\ell|y \sim p(z_\ell|y)$ and the z'_ℓ are independent of one another. This can be achieved by, for example, taking L independent samples from the joint $Z_\ell \overset{i.i.d.}{\sim} p(z_{1:L}|y)$ and using the ℓ^{th} such draw for the ℓ^{th} dimension of z', i.e. setting $z'_\ell = \{Z_\ell\}_\ell$. For every $y \in \mathcal{Y}$ we now have

$$\prod_{\ell=1}^{L} \mathbb{E}_{p(z_\ell|y)}[\psi_\ell(y, z_\ell)] = \prod_{\ell=1}^{L} \mathbb{E}_{p(z'_\ell|y)}[\psi_\ell(y, z'_\ell)] = \mathbb{E}_{p(\{z'_\ell\}_{\ell=1:L}|y)}\left[\prod_{\ell=1}^{L} \psi_\ell(y, z'_\ell)\right]$$

which is a single expectation on an extended space and implies that (10.8) fits the NMC formulation. Furthermore, we can now show that if f is linear, the MSE of the NMC estimator (10.8) converges at the standard Monte Carlo rate $O(1/N)$, provided that the number of samples used in the inner estimators remains fixed.

Theorem 10.2. *Consider the NMC estimator*

$$I_N = \frac{1}{N}\sum_{n=1}^{N} f\left(y_n, \prod_{\ell=1}^{L}\frac{1}{M_\ell}\sum_{m=1}^{M_\ell}\psi_\ell(y_n, z'_{n,\ell,m})\right)$$

where each $y_n \in \mathcal{Y}$ and $z'_{n,\ell,m} \in \mathcal{Z}_\ell$ are independently drawn from $y_n \sim p(y)$ and $z'_{n,\ell,m}|y_n \sim p(z_\ell|y_n)$, respectively. If f is linear, the estimator converges almost surely to I, with a convergence rate of $O(1/N)$ in the mean square error for any fixed choice of $\{M_\ell\}_{\ell=1:L}$.

Proof. See [Rainforth et al., 2017a]. □

As this result holds in the case $L = 1$, an important consequence is that whenever f is linear, the same convergence rate is achieved regardless of whether we reformulate the problem to a single expectation or not, provided that the number of samples used by the inner estimator remains fixed. Conversely, as we will show in Section 10.4, if f is nonlinear, it will be necessary for the number of samples in *both* the inner and the outer estimators, i.e. N and each M_ℓ, to increase in order to achieve the convergence of (10.3).

10.3.4 Polynomial f

Perhaps surprisingly, when f is of the form

$$f(y, \gamma(y)) = \sum_{j=1}^{J} g_j(y)\,\gamma_j(y)^{\alpha_j} \quad \text{where} \quad \alpha_j \in \mathbb{Z}_{\geq 0},\ \forall j \in \{1, \ldots, J\} \tag{10.9}$$

then it is also possible to construct a standard Monte Carlo estimator by building on the ideas introduced in Section 10.3.3 and those introduced by Goda [2016]. The key idea is to note that

$$\left(\mathbb{E}\left[z\right]\right)^2 = \mathbb{E}\left[z\right]\mathbb{E}\left[z'\right] = \mathbb{E}\left[zz'\right]$$

where z and z' are i.i.d. Therefore, assuming appropriate integrability requirements, we can construct the following non-nested Monte Carlo estimator:

$$\begin{aligned}\mathbb{E}\left[\sum_{j=1}^{J} g_j(y)\,\gamma_j(y)^{\alpha_j}\right] &= \sum_{j=1}^{J}\mathbb{E}\left[g_j(y)\,\mathbb{E}\left[\phi(y, z_j)|y\right]^{\alpha_j}\right] = \sum_{j=1}^{J}\mathbb{E}\left[g_j(y)\mathbb{E}\left[\prod_{\ell=1}^{\alpha_j}\phi(y, z_{j,\ell})\middle| y\right]\right] \\ &= \sum_{j=1}^{J}\mathbb{E}\left[g_j(y)\prod_{\ell=1}^{\alpha_j}\phi(y, z_{j,\ell})\right] \approx \frac{1}{N}\sum_{n=1}^{N}\sum_{j=1}^{J} g(y_n)\prod_{\ell=1}^{\alpha_j}\phi(y_n, z_{j,\ell,n}).\end{aligned} \tag{10.10}$$

Here each $z_{j,\ell,n} \sim p(z_j|y)$ is drawn i.i.d. from the corresponding conditional distribution.

10.4 Convergence of Nested Monte Carlo

Since we cannot always unravel the target estimation using one of the special cases in the previous section, we must resort to NMC (or another nested estimation scheme) in order to compute I in general. Our aim here is to show that approximating $I \approx I_{N,M}$ is in principle possible, at least when f is well-behaved. In particular, we establish a convergence rate of the mean squared error of $I_{N,M}$ and prove a form of almost sure convergence to I. We further generalize our convergence rate to apply to the case of multiple levels of estimator nesting.

Before providing a formal examination of the convergence of NMC, we first provide more intuition about how we might expect to construct a convergent NMC estimator. Consider the diagram shown in Figure 10.1, and suppose that we want our error to be less than some arbitrary ε. Assume that f is sufficiently smooth that we can choose M large enough to make $|I - \mathbb{E}\left[f(y_n, (\hat{\gamma}_M)_n)\right]| < \varepsilon$ (we will characterize the exact requirements for this later). For this fixed M, we have a standard Monte Carlo estimator on an extended space $y, z_1, \ldots, z_M$ such that each sample constitutes one of the red boxes. As we take $N \to \infty$, i.e. taking all the samples in the green box, this estimator converges such that $I_{N,M} \to \mathbb{E}\left[f(y_n, (\hat{\gamma}_M)_n)\right]$ as $N \to \infty$ for fixed M. As we can make ε arbitrarily small, we can also achieve an arbitrarily small error.

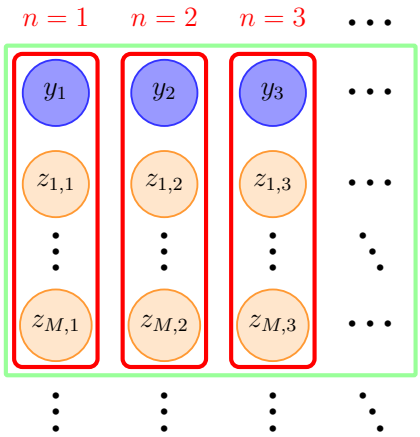


Figure 10.1: Convergence representation

Convergence bounds for NMC have previously been shown by Hong and Juneja [2009] in the financial statistics literature. Under the assumption that $\hat{\gamma}$ is Gaussian distributed (which is often reasonable due to the central limit theorem) and that f is thrice differentiable other than at some finite number of points, they have shown that it is possible to achieve a converge rate of $O(1/N + 1/M^2)$. We now show that these assumptions can be relaxed to only requiring f to be Lipschitz continuous, at the expense of weakening this bound to $O(1/N + 1/M)$.

Theorem 10.3. *If f is Lipschitz continuous and $f(y_n, \gamma(y_n)), \phi(y_n, z_{n,m}) \in L^2$, the mean squared error of $I_{N,M}$ converges to 0 at rate $O\left(1/N + 1/M\right)$.*

Proof. The Theorem follows as a special case of Theorem 10.5 which provides an exact form for this bound and a tighter bound that holds when f is continuously differentiable. See also Rainforth et al. [2017a] for a more accessible proof of this particular result. □

Remark 10.1. *This result can be carried over to cases where there are finite numbers of points for which f is not Lipschitz continuous by decomposing the estimator into separate truncated functions as per Hong and Juneja [2009].*

Inspection of the convergence rate above shows that, given a total number of samples $T = MN$, our bound is tightest when $N \propto M$, with a corresponding rate $O(1/\sqrt{T})$ (see Rainforth et al. [2017a]). When the additional assumptions of Hong and Juneja [2009] apply, this rate can be lowered to $O(1/T^{2/3})$ by setting $N \propto M^2$. We note that the result of Theorem 10.3 was recently independently derived by Fort et al. [2017] in work published shortly after our own [Rainforth et al., 2016a]. Fort et al. [2017] also show that a $O(1/N + 1/M^2)$ convergence rate can be achieved under weaker assumptions than those of Hong and Juneja [2009], namely that f is continuously differentiable, and that these results hold when the y_n are generated by a valid Markov chain, instead of being i.i.d..

We next consider the question of what is the minimal requirement on f to ensures some form of convergence? For a fixed single y_1, we have that $(\hat{\gamma}_M)_1 = \frac{1}{M}\sum_{m=1}^{M} \phi(y_1, z_{1,m}) \rightarrow \gamma(y_1)$ almost surely as $M \rightarrow \infty$, because the left-hand side is a Monte Carlo estimator. If f is continuous around y_1, this also implies $f(y_1, (\hat{\gamma}_M)_1) \rightarrow f(y_1, \gamma(y_1))$. Our candidate requirement is that this holds in expectation, i.e. that it holds when we incorporate the effect of the outer estimator. More precisely, we define $(\epsilon_M)_n = |f(y_n, (\hat{\gamma}_M)_n) - f(y_n, \gamma(y_n))|$ and require that $\mathbb{E}\left[(\epsilon_M)_1\right] \rightarrow 0$ as $M \rightarrow \infty$ (noting that $(\epsilon_M)_n$ are i.i.d. and so $\mathbb{E}\left[(\epsilon_M)_1\right] = \mathbb{E}\left[(\epsilon_M)_n\right], \forall n \in \mathbb{N}$). Informally, this "expected continuity" requirement is weaker than uniform continuity (and much weaker than Lipschitz continuity) because it does allow (potentially infinitely many) discontinuities in f. More formally we have the following result.

Theorem 10.4. *For $n \in \mathbb{N}$, let $(\epsilon_M)_n = |f(y_n, (\hat{\gamma}_M)_n) - f(y_n, \gamma(y_n))|$. Assume that $\mathbb{E}\left[(\epsilon_M)_1\right] \rightarrow 0$ as $M \rightarrow \infty$. Denote by Ω the sample space of our underlying probability space, so that $I_{\tau_\delta(M),M}$ can be thought of as a map from Ω to $\mathbb{R}$. Then, for every $\delta > 0$, there exists a measurable $A_\delta \subseteq \Omega$ with $\mathbb{P}(A_\delta) < \delta$, and a function $\tau_\delta : \mathbb{N} \rightarrow \mathbb{N}$ such that, for all $\omega \notin A_\delta$,*

$$I_{\tau_\delta(M),M}(\omega) \rightarrow I \quad as \quad M \rightarrow \infty.$$

Proof. See [Rainforth et al., 2017a]. □

As well as providing proof of a different form of convergence to any existing results, this result is particularly important because many, if not most, functions are not Lipschitz continuous due to the their behavior in the limits. For example, even the function $f(y, \gamma(y)) = (\gamma(y))^2$ is not

Lipschitz continuous because the derivative is unbounded as $|\gamma(y)| \to \infty$. On the other hand, the vast majority of problems involving this f will satisfy $\mathbb{E}\left[(\epsilon_M)_1\right] \to 0$.

We next consider the case of multiple levels of nesting. To formalize what we mean by arbitrary nesting, we first assume some fixed integral depth $D > 0$, and real-valued functions $f_0, \cdots, f_D$. We then define

$$\gamma_D\left(y^{(0:D-1)}\right) = \mathbb{E}\left[f_D\left(y^{(0:D)}\right)\middle| y^{(0:D-1)}\right] \quad \text{and}$$
$$\gamma_k(y^{(0:k-1)}) = \mathbb{E}\left[f_k\left(y^{(0:k)}, \gamma_{k+1}\left(y^{(0:k)}\right)\right)\middle| y^{(0:k-1)}\right],$$

for $0 \leq k < D$, where $y^{(k)} \sim p\left(y^{(k)}|y^{(0:k-1)}\right)$. Note that our single nested case corresponds to the setting of $D = 1$, $f_0 = f$, $f_1 = \phi$, $y^{(0)} = y$, $y^{(1)} = z$, $\gamma_0 = I$, and $\gamma_1 = \gamma$. Our goal is to estimate $\gamma_0 = \mathbb{E}\left[f_0\left(y^{(0)}, \gamma_1\left(y^{(0)}\right)\right)\right]$. To do so we will use the following NMC scheme:

$$I_D\left(y^{(0:D-1)}\right) = \frac{1}{N_D}\sum_{n=1}^{N_D} f_D\left(y^{(0:D-1)}, y_n^{(D)}\right) \quad \text{and}$$
$$I_k\left(y^{(0:k-1)}\right) = \frac{1}{N_k}\sum_{n=1}^{N_k} f_k\left(y^{(0:k-1)}, y_n^{(k)}, I_{k+1}\left(y^{(0:k-1)}, y_n^{(k)}\right)\right)$$

for $0 \leq k \leq D-1$, where each $y_n^{(k)} \sim p\left(y^{(k)}|y^{(0:k-1)}\right)$ is drawn independently. Note thus that there are multiple values of $y_n^{(k)}$ for each possible $y^{(0:k-1)}$ and that $I_k\left(y^{(0:k-1)}\right)$ is still a random variable given $y^{(0:k-1)}$.

We are now ready to provide our general result for the convergence bounds that applies to cases of repeated nesting, provides constant factors (rather than just using big O notation), and shows how the bound can be improved if the additional assumption of continuous differentiability holds. We note that for the particular case of a single nesting without continuous differentiability, then the bound is exact is the sense that, in addition to providing the required constant factors, there are no higher-order terms that are only dominated asymptotically.

Theorem 10.5. *If $f_0, \cdots, f_D$ are all Lipschitz continuous with associated Lipschitz constants*

$$K_k = \sup_{y^{(0:k)}}\left|\frac{\partial f_k\left(y^{(0:k)}, \gamma_{k+1}(y^{(0:k)})\right)}{\partial \gamma_{k+1}}\right|, \quad \forall k \in 0, \ldots, D-1$$

and if

$$\varsigma_k^2 = \mathbb{E}\left[\left(f_k\left(y^{(0:k)}, \gamma_{k+1}\left(y^{(0:k)}\right)\right) - \gamma_k\left(y^{(0:k-1)}\right)\right)^2\right] < \infty \quad \forall k \in 0, \ldots, D$$

then the mean squared error $\mathbb{E}\left[(I_0 - \gamma_0)^2\right]$ converges to 0 with rate

$$\mathbb{E}\left[(I_0 - \gamma_0)^2\right] \leq \frac{\varsigma_0^2}{N_0} + \sum_{k=1}^{D}\left(\prod_{\ell=0}^{k-1} K_\ell^2\right)\frac{\varsigma_k^2}{N_k} + O(\epsilon) \tag{10.11}$$

where $O(\epsilon)$ represents terms that become dominated as $N_0, \ldots, N_D \to \infty$.

If $f_0, \cdots, f_D$ are also continuously differentiable with first derivative bounds $K_0, \ldots, K_D$ and second derivative bounds $C_k = \sup_{y^{(0:k)}} \left| \frac{\partial^2 f_k\left(y^{(0:k)}, \gamma_{k+1}(y^{(0:k)})\right)}{\partial \gamma_{k+1}^2} \right|, \forall k \in 0, \ldots, D-1$, then this mean square error bound can be tightened to

$$\mathbb{E}\left[(I_0 - \gamma_0)^2\right] \leq \frac{\varsigma_0^2}{N_0} + \frac{1}{4}\left(\frac{C_0 \varsigma_1^2}{N_1} + \sum_{k=0}^{D-2}\left(\prod_{d=0}^{k} K_d\right)\frac{C_{k+1}\varsigma_{k+2}^2}{N_{k+2}}\right)^2 + O(\epsilon). \tag{10.12}$$

In the case of a single nesting we can further characterize $O(\epsilon)$ to give

$$\mathbb{E}\left[(I_0 - \gamma_0)^2\right] \leq \frac{\varsigma_0^2}{N_0} + \frac{4K_0^2\varsigma_1^2}{N_0 N_1} + \frac{2K_0\varsigma_0\varsigma_1}{N_0\sqrt{N_1}} + \frac{K_0^2\varsigma_1^2}{N_1} \tag{10.13}$$

$$\mathbb{E}\left[(I_0 - \gamma_0)^2\right] \leq \frac{\varsigma_0^2}{N_0} + \frac{C_0^2\varsigma_1^4}{4N_1^2}\left(1 + \frac{1}{N_0}\right) + \frac{K_0^2\varsigma_1^2}{N_0 N_1} + \frac{2K_0\varsigma_1}{N_0\sqrt{N_1}}\sqrt{\varsigma_0^2 + \frac{C_0^2\varsigma_1^4}{4N_1^2}} + O\left(\frac{1}{N_1^3}\right) \tag{10.14}$$

for when the continuous differentiability assumption does not hold and holds respectively.

Proof. See [Rainforth et al., 2017a]. □

At a high level, the key points of these results are a convergence rate of $O(\sum_{k=0}^{D} 1/N_k)$ when only Lipschitz continuity holds and $O(1/N_0 + (\sum_{k=1}^{D} 1/N_k)^2)$ when all the f_k are also continuously differentiable. As estimation requires drawing $T = \prod_{k=0}^{D} N_k$ samples, the convergence rate will rapidly diminish with repeated nesting. More precisely then the optimal convergence rates, as shown Rainforth et al. [2017a], are $O(T^{\frac{-1}{D+1}})$ and $O(T^{\frac{-2}{D+2}})$ respectively for the two cases (note the single nesting case is $D = 1$), both of which imply that exponentially more samples are required to achieve a given error as D is increased.

10.5 The Inevitable Bias of Nested Estimation

The previous section confirmed the capabilities of NMC; in this section we establish a limitation by proving that NMC schemes must produce biased estimates of $I(f)$ for certain functions f. In fact, our result applies more generally: we show that this holds for any Monte Carlo scheme that makes use of imperfect estimates $\hat{\zeta}_n$ of $\gamma(y_n)$, either via a nested Monte Carlo procedure (e.g. $\hat{\zeta}_n = (\hat{\gamma}_M)_n$), or when these inner estimates are generated by some other potentially deterministic methods such as a variational approximation [Blei et al., 2016] or Bayesian quadrature [O'Hagan, 1991].

Theorem 10.6. *Suppose that the random variables $\hat{\zeta}_n$ satisfy $\mathbb{P}(\hat{\zeta}_n \neq \gamma(y_n)) > 0$. Then we can choose f such that if $y_n \sim p(y)$, $\mathbb{E}\left[\frac{1}{N}\sum_{n=1}^{N} f(y_n, \hat{\zeta}_n)\right] \neq I(f)$ for any N (including $N \to \infty$).*

Proof. Take $f(y, w) = (\gamma(y) - w)^2$. Then $f(y, \gamma(y)) = 0$, so that $I(f) = 0$. On the other hand, $f(y_n, \hat{\zeta}_n) \geq 0$ since f is non-negative. Moreover, $f(y_n, \hat{\zeta}_n) > 0$ on the event $\{\hat{\zeta}_n \neq \gamma(y_n)\}$. Since we assumed this event has nonzero probability, it follows that $\mathbb{E}\left[f(y_n, \hat{\zeta}_n)\right] > 0$ and hence

$$\mathbb{E}\left[\frac{1}{N}\sum_{n=1}^{N} f(y_n, \hat{\zeta}_n)\right] = \frac{1}{N}\sum_{n=1}^{N} \mathbb{E}\left[f(y_n, \hat{\zeta}_n)\right] > 0 = I(f)$$

which gives the required result. □

By Jensen's inequality, it also follows relatively straightforwardly (see Rainforth et al. [2017a]) that *any* strictly convex or concave f entails a biased estimator when $\hat{\zeta}_n$ is unbiased but has non-zero variance given y_n, e.g. when $\hat{\zeta}_n$ is a Monte Carlo estimate. For certain nonlinear f, it may still be possible to develop unbiased estimation schemes using Russian roulette sampling [Lyne et al., 2015] or other debiasing techniques. However, this induces its own complications: for some problems the resultant estimates have infinite variance [Lyne et al., 2015] and as shown by Jacob et al. [2015], there are no general purpose "f-factories" that produce both non-negative and unbiased estimates for non-constant, positive output functions $f : \mathbb{R} \to \mathbb{R}^+$, given unbiased estimates for the inputs.

10.6 Empirical Verification

The convergence rates proven in Section 10.4 are only *upper bounds* on the worst-case performance we can expect. We will now examine whether these convergence rates are tight in practice, investigate what happens when our guidelines are not followed, and examine the effect of different strategies for choosing N_k under a sample budget. We note that one of the main applications of our results is provided later in Chapter 11, while an additional real-world example is given in [Rainforth et al., 2017a].

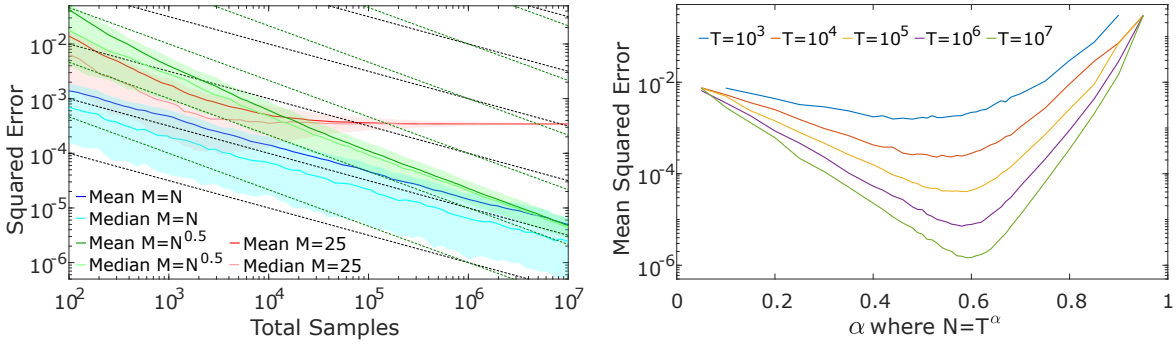
10.6.1 Simple Analytic Model

We start with the following simple analytically calculable problem

$$\begin{aligned} &y \sim \text{Uniform}(-1, 1), && z \sim \mathcal{N}(0, 1), \\ &\phi(y, z) = \sqrt{2/\pi}\exp\left(-2(y-z)^2\right), && f(y, \gamma(y)) = \log(\gamma(y)) = \log(\mathbb{E}_z[\phi(y, z)]). \end{aligned} \tag{10.15}$$

for which $I = \mathbb{E}_{p(y)}\left[f\left(y, \mathbb{E}_{p(z|y)}\left[\phi(y, z)\right]\right)\right] = \frac{1}{2}\left(\log(2) - \log(5) - \log(\pi)\right) - \frac{2}{15}$. Figure 10.2a shows the corresponding empirical convergence obtained by applying (10.3) to (10.15) directly. It shows that for this problem the theoretical convergence rates from Theorem 10.5 are indeed realized.[1] The figure also demonstrates the danger of not increasing M with N, showing that

[1] Note the continuous differentiability assumption is satisfied as $\gamma(y) \neq 0$ or ∞ for any $y \in [-1, 1]$.



(a) Convergence of NMC for different τ.

(b) Final error for different T and N.

Figure 10.2: Empirical convergence of NMC for (10.15). Shown left is the convergence in total samples for different ways of setting M and N. Results are averaged over 1000 independent runs, while shaded regions give the 25%-75% quantiles. We see that the theoretical convergence rates (as shown by the dash lines) are observed. The fixed M case converges at the Monte Carlo error rate, but to a biased solution. Shown right is the final error for different total sample budgets as a function of α where $N = T^\alpha$ and $M = T^{1-\alpha}$ iterations are used for the outer and inner estimators respectively. This shows that even though $\alpha = 2/3$ is the asymptotically optimal allocation strategy, this is not the optimal solution for finite T. Nonetheless, as T increases, the optimum value of α increases, starting at around 0.5 for $T = 10^3$ and reaching around 0.6 for $T = 10^7$.

the NMC estimator converges to an incorrect solution when M is held constant. Figure 10.2b shows the effect of varying N and M for various fixed sample budgets T and demonstrates that the asymptotically optimal strategy can be suboptimal for finite budgets.

10.6.2 Repeated Nesting

We next consider some simple models with multiple levels of nesting, starting with

$$y^{(0)} \sim \text{Uniform}(0,1), \quad y^{(1)} \sim \mathcal{N}(0,1), \quad y^{(2)} \sim \mathcal{N}(0,1), \tag{10.16a}$$

$$f_0\left(y^{(0)}, \gamma_1\left(y^{(0)}\right)\right) = \log \gamma_1\left(y^{(0)}\right) \tag{10.16b}$$

$$f_1\left(y^{(0:1)}, \gamma_2\left(y^{(0:1)}\right)\right) = \exp\left(-\frac{1}{2}\left(y^{(0)} - y^{(1)} - \log \gamma_2\left(y^{(0:1)}\right)\right)\right) \tag{10.16c}$$

$$f_2\left(y^{(0:2)}\right) = \exp\left(y^{(2)} - \frac{y^{(0)} + y^{(1)}}{2}\right) \tag{10.16d}$$

which has analytic solution $I = \mathbb{E}\left[f_0\left(y^{(0)}, \mathbb{E}\left[f_1\left(y^{(0:1)}, \mathbb{E}\left[f_2\left(y^{(0:2)}\right)\middle|y^{(0:1)}\right]\right)\middle|y^{(0)}\right]\right)\right] = -3/32$. The converge plot shown in Figure 10.3 shows that the theoretically expected convergence[2] behaviors are observed for different methods of setting N_0, N_1, and N_2.

We next consider the empirical performance of different strategies for choosing N_0, N_1, N_2 under a finite fixed budget $T = N_0 N_1 N_2$. To do this, we define α_1 and α_2 such that $N_0 = T^{\alpha_1}$,

[2]Strictly speaking the assumptions of Theorem 10.3 are not actually satisfied for this model and the later modifications because $K_1 = C_1 = \infty$. However, we see convergence behavior as if the assumptions were satisfied. This highlights the importance of Theorem 10.4 because f_1 does satisfy this weaker assumption.

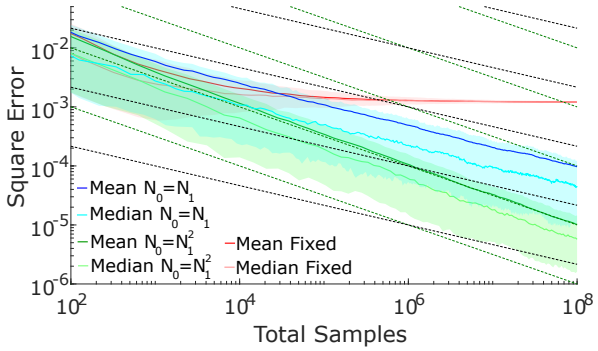


Figure 10.3: Empirical convergence of NMC to (10.16) for an increasing total sample budget $T = N_0 N_1 N_2$. Results are averaged over 1000 independent runs, while shaded regions give the 25%-75% quantiles. Shown in red is the convergence with a fixed $N_2 = 5$ and $N_1 = N_2^2$, for which we see gives convergence to a biased solution, such that the error remains bounded as T increases. Shown in blue is the convergence when setting $N_0 = N_1 = N_2$, which we see converges at the expected $O(T^{-1/3})$ rate. The black dashed lines provide a reference for this expected convergence of this estimator. Shown in green is the convergence when setting $N_0 = N_1^2 = N_2^2$ which we see again gives the theoretical convergence rate, this time $O(T^{-1/2})$ with the dashed green lines providing reference.

$N_1 = T^{\alpha_2(1-\alpha_1)}$, and $N_2 = T^{(1-\alpha_1)(1-\alpha_2)}$ and then consider the variation in the mean squared error with (α_1, α_2). In particular, we look to establish the optimal empirical setting under the fixed budget $T = 10^6$. To investigate how sensitive the empirically optimal allocation is to the problem at hand, we consider both the model described in (10.16) and also two slight variations. The first keeps everything the same as (10.16) except the following replacements

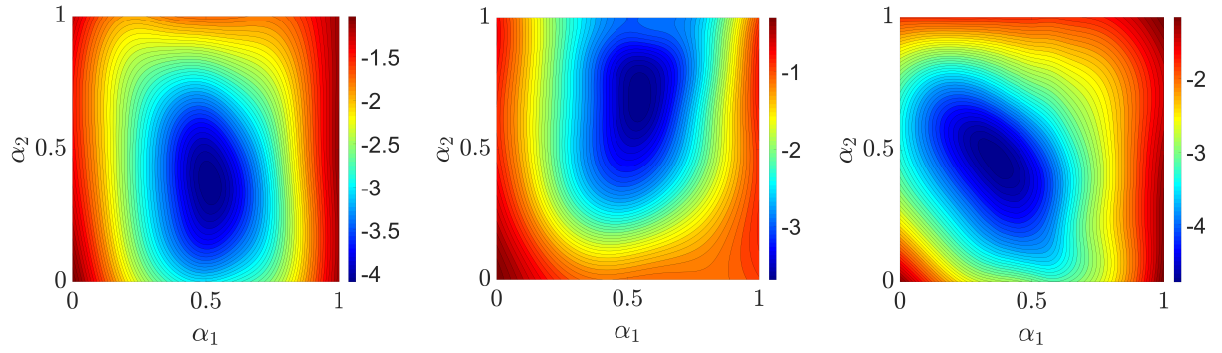
$$y^{(0)} \sim \text{Uniform}(-1, 1), \tag{10.17a}$$

$$f_2\left(y^{(0:2)}\right) = \exp\left(\frac{y^{(1)} + y^{(2)} - y^{(0)}}{2}\right) \tag{10.17b}$$

while the second instead replaces $y^{(0)}$ with $y^{(0)}/10$ in the definitions of f_1 and f_2. The two respectively have analytic solutions of $I = 11/32$ and $I = 39/160$.

To try and find the optimal (α_1, α_2) and establish the variation of the MSE more generally, we ran a Bayesian optimization algorithm (specifically the "Black-box Bayesian optimization" algorithm of Rainforth et al. [2015]) to optimize the mean squared error, estimated by averaging 1000 trials at each iteration, with respect to (α_1, α_2). This produced the performance characterizations shown in Figure 10.4, corresponding to contour plots of $\log_{10}\left(\mathbb{E}\left[(I_0 - \gamma_0)^2\right]\right)$ as estimated by the surrogate function generated after 200 Bayesian optimization iterations. It further found respective optimal values for (α_1, α_2) of $(0.53, 0.36)$, $(0.55, 0.69)$, and $(0.38, 0.45)$, giving corresponding values for (N_0, N_1, N_2) of $(1563, 10, 64)$, $(1957, 73, 7)$, and $(192, 47, 111)$.[3] By comparison,

[3]Note that there is inevitably a small bias introduced by the fact that each N_k has to be a whole number. The true values of T used for these solutions were 1000320, 1000027, and 1001664 respectively.



(a) Original **(b)** First modification **(c)** Second modification

Figure 10.4: Contour plots of $\log_{10}\left(\mathbb{E}\left[(I_0 - \gamma_0)^2\right]\right)$ (i.e. $\log_{10}$ MSE, estimated by averaging over 1000 individual estimations) for different allocations of the sample budget $T = 10^6$ between N_0, N_1, and N_2 for problem shown in (10.16) and two modifications explained in the text.

the asymptotically optimal setup suggested by our theoretical results is $\alpha_1 = \alpha_2 = 0.5$ giving $N_0 = 977$, $N_1 = 32$, and $N_2 = 32$. This demonstrates that even for relatively large values of T, the finite budget optimal allocation can vary significantly from the asymptotically optimal solution. We also see that the relative optimally can change quite significantly with changes to the target problem. For example, the second modification meant it was more preferable to run fewer iterations of the outer estimator, perhaps because it reduced the relative importance of $y^{(0)}$ compared to the other variables in f_1 and f_2. However, further work is required to establish concrete empirical guidelines and investigate whether methods that adaptively allocate the number of samples might be feasible. In the meantime, the asymptotically optimal choice (presuming continuous differentiability) of $N_0 \propto N_1^2 \propto \cdots \propto N_D^2$ seems to a reasonable practical choice on average.

10.7 Implications For Nesting Probabilistic Programs

Given our results, it is now natural to ask: what are the implications for nesting probabilistic programming queries?[4] In particular, when is doing this valid and are there any precautions we can take to avoid problems? Before we get to these questions though, we first need to explicitly define what we mean by a nested query. To this end, we distinguish between nested calling structures (such as the example given in Figure 4.7) and queries that do not represent a single estimation, which we will refer to as nested queries. Our motivation for defining only the latter as nested is that cases of the former can always be represented as a single query, because they still represent a single expectation and define models for which the unnormalized target distribution (4.3) can be evaluated exactly. Therefore, although these models are of clear

[4] A more up-to-date version of the work in this section is provided in [Rainforth, 2018].

importance, they are not a fundamentally different problem class to standard Bayesian inference problems and so we assert that existing Monte Carlo converge results directly apply. However, it can be deceptively difficult to establish whether a model requires query nesting or just nested calling. For example, the Schelling coordination example we gave in Figure 4.7 is clearly an example of a nested calling structure as it is explicitly a single query, while the original version of this problem [Stuhlmüller and Goodman, 2014, Figure 6] is actually an (equivalent!) example of a true nested query. It turns out that this constitutes one of our special cases that can be reformulated to a single estimate (namely no random variables are passed *to* the nested query). In the rest of this section, we will delineate possible ways one might want to nested probabilistic program queries and assert their respective correctness (or lack thereof) and required conditions. Namely, we will consider *sampling* from the conditional distribution of another query, *observing* another query, and using *estimates as variables* in another query. We will further identify some special cases which can be rearranged to single estimators or provide Monte Carlo convergence rates even when expressed as nested queries.

10.7.1 Nested Sampling

One of the clearest ways one might want to nest queries is by sampling from the conditional distribution of one query inside another query. Due to the "sample then check" setup of Church, all of the examples given in Stuhlmüller and Goodman [2014] are by proxy this type of query nesting,[5] as is any usage of the `conditional` predicate in Anglican.[6] Such problems fall under a more general framework of *nested inference* Mantadelis and Janssens [2011] or equivalently inference for so-called *doubly intractable* distributions [Murray et al., 2006] (or multiply intractable distributions for repeated nesting [Stuhlmüller and Goodman, 2014]). The key feature of these problems is that they include terms with unknown, *parameter dependent*, normalization constants. For nested probabilistic programming queries, this is manifested in *conditional normalization* within a query. Consider as an example the following model using a nested calling structure

```
(defm inner [y D]
 (let [z (sample (gamma y 1))]
  (observe (normal z y) D)
  z))
```

```
(defquery outer [D]
 (let [y (sample (beta 2 3))
       z (inner y D)]
  (* y z)))
```

[5] Even though the nested calls are made inside the conditioning predicate for these examples, the semantics of the language is to sample these and then check if the predicate holds as a hard constraint. This is equivalent to doing a nested sampling in the program and then evaluating a deterministic predicate using our PPL notation.

[6] This is because `conditional` is only currently supported for use with `sample` statements and inference algorithms that do not require the density to be evaluated.

compared to a following nested query model

```
(defquery inner [y D]
 (let [z (sample (gamma y 1))]
  (observe (normal z y) D)
  z))
```

```
(defquery outer [D]
 (let [y (sample (beta 2 3))
       dist (conditional inner)
       z (sample (dist y D))]
  (* y z)))
```

where we have replaced the **defm** with a query nesting. The unnormalized distribution on traces for the first model is straightforwardly given by

$$\gamma_1(z, y, D) = p(y)p(z|y)p(D|y, z) = \text{BETA}(y; 2, 3)\text{GAMMA}(z; y, 1)\mathcal{N}(D; z, y^2),$$

for which we can clearly use conventional Monte Carlo inference schemes. The second model is only subtly, but critically different in that $p(z|y)p(D|y, z)$ is conditionally normalized before being used in the outer query

$$\gamma_2(z, y, D) = p(y)p(z|y, D) = p(y)\frac{p(z|y)p(D|y, z)}{\int p(z|y)p(D|y, z)dz} = p(y)\frac{p(z|y)p(D|y, z)}{p(D|y)} \neq \gamma_1(z, y, D).$$

The key point here is that the partial normalization constant $p(D|y)$ depends on y and so $\gamma_2(z, y, D)$ is doubly intractable because we cannot evaluate our unnormalized target distribution exactly. By normalizing $p(z|y)p(D|y, z)$ within the outer query, **conditional** has changed the distribution defined by the program. Another interesting way of looking at this is that wrapping `inner` in **conditional** has "protected" y from the conditioning in `inner` – noting that $\gamma_1(z, y, D) \propto p(y|D)p(z|y, D)$ – such that the **observe** statement only affects the probability of z given y and not the probability of y.

Unfortunately the inverse function $f(x) = 1/x$ constitutes a nonlinear mapping and so expectations with respect to nested queries defined in this way using **conditional** will generally require nested estimation schemes unless we can find a means of sampling from $p(z|y, D)$ exactly with finite computational budget. The obvious question is therefore what behavior do we need for **conditional**, or query nesting through sampling more generally, in order to ensure consistency?

One possible approach would be to attempt to use an exact sampling method for sampling from **conditional**. Presuming we cannot do this analytically [Cornish et al., 2017], the most obvious way to generate such exact samples is using rejection sampling (see Section 5.3.1), as done in Church.[7] However, even this opens up a minefield of theoretical and practical questions. For example, because the acceptance rate of an inner query might depend on the value of the input variable, it would presumably not be valid to generate all the variables at once and then only accept the sample if every condition is satisfied. Church instead takes an approach whereby no sample

[7] Note, though, that this is not the only way to generate exact samples from an unnormalized target distribution (see e.g. [Craiu and Meng, 2011]).

ever returns until it passes its local acceptance criterion. Although doing this for soft conditioning in an automated way might be insurmountably problematic (because of the need to choose a valid C in (5.18)), it does open up the intriguing prospect of providing a standard Monte Carlo convergence rate for a set of inherently nested problems. This is because although the performance clearly gets exponentially worse with increased nesting (as the probability of a successful return is the product of an increasing number of expected acceptance rates), this is a change in the constant factor of the computation, not its scaling with the number of samples taken. This is perhaps easiest to see by noting that generating a single exact sample of the distribution can be done in finite time using rejection sampling (and thus still converges with a fixed $N_1, \ldots, N_D$), whereas in general NMC we had that $N_1, \ldots, N_D$ need to increase as N_0 increases for convergence. If correct, this assertion would be a somewhat startling result: it suggests that Monte Carlo estimates made using nested rejection sampling have a fundamentally different convergence rate for nested inference problems (though not nested estimation problems more generally) than, say, nested self-normalized importance sampling (SNIS). Amongst other things, this might lead, perhaps through combination with the work of Møller et al. [2006], to a contradiction of the, ostensibly very reasonable, conjecture made by Murray and Ghahramani [2004] that there are no generic tractable MCMC schemes for doubly intractable distributions. It is beyond the scope of the work to analyze this hypothesis more formally, or the rabbit hole of nested exact sampling methods more generally, but we highlight that this constitutes a potentially fascinating line of inquiry.

Without access to exact sampling methods, it is straightforward to see that asymptotic bias (i.e. bias even if we take an infinite number of samples from the outer query) is, in general, unavoidable for query nesting if each call of `conditional` only carries out finite computation. This follows from our general NMC results and the fact that normalization constitutes a nonlinear mapping. For example, imagine we use SNIS to generate outputs from `conditional` with a fixed number of internal samples proposed for each given as output. As we showed in Section 5.3.2.1, expectations with respect to these samples will be biased, which we can think of in the NMC context as occurring because the inversion of the unbiased marginal likelihood estimate induces a bias. In addition to the possibility that we might be able to get around this problem using exact sampling as discussed in the last paragraph, it might also be possible to use debiasing techniques to remove the particular bias that is generated, see e.g. Glynn and Rhee [2014]; Jacob et al. [2017].

One special case where consistency can be maintained, even without rearrangement, is if the inner query takes no random variables as inputs, as is the case for the Schelling coordinate example in [Stuhlmüller and Goodman, 2014, Figure 6]. A condition for this is that we use a

method that provides an unbiased estimate of the marginal likelihood (e.g. importance sampling or SMC) and weight the trace appropriately with the product of this marginal likelihood estimate and the internal self-normalized weight of the returned sample. If this conditioned is maintained, we can demonstrate the validity of the approach by noting that the normalization constant of the nested query is not parameter dependent, i.e. $p(D|y) = p(D)$ using the notation from our earlier example. It can, therefore, be factored into that of the outer query as the nested query now defines the same distribution as if the conditional query were replaced with a **defm**. We can therefore actually view inference on such problems as taking a pseudo-marginal approach, provided that all samples returned from conditional are properly weighted [Naesseth et al., 2015] as previously described. The correctness of this approach then clearly follows from a combination of Theorems 10.1 and 10.2. However, Anglican's current use of **conditional** does not satisfy our requirement as it only returns unweighted samples.[8] Therefore, for Anglican to provide consistent estimates in such situations, we would need to change the probabilistic semantics of **conditional**, such that it both produces a sample and simultaneously weights the trace using the unnormalized weight of the returned particle in the inner query. In other words, it would need to operate in a similar fashion to the following simplified code

```
(defn conditional-sample [inner-query inputs inf-alg M inf-opts]
 (let [samp (nth M (doquery inf-alg inner-query inputs inf-opts))]
  (observe (factor) (get-log-weight samp))
  (:result samp)))
```

where we first run inference with budget `M` to produce a sample, then weight the program trace using the (unnormalized) sample weight, and finally return the value of the sample. Remember though that this only addresses the special case where `input` is not a random variable and `inf-alg` is an inference algorithm producing properly weighted samples.

Another special case is problems where the variables passed to the inner query can only take on, say C, finite possible values. As explained in Section 10.3.1, such problems can always be rearranged to C separate estimators such that the standard Monte Carlo error rate can be achieved. For repeated nesting, this rearrangement can be recursively applied until one achieves a complete set of non-nested estimators. This special case is one which requires rearrangement or a specialist consideration by the language back-end (as done by Stuhlmüller and Goodman [2012, 2014]; Cornish et al. [2017]), with naïve implementations leading to NMC convergence rates. It emphasizes that nested problems with continuous variables are a fundamentally different

[8] Unfortunately, this means that, though still practically potentially useful, Anglican's current usage of **conditional** for nesting never provides consistent estimates unless the inner query contains no conditioning, in which case its behavior is equivalent to **defm** anyway. It should be noted, though, that no claims to the contrary have ever been made, with **conditional** still very much a developmental feature of the language.

problem class to those with only discrete bounded variables (or only continuous variables at the bottom layer) and we must, therefore, be wary of presuming that results from the discrete case carry over. Nonetheless, in the same way that it is often necessary to use Monte Carlo rather than enumeration for inference in discrete problems because there are too many possible outcomes to enumerate, it will still often be necessary from a practical perspective to use NMC for discrete problems. Ideally one should then do this in a manner that identifies and exploits the fact that the inner estimator is often called multiple times with the same inputs such that the samples from these calling can be combined (e.g. as done by Stuhlmüller and Goodman [2012]).

10.7.2 Nested Observation

An alternative way one might wish to nest queries is through nested observation, whereby we observe the output of another query rather than sampling from it. For hard-constraints such as used in Church, this is equivalent to query nesting through sampling. However, for soft constraints it can be substantially different. The distinction is the same as that between `sample` and `observe` – we are observing an outcome of a program, rather than sampling from it. As `observe` returns no outputs, the mathematical interpretation of observing the output of another query is that we are weighting the outer program trace by the partition function estimate of the inner query. Presuming that our partition function estimates are unbiased (which they will be if we use, say, SMC for the inner estimator), then this corresponds to a particular instance of the products of expectations special case covered by Theorem 10.2. It, therefore, immediately follows that if the outer estimator uses importance sampling, then such an approach is consistent. More generally, this case corresponds to a pseudo-marginal approach where the samples from the inner estimator are never returned. We can, therefore, use existing results to demonstrate the consistency of the approach for other choices of outer estimator. For example, consistency when using SMC in the outer estimator follows directly from a combination of Theorem 10.2 and the results of Naesseth et al. [2015]. It should also be possible to construct consistent approaches using MCMC samplers at the top level as per Andrieu and Roberts [2009], but this would require more careful consideration.

In addition to the statistical complications just discussed, there are some significant semantic complications associated with observing another query: there are actually two ways we might want to "observe" a nested query. The first is simply to run the nested query unmodified and factor the trace probability by the marginal likelihood estimate. This is a perfectly reasonable thing to want to do, allowing us to, for example, construct a PMMH sampler (see Section 6.2.4).

```
(defm prior [] (normal 0 1))
(defm lik [theta d] (normal theta d))

(defquery inner-q [y d]
 (let [theta (sample (prior))]
  (observe (lik theta d) y)))

(defn inner-E [y d M]
 (->> (doquery :importance
        inner-q [y d])
      (take M)
      log-marginal))

(with-primitive-procedures [inner-E]
 (defquery outer-q [d M]
  (let [theta (sample (prior))
        y (sample (lik theta d))
        log-lik (observe*
                  (lik theta d) y)
        log-marg (inner-E y d M)]
   (- log-lik log-marg))))

(defn outer-E [d M N]
 (->> (doquery :importance
        outer-q [d M])
      (take N)
      (map :result)
      mean))
```

Figure 10.5: Anglican code for Bayesian experimental design corresponding to the estimator given in (11.8). By changing the definitions of `prior` and `lik`, this code can be used as a NMC estimator (consistent as $N, M \to \infty$) for any static Bayesian experimental design problem. Here **`observe*`** is a function for returning the log likelihood (it does not affect the trace probability), **`log-marginal`** produces a marginal likelihood estimated from a collection of weighted samples, and **`->>`** successively applies a series of functions calls, using the result of one as the last input the next. When `outer-E` is invoked, this runs importance sampling on `outer-q`, which, in addition to carrying out its own computation, calls `inner-E`. This in turn invokes another inference over `inner-q`, such that a Monte Carlo estimate using `M` samples is constructed for each sample of `outer-q`. Thus `log-marg` is Monte Carlo estimate itself. The final return is the (weighted) empirical mean for the outputs of `outer-q`.

In fact, this was exactly what we did alongside some manual code transformations to construct the PMMH benchmarks for BOPP in Section 9.5. However, this is not really observing the output of the inner query: it is instead a means of replacing an analytic likelihood term with an unbiased estimate. The other way we might thus want to observe a query is to estimate the likelihood of it producing a particular output. Unfortunately, in general, this will be infeasible (at least without code analysis) because the density of the output variables is affected by deterministic computations in the query (see discussion of program outputs in Section 4.3.1). This is identical to the problem we encountered for BOPP, where we could not optimize with respect to arbitrary variables in the program because of not being able to calculate their density. This now presents an interesting possibility – we could use exactly the same marginal transformation as in BOPP, but then use this to estimate the partition function with certain variables "clamped". We can actually think of BOPP as being a nested estimation scheme where the outer estimator is an optimizer. Therefore, we could use the same framework as BOPP but replace the BO optimizer with an estimation scheme, i.e. the outer query. In fact, if desired, we could even maintain the same usage of the GP surrogate in BOPP to, for example, carry out estimation using Bayesian quadrature [Osborne et al., 2012] or the scheme laid out by [Gutmann and Corander, 2016].

10.7.3 Estimates as Variables

The third and final way we consider using nested estimation in probabilistic programming is to use expectation estimates as first class variables. In our previous examples, nonlinear usage of expectations only ever manifested through inversion of the normalization constant. These methods are therefore clearly insufficient for expressing more general nested estimation problems as would be required by, for example, experimental design. Anglican does not explicitly support such behavior inside pure Anglican code, but one can still enforce such behavior by either calling `doquery` inside a `defdist` deceleration or in a `defn` function passed to a query using `with-primitive-procedures`. An example of this is shown in Figure 10.5 corresponding to a program for carrying out Bayesian experimental design (see Chapter 11). The validity of this "estimates as variables" approach is effectively equivalent to that of NMC more generally (as this effectively allows any nested estimation problem to be defined) and needs to be considered on a case-by-case basis as per the general guidelines we have provided throughout this chapter.

11 Automated Adaptive Design

From tuning microscopes to designing surveys on politics, all problems of experimental design can be reduced to the same mathematical abstraction: choosing a design that maximizes the expected information gained from the experiment. Bayesian experimental design (BED) [Chaloner and Verdinelli, 1995; Sebastiani and Wynn, 2000] provides a powerful framework for this abstraction. In BED one constructs a likelihood model for the experiment outcome and then uses this, along with a prior on the parameters one wishes to learn about, to find the experimental setup that maximizes the expected information gathered by the experiment. If the model is correct, this forms a design strategy that is optimal from an information-theoretic viewpoint [Sebastiani and Wynn, 2000]. The general nature of its formulation means that BED can, at least in principle, be applied to almost any experimental design situation and it has been successfully used in a wide array of fields such as psychology [Myung et al., 2013; Cavagnaro et al., 2014; Vincent and Rainforth, 2017], Bayesian optimization [Hennig and Schuler, 2012; Hernández-Lobato et al., 2014], and bioinformatics [Vanlier et al., 2012]. BED can be particularly useful in *sequential* design settings, where it provides a means of constructing adaptive strategies that use previous experience in order to make decisions which optimize the sequential learning process.

Unfortunately, as we alluded to in the previous chapter, BED problems, in general, require the calculation of a nested estimation. In this chapter, we will provide a brief introduction to BED and show how our results from the last chapter can be used to derive a new estimator for BED equations which has a better convergence rate for discrete output problems than the naïve estimator often used. We will further introduce a novel framework for automating the design of sequential experiments and demonstrate how this can be effectively applied to automating psychological trials [Vincent and Rainforth, 2017].

11.1 Bayesian Experimental Design

In this section, we will introduce the BED equations more formally. Let the parameters of interest be denoted by $\theta \in \Theta$, for which we define a prior distribution $p(\theta)$. Let the probability of achieving outcome $y \in \mathcal{Y}$, given parameters θ and a design $d \in \mathcal{D}$, be defined by the

likelihood model $p(y|\theta, d)$. Under our model, the outcome of the experiment given a chosen d is distributed according to

$$p(y|d) = \int_\Theta p(y, \theta|d)d\theta = \int_\Theta p(y|\theta, d)p(\theta)d\theta. \tag{11.1}$$

where we have used the fact that $p(\theta) = p(\theta|d)$ because θ is independent of the design. Our aim is to choose the optimal design d under some criterion. We, therefore, define a utility function, $U(y, d)$, representing the utility of choosing a design d and getting a response y. Typically our aim is to maximize information gathered from the experiment, and so we set $U(y, d)$ to be the gain in Shannon information between the prior and the posterior

$$U(y, d) = \int_\Theta p(\theta|y, d) \log(p(\theta|y, d))d\theta - \int_\Theta p(\theta) \log(p(\theta))d\theta. \tag{11.2}$$

However, we are still uncertain about the outcome. Thus, we use the expectation of $U(y, d)$ with respect to $p(y|d)$ as our target:

$$\begin{aligned} \bar{U}(d) &= \int_\mathcal{Y} p(y|d) \left(\int_\Theta p(\theta|y, d) \log(p(\theta|y, d))d\theta - \int_\Theta p(\theta) \log(p(\theta))d\theta \right) dy \\ &= \int_\mathcal{Y} \int_\Theta p(y, \theta|d) \log \left(\frac{p(\theta|y, d)}{p(\theta)} \right) d\theta dy \end{aligned} \tag{11.3}$$

noting that this corresponds to the mutual information between the parameters θ and the observations y. The Bayesian-optimal design is then given by

$$d^* = \arg\max_{d \in \mathcal{D}} \bar{U}(d). \tag{11.4}$$

We can intuitively interpret d^* as being the design that most reduces the uncertainty in θ on average over possible experimental results. If our likelihood model is correct, i.e. if experimental outcomes are truly distributed according to $p(y|\theta, d)$ for a given θ and d, then it is easy to see from the above definition that d^* is the true optimal design, in terms of information gain, given our current information about the parameters $p(\theta)$. In practice, our likelihood model is an approximation of the real world. Nonetheless, BED remains a very powerful and statistically principled approach that is typically significantly superior to other, more heuristic-based, alternatives. For example, the state-of-the-art entropy based Bayesian optimization acquisition strategies we discussed in Section 8.3.1.4 are particular cases of BED. However, a major drawback to the BED approach is that it is typically difficult and computationally intensive to carry out. Not only does it represent an optimization of an intractable expectation, this expectation is itself nested because the integrand is itself intractable due to the $p(\theta|y, d)$ term.

11.2 An Improved Estimator for Discrete Problems

As we explained in the last section, estimating d^* is in general challenging because the posterior $p(\theta|y,d)$ is rarely known in closed form. One popular method within the psychology literature is to use a grid-search approach to the required inference [Kontsevich and Tyler, 1999; Prins, 2013], which subsequently provides (biased) estimates for $p(\theta|y,d)$. However, this is highly unsatisfactory for a host of obvious reasons such as the generally poor performance of grid-search for inference, the use of a nested estimation procedure with accompanying poor performance as shown in Chapter 10, and the need to carefully choose the grid.

To address these issues, we instead derive a nested Monte Carlo estimator as per, for example, Myung et al. [2013] and further show how, when y can only take a finite number of values, a substantially improved, non-nested, Monte Carlo estimator can be derived using the results from Chapter 10. The first step is to rearrange (11.3) using Bayes' rule (remembering $p(\theta) = p(\theta|d)$)

$$\begin{aligned}\bar{U}(d) &= \int_{\mathcal{Y}}\int_{\Theta} p(y,\theta|d)\log\left(\frac{p(y|\theta,d)}{p(y|d)}\right)d\theta dy \\ &= \int_{\mathcal{Y}}\int_{\Theta} p(y,\theta|d)\log(p(y|\theta,d))d\theta dy - \int_{\mathcal{Y}} p(y|d)\log(p(y|d))dy.\end{aligned} \tag{11.5}$$

The former term can now be evaluated using standard MC approaches as the integrand is analytic, namely we can use

$$\bar{U}_1(d) = \int_{\mathcal{Y}}\int_{\Theta} p(y,\theta|d)\log(p(y|\theta,d))d\theta dy \approx \frac{1}{N}\sum_{n=1}^{N}\log(p(y_n|\theta_n,d)) \tag{11.6}$$

where $\theta_n \sim p(\theta)$ and $y_n \sim p(y|\theta=\theta_n,d)$.[1] In contrast, the second term of (11.5) is not directly amenable to standard MC estimation as the marginal $p(y|d)$ represents an expectation and taking its logarithm represents a nonlinear functional mapping. Here we have

$$\bar{U}_2(d) = \int_{\mathcal{Y}} p(y|d)\log(p(y|d))dy \approx \frac{1}{N}\sum_{n=1}^{N}\log\left(\frac{1}{M}\sum_{m=1}^{M} p(y_n|\theta_{n,m},d)\right) \tag{11.7}$$

where $\theta_{n,m} \sim p(\theta)$ and $y_n \sim p(y|d)$. We can sample the latter by first sampling an otherwise unused $\theta_{n,0} \sim p(\theta)$ and then sampling a single corresponding $y_n \sim p(y|\theta_{n,0},d)$.

Putting (11.6) and (11.7) together (and renaming θ_n from (11.6) as $\theta_{n,0}$ for notational consistency with (11.7)) we now have the following complete estimator, implicitly used by

[1] Note that evaluating (11.6) involves both sampling from $p(y|\theta,d)$ and directly evaluating it point-wise. The latter of these cannot easily be avoided, but in the scenario where we do not have direct access to a sampler for $p(y|\theta,d)$, we can use the standard importance sampling trick, sampling instead $y_n \sim q(y|\theta=\theta_n,d)$ and weighting the samples in (11.6) by $w_n = \frac{p(y_n|\theta_n,d)}{q(y_n|\theta_n,d)}$.

Myung et al. [2013] amongst others,

$$\bar{U}(d) \approx \frac{1}{N}\sum_{n=1}^{N}\left[\log(p(y_n|\theta_{n,0},d)) - \log\left(\frac{1}{M}\sum_{m=1}^{M} p(y_n|\theta_{n,m},d)\right)\right] \tag{11.8}$$

where $\theta_{n,m} \sim p(\theta)\ \forall m \in 0:M,\ n \in 1:N$ and $y_n \sim p(y|\theta=\theta_{n,0},d)\ \forall n \in 1:N$. This naïve NMC estimator, Anglican code for which is provided in Figure 10.5, gives a convergence rate of $O(1/N + 1/M^2)$ as per Theorem 10.5.

We now show that if y can only take on one of C possible values $(y_1,\ldots,y_C)$, we can achieve significant improvements in the convergence rate by using a similar approach to that introduced in Section 10.3.2 to derive the following non-nested estimator for (11.5)

$$\begin{aligned}
\bar{U}(d) &= \int_{\mathcal{Y}}\int_{\Theta} p(y,\theta|d)\log(p(y|\theta,d))d\theta dy - \int_{\mathcal{Y}} p(y|d)\log(p(y|d))dy \\
&= \int_{\Theta}\left[\sum_{c=1}^{C} p(\theta)p(y_c|\theta,d)\log(p(y_c|\theta,d))\right]d\theta - \sum_{c=1}^{C} p(y_c|d)\log(p(y_c|d)) \\
&\approx \frac{1}{N}\sum_{n=1}^{N}\sum_{c=1}^{C} p(y_c|\theta_n,d)\log\left(p(y_c|\theta_n,d)\right) - \sum_{c=1}^{C}\left[\left(\frac{1}{N}\sum_{n=1}^{N} p(y_c|\theta_n,d)\right)\log\left(\frac{1}{N}\sum_{n=1}^{N} p(y_c|\theta_n,d)\right)\right]
\end{aligned} \tag{11.9}$$

where $\theta_n \sim p(\theta)\ \forall n \in 1,\ldots,N$ and y_c are fixed constants. Here our estimate is a deterministic function of a number of separate Monte Carlo estimators. Because there is no trivial canceling of terms and there are no issues with continuity of the function because $x\log(x) = 0$, and each $p(y_c|\theta_n,d) \leq 1$, no complications originate from this function application and the MSE of our estimator will converge at the standard MC error rate of $O(1/N)$. This requires $T = O(NC)$ calculations and so we can express the per evaluation convergence rate as $O(C/T)$. This compares to an optimal rate of $O(1/T^{2/3})$ for (11.8). Though relatively straightforward, to the best of our knowledge, this superior estimator has not appeared in the literature prior to our own work [Rainforth et al., 2017a; Vincent and Rainforth, 2017].

11.3 Automating Sequential Design Problems

We have thus far assumed that there is no previous data (i.e. design-outcome pairs). Though this static experimental design setup is of use in its own right, the real potential of BED is not realized until one considers using it in *sequential* settings. Here BED provides a framework for adaptively making an optimal series of decisions in an online fashion in the presence of uncertainty. For example, imagine a psychology trial where we ask a participant a series of questions to learn about certain behavior characteristics. If a human is conducting this experiment they are likely to adapt the questions they ask as they learn about the participant to try and

maximize the information gathered. Similarly, a detective conducting an investigation will adapt the questions they ask about a suspect as they learn information from previous answers. Sequential BED provides a mathematical framework for reasoning about and optimizing such processes, thereby providing a means of developing effective machine learning systems to carry out such tasks. It is quite incredibly general and can, at least in theory, be applied to any situation where we aim to sequentially gather information. For example, in Section 8.3.1.4 we described strategies for Bayesian optimization which explicitly use the BED equations to sequentially gather information about the location of the optimum. One could envisage similarly using the framework for conducting proposal adaptation for inference problems. We could use it in vision tasks to prioritize where to look in a scene. The number of potential applications is endless.

From a mathematical perspective, we can generalize to the sequential design setting by incorporating data in the standard Bayesian fashion such that at experiment iteration t, we replace $p(\theta)$ with $p(\theta|d_{1:t-1}, y_{1:t-1})$, where $d_{1:t-1}$ and $y_{1:t-1}$ are respectively the designs and outcomes at previous iterations. The likelihood $p(y_t|\theta, d_t)$, on the other hand, is unchanged (presuming it is a parametric distribution) as, conditioned on θ and d, the current outcome is independent of the previous data. Putting this together, we get that the expected information gain criteria for the sequential case is

$$\begin{aligned}\bar{U}_t(d) = &\int_{\mathcal{Y}} \int_{\Theta} p(\theta|d_{1:t-1}, y_{1:t-1})\, p(y_t|\theta, d_t) \log(p(y_t|\theta, d_t)) d\theta dy_t \\ &- \int_{\mathcal{Y}} p(y_t|y_{1:t-1}, d_{1:t}) \log(p(y_t|y_{1:t-1}, d_{1:t})) dy_t.\end{aligned} \tag{11.10}$$

We can now see that these terms are the same as in the non-sequential case, except that expectations are taken with respect to $p(\theta|d_{1:t-1}, y_{1:t-1})$ rather than $p(\theta)$. Therefore, we can use the same Monte Carlo estimators (11.8) and (11.9), but instead drawing $\theta_n \sim p(\theta|d_{1:t-1}, y_{1:t-1})$.

In the rest of this section, we will outline a framework for automating such sequential adaptive design problems. This takes on a somewhat probabilistic programming flavor, as the aim is to develop a system where the user provides only information about the model and design space, with the next chosen design calculated automatically when provided with the outcome of the last. However, as we will discuss in Section 12.4, this actually goes beyond the capabilities of current PPS. What we present is by no means a complete solution to this problem and certainly does not represent a PPS itself, but it does perhaps provide an important step towards building such a hypothetical system. Indeed the framework has already been applied to provide automated sequential BED for a very restricted model class existing in the form of the DARC toolbox [Vincent and Rainforth, 2017] introduced in Section 11.4.

11.3.1 Parameter Inference

Unlike $p(\theta)$, it is typically not possible to evaluate, or sample from, $p(\theta|d_{1:t-1}, y_{1:t-1})$ exactly – we need to perform Bayesian inference to estimate it. Noting the implicit assumption of the likelihood model that observations are independent of one other given the parameters we have

$$p(\theta|d_{1:t-1}, y_{1:t-1}) \propto p(\theta) \prod_{i=1}^{t-1} p(y_i|\theta, d_i) \tag{11.11}$$

which is a typical Bayesian inference problem, such that we can use Monte Carlo inference to generate the required samples θ_n. Note that this setup is very general. For example, there might be a term in θ for each round of the experiment, leading to a non-Markovian state space model as per Section 6.1.1. There are many suitable candidates for the required inference such as those discussed in Chapters 5 and 6. The relative merit of these will depend on characteristics of the problem such as multi-modality, the dimensionality of θ, and the total number of questions that will be asked. For the DARC toolbox introduced later we will use population Monte Carlo (PMC) [Cappé et al., 2004]. However, our contribution here is independent of the choice of inference algorithm, provided that algorithm can be appropriately automated for the range of problems one wishes to run, as is the case for a PPS. One salient feature of note though is that, by its nature, the inference problem requires a series of targets to be approximated and so inference methods that can effectively exploit the approximation of the previous target (such as particle-based methods, see Chapter 6) are likely to be particularly effective.

11.3.2 Design Optimization

We have now derived an estimator for $\bar{U}_t(d)$ and demonstrated how we can generate the samples required to evaluate this estimator. We will now look at how to optimize for d. Our problem requires the optimization of an intractable expectation and thus shares substantial similarity to the problems considered in Chapter 9. Indeed BOPP could form a suitable approach to estimating d^* at each iteration for many problems. However, BOPP is a somewhat heavy duty estimation algorithm that may not be suitable for problems that are relatively simple, but for which speed is paramount. Many of the possible applications of our approach, such as the DARC toolbox, require rerunning in real-time with human participants, for which BOPP would thus be inappropriate.[2] We instead construct a new approach that exploits the fact that (11.9) need not generate distinct θ_n to evaluate different designs. Our novel method combines ideas from both the bandit and Monte

[2]Nonetheless, if the surrogate in BOPP was replaced with a more lightweight regressor, e.g. a random forest [Hutter et al., 2011], then the resulting algorithm could prove a viable alternative even under strict time pressure.

Algorithm 11.1 Design optimisation

Inputs: Reservoir $\{\theta_m\}_{m=1:M}$, likelihood $p(y|\theta, d)$, candidate designs $\mathcal{D} = \{d_k\}_{k=1:K}$, number of particles N, number of steps L, annealing schedule $\gamma : \mathbb{Z}^+ \to \mathbb{R}$, minimum selection probability ρ

Outputs: Chosen design d^*

$\hat{U}_k \leftarrow 0 \quad \forall k \in \{1, \ldots, K\}$ ▷ Initialize estimates

$q_k \leftarrow 1/K \quad \forall k \in \{1, \ldots, K\}$ ▷ Initialize selection probability

for $\ell = 1 : L$ **do**

  $\{n_k\}_{k=1:K} \sim \text{MULTINOMIAL}(N, \{q_k\}_{k=1:K})$ ▷ Sample number of new samples to add[3]

  for $k = 1 : K$ **do**

    $\hat{\theta}_j \leftarrow \text{DISCRETE}\left(\{\theta_m\}_{m=1:M}\right) \quad \forall j = 1, \ldots, n_k$ ▷ Draw n_k samples from reservoir

    $\hat{U}_k \leftarrow \text{UPDATE}\left(\hat{U}_k, \{\hat{\theta}_j\}_{j=1:n_k}, p(y|\theta, d), d_k\right)$ ▷ Use new samples to refine estimate

  end for

  $\tilde{q}_k \leftarrow \hat{U}_k^{\gamma(\ell+1)} \quad \forall k = 1, \ldots, K$ ▷ Unnormalized selection probability = annealed $\hat{U}_k$

  $Z \leftarrow \sum_{k=1}^{K} \tilde{q}_k$ ▷ Normalization constant

  $q_k \leftarrow \max\left(\frac{\rho}{N}, \frac{\tilde{q}_k}{Z}\right) \quad \forall k = 1, \ldots, K$ ▷ Ensure probabilities are all at least minimum value

  $q_k \leftarrow \frac{q_k}{\sum_{k=1:K} q_k} \quad \forall k = 1, \ldots, K$ ▷ Renormalize

end for

$k^* \leftarrow \arg\max_{k \in \{1,\ldots,K\}} \hat{U}_k$ ▷ Best design is one with highest $\hat{U}_k$

return d_{k^*}

Carlo literatures [Amzal et al., 2006; Neufeld et al., 2014] by adaptively allocating resources to a set of candidate designs. We will presume that there are some fixed number of candidate designs $\mathcal{D} = \{d_k\}_{k=1:K}$ which could be randomly (or quasi-randomly) sampled up front if the design space is continuous. This assumption is made because the algorithm was designed with a particular application in mind, namely the DARC toolbox for which K is generally not more than a few thousand. We note, though, that it should be possible to relax this assumption by incorporating MCMC transitions in a manner akin to [Amzal et al., 2006].

The underlying idea of the approach is that we are willing to accept higher variance in the integral estimates for lower utility designs as this error is unlikely to lead to an incorrect estimation of the optimum. Therefore, we desire to take more Monte Carlo samples, and thus achieve a more accurate estimate, for the designs we estimate to have a higher chance of being the optimal design, given the current noisy estimates. At a high level, this is similar to the idea of Thompson sampling where a candidate is sampled in proportion to its probability of being the best [Thompson, 1933], a technique commonly used in the bandit literature [Agrawal and Goyal, 2012].

An overview of our approach is given in Algorithm 11.1. As this shows, our algorithm uses a *reservoir* of samples $\{\theta_m\}_{m=1:M}$ instead of drawing samples from $p\left(\theta|d_{1:t-1}, y_{1:t-1}\right)$ on-demand. In short, we draw M samples from the posterior upfront to create the reservoir, then

[3] One could use systematic "resampling" here too as per Section 5.3.2.4.

instead of sampling directly from $p\left(\theta|d_{1:t-1}, y_{1:t-1}\right)$ during the design optimization, we draw a sample from the reservoir. Careful coding (not shown in the algorithm block) ensures that each sample in the reservoir is not used more than once in any particular estimate $\hat{U}_k$, while by using a reservoir that is larger than the number of samples we will need for any individual estimate, we ensure that the reservoir itself does not cause any approximation in the posterior (though it can cause correlation between the $\hat{U}_k$). The use of a reservoir is a key innovation of our approach as it allows us to modularize the parameter inference and design optimization problems: the inference algorithm produces a reservoir as output and the optimizer can then run without having to refer back to the inference algorithm.

The second key innovation of our approach is that rather than calculating each estimate $\hat{U}_k$ in one go, we exploit the intermediate information from incomplete evaluations to reallocate computational resources in a manner that has parallels with SMC, SMC search [Amzal et al., 2006], annealed importance sampling [Neal, 2001], adaptive Monte Carlo via bandit allocation [Neufeld et al., 2014], and the Hyperband algorithm [Li et al., 2016]. Imagine that we are considering K designs $\mathcal{D} = \{d_k\}_{k=1:K}$ with a total sample budget $\bar{N}K$ and decide to use the same number of samples, $\bar{N}$, to calculate an estimate $\hat{U}_k$ for the utility of each design. After we have taken say n samples for each estimate, where $1 \leq n < \bar{N}$, then we will have a set of intermediate estimates $\hat{U}^n_{1:K}$ for each utility. Now remembering that our aim is to establish which design is best (i.e. which $\bar{U}_k$ is largest), we note that our intermediate estimates convey valuable information – they show that some designs are more likely to be optimal than others. Consequently, sticking to our original plan and using the same number of samples, $\bar{N} - n$, to complete our final estimates will be inefficient as some designs will be highly unlikely to be optimal and therefore it is pointless to spend a noticeable proportion of our computational budget to carefully refine their estimates. It is clearly more efficient, in terms of the final optimization problem, to take relatively more samples for the promising designs, so that we can better distinguish between their relative utilities, even if this results in very poor estimates for the low utility designs. This is the key idea of our approach – to adaptively change the number of samples used to estimate the utilities for different designs, based on the probability the design may end up being optimal given its intermediate estimates.

More specifically, given a total budget of $NL = \bar{N}K$ samples for all our estimates, then we will carry out L rounds of sampling, where at each round we adaptively allocate N samples between the different estimators in proportion to how relatively promising their respective designs appear. To do this, we *sample a number of samples* to take for each design at each round in proportion to an annealed version of its utility estimate (line 4 of Algorithm 11.1).

Namely, if $q_{k,\ell}$ is the probability each sample from our budget is assigned to design k at round ℓ and $\hat{U}_{k,\ell}$ is the corresponding running estimate for the utility of that design, then we have (line 9 of Algorithm 11.1)

$$q_{k,\ell} \propto (\hat{U}_{k,\ell})^{\gamma(\ell)} \tag{11.12}$$

where $\gamma : \mathbb{Z}^+ \to \mathbb{R}$ is an annealing function (see next paragraph). Thus we will, on average, add $\mathbb{E}[n_{k,\ell}] = q_{k,\ell}N$ samples of θ to the estimate for $\bar{U}_k$ at round ℓ (lines 6 and 7 of Algorithm 11.1). By storing appropriate running estimates (e.g. for $p(y|d_k)$), the estimates can be updated with the n_k new samples from the reservoir $\{\hat{\theta}_j\}_{j=1:n_k}$ at each round. This corresponds to the UPDATE function in Algorithm 11.1.

The purpose of the annealing function is to control how aggressive our algorithm is in its allocation of computational resources to promising designs. At the early rounds, we are quite uncertain about the relative utilities and therefore wish to allocate samples fairly evenly. At the later rounds, we become more certain about the estimates so we become more aggressive about concentrating our efforts on promising designs. Therefore $\gamma(\ell)$ is set to increase with ℓ.

We finish the subsection by noting a small, but important, subtlety of our method. As introduced so far, our algorithm is not guaranteed to converge. For example, it might be that $\hat{U}_{k^*} = 0$ after the first round of sampling where k^* indicates the true optimal design and $\bar{U}_{k^*} \neq 0$. Presuming that another design has a non-zero estimate at this point, sampling naïvely from (11.12) would mean that no additional samples are ever added to $\hat{U}_{k_*}$, even if infinite rounds are undertaken, meaning that the true optimum will be missed. To guard against this, we introduce a new parameter $0 < \rho \leq 1$, with lines 9 to 12 in Algorithm 11.1 ensuring that each $q_{k,\ell} > \frac{\rho}{2N}$ (see Vincent and Rainforth [2017]), which in turn ensures that $\mathbb{E}\left[\sum_{\ell=1}^{L} n_{k,\ell}\right] > \rho L/2$. Therefore, for any finite value of ρ, each estimate will, in expectation, be assigned infinitely many samples as $L \to \infty$, regardless of the true utilities and the values of $\gamma(\ell)$.[4] As we show in the next section, we can use this to prove the convergence of the algorithm, in the sense of guaranteeing the optimal design will be found given infinite computational budget.

11.3.3 Tying it All Together

In the previous sections, we have shown how to construct an appropriate target for our design optimization, estimate this target for a particular design given samples from the posterior on parameters, produce the required approximate samples for this posterior, and use these samples

[4]Note that one should, if desired, be able to instead use an upper confidence bounding strategy [Auer, 2002] to produce a zero-regret strategy whilst maintaining convergence

Algorithm 11.2 Sequential BED

Inputs: Prior $p(\theta)$, likelihood $p(y|\theta, d)$, number of experiment iterations T, candidate designs $\mathcal{D}$
Outputs: Design-outcome pairs $\{d_t, y_t\}_{t=1:T}$, posterior samples $\{\theta_m\}_{m=1:M}$

$\theta_m \sim p(\theta) \quad \forall m = 1, \ldots, M$
for $t = 1 : T$ **do**
  $d_t \leftarrow$ DESIGNOPTIMIZATION $(\{\theta_m\}_{m=1:M}, p(y|\theta, d), \mathcal{D})$ ▷ Find next step optimal design
  $y_t \leftarrow$ RUNEXPERIMENT(d_t) ▷ Get outcome using design d_t
  $\mathcal{D} \leftarrow \mathcal{D} \backslash d_t$ ▷ Eliminate new design from candidate set (optional)
  $\{\theta_m\}_{m=1:M} \leftarrow$ INFERENCE $(p(\theta), p(y|\theta, d), \{d_i, y_i\}_{i=1:t}, \{\theta_m\}_{m=1:M})$ ▷ Update the posterior
end for
return $\{d_t, y_t\}_{t=1:T}$, $\{\theta_m\}_{m=1:M}$

to find the optimal design. Algorithm 11.2 shows how these components fit together to produce a framework for automated online sequential design optimization. Algorithm parameters have been omitted for clarity, but we note that it should, in general, be possible to set these to default values if required, e.g. using the general purpose inference strategies taken by PPSs. We finish with the following theoretical result that shows that the method is guaranteed to return the optimal design at each iteration in the limit of large computational resources.

Theorem 11.1. *Assume that each $\bar{U}_t$ is a measurable function and that all other requirements for consistency of the chosen inference algorithm are satisfied.*[5] *Given a finite number of candidate designs $\{d_k\}_{k=1:K}$, fixed parameters $\rho > 0$ and $N \in \mathbb{N}^+$, and a function $\tau : \mathbb{R} \to \mathbb{R}$ such that $\tau(L) \geq NL \ \forall L$, then using Algorithms 11.1 and 11.2 to choose design outcome pairs with $M = \tau(L)$ ensures that the chosen design at each iteration d_t satisfies*

$$\bar{U}_t(d_t) = \max_{d \in \mathcal{D}} \bar{U}_t(d) \tag{11.13}$$

with high probability in the limit $L \to \infty$, where $\bar{U}_t(d)$ is defined as per (11.10).

Proof. We start by noting that by the definition of $\tau(\cdot)$ then $L \to \infty$ also predicates that $M \to \infty$ and so by assumption our Monte Carlo estimates made using the output of our inference algorithm converge in probability (noting $M \geq NL$ also ensures the reservoir is large enough to not need to reuse samples).

As the $n_{k,\ell}$ are not independent, we break each down into two terms $n_{k,\ell} = m_{k,\ell} + r_{k,\ell}$ where $m_{k,\ell} \sim$ BINOMIAL$(N, \frac{\rho}{2N})$ and $r_{k,\ell} \sim$ BINOMIAL$(N, q_{k,\ell} - \frac{\rho}{2N})$ (which induces the appropriate distribution on $n_{k,\ell}$). For any given k, all the $m_{k,\ell}$ are mutually independent and $P(m_{k,\ell} \geq 1) = 1 - (1 - \frac{\rho}{2N})^N$ ($\to 1 - \exp(-\rho/2)$ as $N \to \infty$). Therefore, for any possible

[5] If the chosen inference algorithm also provides almost sure convergence, the result also holds almost surely.

N (including $N \to \infty$) we have $\sum_{\ell=1}^{\infty} P(m_{k,\ell} \geq 1) = \infty$ and so by the second Borel-Cantelli lemma, the event $m_{k,\ell} \geq 1$ occurs infinitely often with probability 1. Thus, as each $r_{k,\ell} \geq 0$, it must also be the case that $P(\lim_{L\to\infty} \sum_{\ell=1}^{L} n_{k,\ell} = \infty) = 1$.

Combining these two results, we have convergence of each of the K estimates $\hat{U}_k$ at each iteration and so we must choose the optimal design (or one the equally best designs if there is a tie) with high probability in the limit $L \to \infty$. □

11.4 The DARC Toolbox

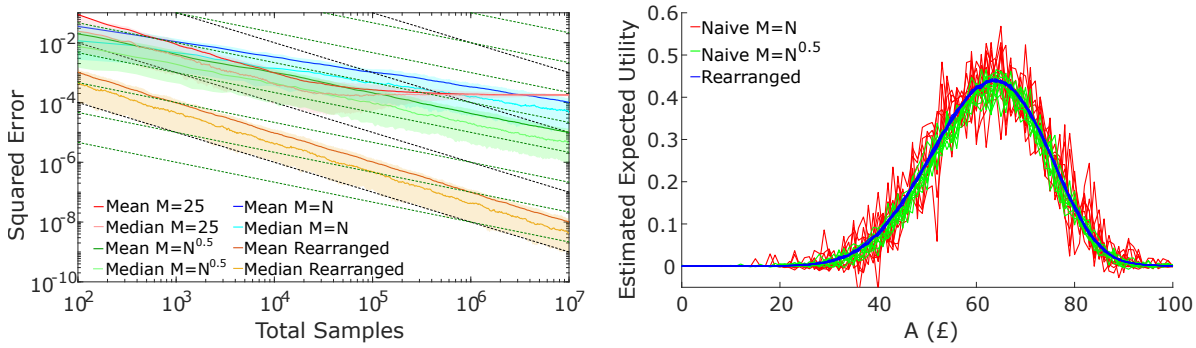
We now demonstrate the utility of our approach by considering its application to automating the adaptive design psychology experiments via the DARC (Delay And Risky Choice) toolbox introduced in Vincent and Rainforth [2017].[6] We will only provide a short introduction here and refer the reader to Vincent and Rainforth [2017] for further details.

The DARC toolbox allows users to encode their own (or use one of the provided) models for human *discounting* behavior, whereby psychologists want to learn how delays and uncertainties affect the subjective value people place on (typically monetary) rewards or costs. Given a model, the toolbox uses the framework from the previous section to automate in an online fashion both the inference of the model parameters and the sequential adaptation of the experiment itself through choosing the questions the participant is asked. It, therefore, automates the full run-time experimental pipeline by using the responses from previous questions to perform inference and update the internal representation of the participant and then using this information to ask the most informative questions for a particular participant as per the BED equations. The technical innovations introduced earlier in this chapter are essential in allowing this to be done sufficiently efficiently to permit real-time usage required for deployment with real participants.

A common style of experiment for the toolbox comprises of asking questions of the form *"Would you prefer £A now, or £B in D^b days?"*, where we desire to choose the question variables $d = \{A, B, D^b\}$ in the manner that will give the most incisive questions. One possible participant model, given in [Vincent, 2016], presumes that participants have parameters $\theta = \{\log k, \alpha\}$, where $\log k \sim \mathcal{N}(-4.5, 0.5^2)$ represents a log discount rate and $\alpha \sim \text{GAMMA}(2, 0.5)$ (using the shape-rate parameterization) a variability, and the following response model

$$y \sim \text{Bernoulli}\left(0.01 + 0.98\ \Phi\left(\frac{1}{\alpha}\left(\frac{B}{1 + kD^b} - A\right)\right)\right) \tag{11.14}$$

[6] Code available at `http://github.com/drbenvincent/darc-experiments-matlab`.



(a) Convergence of BED

(b) Estimated expected utilities

Figure 11.1: (Left) convergence of both NMC and our reformulated estimator (11.9) for the BED problem. A ground truth estimate was calculated using a single run of the reformulated estimator with 10^{10} samples. Results are averaged over 1000 independent runs, while shaded regions give the 25%-75% quantiles. We see that the theoretical convergence rates (shown by the dashed lines) are observed in all cases, with the advantages of the reformulated estimator particularly pronounced. (Right) estimated expected utilities $\bar{U}(d)$for different values of one of the design parameters $A \in \{1, 2, \ldots, 100\}$ given a fixed total sample budget of $T = 10^4$. Here the lines correspond to 10 independent runs, showing that the variance of (11.8) is far higher than (11.9).

where $y = 1$ indicates choosing the delayed response and Φ represents the cumulative normal distribution. As more questions are asked, the distribution over the parameters θ is updated in the standard Bayesian fashion, such that the most optimal question to ask at a particular time depends on the previous questions and responses.

Before considering the full BED pipeline, we will first examine the convergence and empirical performance of our reformulated estimator (11.9) compared to the naïve alternative (11.8). For simplicity, we will consider the case where $B = 100$ and $D^b = 50$ are fixed and we are only choosing the delayed value A. We first examine the convergence in the estimate of $\bar{U}(d)$ for the case $A = 70$, for which Figure 11.1a demonstrates similar behavior for the naïve NMC estimator as seen in the experiments of Chapter 10 and, as expected, substantial improvements for the reformulated estimator in the form of a $O(1/N)$ convergence rate.

We next consider setting a total sample budget $T = 10^4$ and look at the variation in the estimated values of $\bar{U}(d)$ for different values of A for the two methods as shown in Figure 11.1b. This shows that the improvement in MSE leads to clearly visible improvements in the characterization of $\bar{U}(d)$ that will translate to improvements in seeking the optimum.

Finally, we examine the performance of the DARC toolbox compared to alternatives in the psychology literature [Kirby, 2009; Frye et al., 2016] as shown in Figure 11.2. Using the same likelihood model as (11.14), we fix $\alpha = 2$ and set the prior on k as $\log k \sim \mathcal{N}(-4.5, 0.5^2)$. We fixed $B = 100$ and then use three different methods for sequentially selecting designs

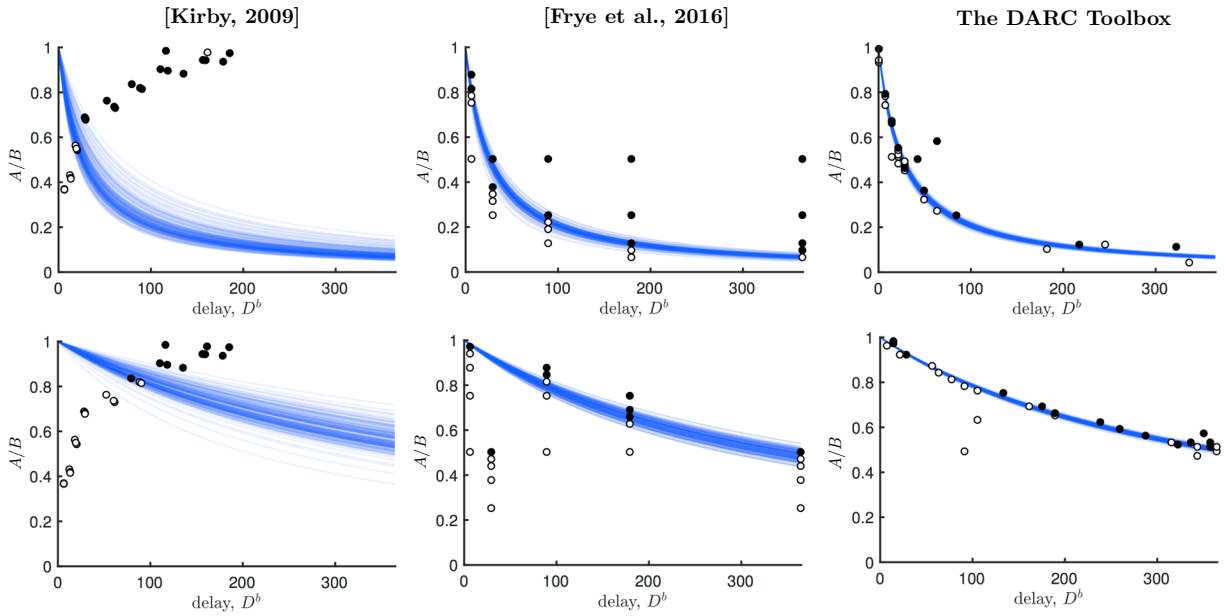


Figure 11.2: Comparison of the DARC toolbox to alternative methods for two behavior styles: major depressive disorder (top) which indicates a relatively strong desire for immediate rewards ($k = 0.04$) and anorexia nervosa (bottom) which indicates relatively low discounting rate ($k = 0.0028$). Lines show posterior samples from the final learned predictive distribution for the indifference curve (i.e. the curve along which the participant is equally likely to choose the immediate or delayed reward), circles represent questions where the delayed reward was preferred, and the filled dots correspond to the questions were the immediate reward was preferred. The columns left to right corresponding to using the approaches of Kirby [2009] which uses pre-fixed questions rather than using online adaptation, Frye et al. [2016] which uses a heuristic approach for choosing the best questions adaptively, and using our DARC toolbox which chooses questions using our sequential BED approach. See Vincent and Rainforth [2017] for further details.

$d_t = \{A_t, D_t^b\}$, for two different hypothetical participants with respective ground truth parameters of $k = 0.04$ and $k = 0.0028$. Responses are generated presuming the likelihood model is correct and sampling directly from (11.14) using the ground truth parameters. We then plot a characterization of the posterior predictive after 27 questions have been asked. All methods are capable of working in real time, taking about a second or less to choose the design at each iteration when run on a mid-range laptop. Though hardly a rigorous performance assessment, these results are still demonstrative of the potential utility of our method. The non-adaptive scheme of Kirby [2009] always asks the same questions regardless of the participant's responses. The heuristic method of Frye et al. [2016] does better, but is still somewhat wasteful. The DARC toolbox, on the other hand, takes only a couple of questions before achieving a reasonable representation of the response surface, such that almost all of the questions asked are clearly pertinent and the final posterior predictive distribution has far lower entropy than the other approaches.

12

Discussions, Conclusions, and Future Directions

In this thesis, we have introduced a *Bayesian* approach to machine learning and demonstrated how *probabilistic programming systems* (PPSs) provide an exciting framework for encoding rich and expressive models in the Bayesian framework and then automating the process of drawing inferences from these models. We explained how the long-term aim of probabilistic programming is to automate the process of drawing inferences from any simulator the user might write, thereby streamlining the modeling process and breaking down barriers to carrying out such analysis. We have made steps towards this eventual aim by improving the inference engines used by PPSs and expanding the probabilistic programming framework beyond its conventional Bayesian inference setting to include, for example, mixed inference-optimization problems, thus allowing automation of tasks such as model learning and engineering design.

We highlighted the importance of taking a Bayesian approach in scenarios where one has access to substantial *prior knowledge* or domain-specific expertise, but noted that *discriminative* machine learning approaches can be more effective, and typically substantially easier to implement, when one has access to large quantities of data, but little prior information. When opting for a Bayesian approach, we explained that two key challenges are in writing good models and solving the resultant *Bayesian inference* problem. PPSs can help in addressing both of these challenges. Firstly, they provide an expressive framework that allows users to write models true to their assumptions in a manner more in line with conventional scientific coding. Secondly, they remove the burden of inference from the user by providing general purpose *inference engines* capable of operating on arbitrary models. Together these mean that PPSs make effective statistical methods accessible to non-experts, while streamlining the process of writing models and inference algorithms for seasoned researchers. However, from a developers perspective, designing inference engines for PPSs is challenging because of the need for them to work on any model. This problem is especially pronounced for so-called *universal* PPSs, which, as we explained, place almost no restrictions on the models the user can write. Consequently, universal PPSs will rarely produce state-of-the-art performance on any particular task, though by directly

using the source code of the target, they are still often able to exploit many salient features of models and thus still provide effective inference for wide array of tasks. Their main utility is thus in *automation* and in allowing models that would difficult, or even impossible, to encode otherwise. In particular, they allow us to go beyond the confines of conventional Bayesian modeling, allowing, for example, one to *nest* inference problems within one another.

There have been two core thrusts to the novel contributions of this thesis: improving the inference engines for PPSs and extending the range of problems to which they can be applied. For the former, we introduced the *interacting particle Markov chain Monte Carlo* (iPMCMC) algorithm [Rainforth et al., 2016c], a particle-based inference method delivering better per-sample performance than existing methods for a given memory budget, while also introducing the opportunity for parallelization. We then showed how iPMCMC is suitable for inference in universal PPSs, detailing its implementation in the PPS *Anglican* [Tolpin et al., 2016]. Towards the aim of increasing the range of problems to which PPSs can be applied, we have firstly introduced *Bayesian optimization for probabilistic programs* (BOPP) [Rainforth et al., 2016b], providing the first framework for mixed inference-optimization problems in probabilistic programs, allowing things such as model learning and principled engineering design to be automated in the same manner as inference. BOPP also provides a significant contribution to the Bayesian optimization literature, constituting a convenient and efficient package for solving conventional optimization problems by exploiting the source code of its target to provide innovations in problem-independent hyperpriors, unbounded optimization, and implicit constraint satisfaction. We have further undertaken theoretical work looking into the convergence of *nested Monte Carlo* (NMC) [Rainforth et al., 2017a], for which *nested inference* problems are a particular example. We demonstrated that the convergence of NMC is possible, but that some additional assumptions need to be satisfied, e.g. the number of samples used for *each* inner estimate needs to be driven to infinity. Our results go beyond the cases considered by previous results and, in particular, apply to cases of repeated nesting, as can occur in PPSs. We then carefully considered the implications of our results for PPSs and provided recommendations about how to ensure consistency is maintained. Our final contribution to extending PPSs was in applying our theoretical results from nested estimation to consider the specific problem of Bayesian experimental design, deriving an improved estimator for discrete problems and outlining a framework for automating the solution of adaptive sequential design problems, as might be experienced in, for example, psychological trials. Our technical innovations were applied to automating a specific class of problems relating to delay and probability discounting experiments

by introducing the DARC toolbox, perhaps providing a first step in demonstrating how one might be able to construct a general purpose language for automating adaptive design problems.

A natural question is now where do we go next? In addition to the obvious ongoing challenges of improving inference and range of problems covered by PPSs, perhaps the most clear-cut extensions for our work is in its direct application. We have introduced various frameworks which, while someway off our lofty long-term aims of tractably and automatically solving almost any problem the user might write, are nonetheless clearly already useful for a number of existing applications. More generally, the biggest potential impact of PPSs is arguably not in what they provide for seasoned machine learning or statistics researchers, but in the democratization of advanced statistical approaches to anybody with sufficient scientific programming background to write a stochastic simulator. The best example of this is perhaps the BUGS system Spiegelhalter et al. [1996], which has seen effective widespread use across many fields. More recently, Stan [Carpenter et al., 2015] and PyMC [Salvatier et al., 2016] are seeing particularly widespread usage that extends well beyond the machine learning community. Though some universal PPSs are already relatively mature and user-friendly with a small base of users, they are arguably still more at the stage of predominantly being a framework for probabilistic programming research, than a commonly used tool by applied communities. Perhaps this is not surprising given their inference engines inevitably tend to be less efficient than more inference-driven packages. However, we believe that as inference engines improve and people's modeling ambitions increase, the flexibility that such systems provide will become increasingly important, particularly if we want to achieve our long-term aims regarding arbitrary simulators. Existing applications requiring this flexibility are already to be found in the form of Bayesian nonparametrics [Dhir et al., 2017] and theory-of-mind style modeling [Stuhlmüller and Goodman, 2014].

We now finish by briefly discussing some interesting high-level questions and outlining some more specific possible future directions for research.

12.1 Shouldn't We Just Use Deep Learning Instead?

In a word, no. As successful as deep learning, or artificial neural network (ANN) training [Bishop, 1995] as it was known before the hype, has been in recent years, it is not a panacea for machine learning. Instead, it is a particular discriminative machine learning approach that has been especially successful on certain kinds of machine learning tasks, namely those with huge quantities of, usually high dimensional, data with rich underlying structure. To suggest that ANNs are universally dominant even among discriminative machine learning tasks is somewhat

preposterous, e.g. they are often, if not usually, inferior to Gaussian processes on low-dimensional, small-data regime, tasks, and still arguably trail behind decision tree ensemble approaches for "out-of-the-box" usage on, for example, generic classification tasks [Rainforth and Wood, 2015]. Perhaps more significantly, many problems call for a generative, rather than discriminative, approach, particularly when data is scarce or significant prior information is available as we explained in Section 3.1. Interestingly, however, this is perhaps where a lot of the successes of ANNs actually originate. Compared to say random forests, ANNs constitute a very flexible and composable framework for discriminative machine learning and perhaps provide a means of indirectly imposing prior knowledge on the ANN through its structuring. A major weakness of declarative generative approaches, such as those employed by PPSs, is that we almost always have to impose more assumptions on the model then we would like – all models will inevitably be misspecified. For ANNs, on the other hand, when there is not an abundance of training data, one often struggles to impose enough assumptions, sometimes, for example, resorting to generating synthetic data to train the network [Von Ahn et al., 2008]. Doing this is, in effect, a somewhat convoluted, though often very effective, method for imposing prior information on the model [Le et al., 2017b]. From this perspective, the Bayesian approach, and in particular probabilistic programming, is perhaps also of significant consequence to the ANN literature [Gal, 2017]. In particular, there may be a bright future for methods which take a model-based approach, but without having assumptions as strict and prone to misspecification as conventional Bayesian modeling, for which ANNs are likely to be a highly useful tool [Siddharth et al., 2017].

12.2 Do We Need Random Numbers?

An interesting question was recently raised by the probabilistic numerics [Hennig et al., 2015] community about whether we should, at least as a long-term idealistic vision, look to do away with randomization entirely [Schober, 2017]. This in many ways corresponds to taking the Bayesian philosophy to its most extreme possible limit, by which we shun all notions of drawing from probability distributions and view probability only as a measure of uncertainty. Given the heavy use of Monte Carlo methods throughout this thesis, it perhaps comes as no surprise that we are vehemently opposed to this idea. Nonetheless, it is still an insightful question that deserves serious consideration. We here present arguments for why we believe randomization is essential on a fundamental level. The intention is that this, inevitably subjective, discussion should be seen as a pairing to [Schober, 2017], who provides the other side of the argument.

At a high-level, the crux of the argument against randomization is that, unless our aim is sampling itself, we always have a certain amount of information available and a final aim we are trying to achieve (e.g. calculating an expectation). Therefore, at least at a philosophical level, randomization only serves to add noise to this process. Similarly to the likelihood principle, one might argue that given the same data and model, we should always carry out the same calculations and thus always reach the same conclusions. Doing otherwise, the argument follows, must always be suboptimal, as it involves inconsequential information affecting the final decision. One compelling motivation for this is that, in certain situations, some deterministic methods can deliver substantially better convergence rates than Monte Carlo [Briol et al., 2015a; Caflisch, 1998], though these are typically crippled by the curse of dimensionality in a way that Monte Carlo is not.

Though eminently reasonable, our opinion is that this argument, which almost completely dismisses the entirety of frequentist statistics, is fundamentally flawed. Our argument can be broken down into four main reasons why we need, and always will need, random numbers:

Honesty and reliability – The alternative to randomness is almost always approximation, introducing bias that might be impossible to effectively quantify.

Lack of repeatability – Though it is always convenient when an experiment is repeatable, systematically returning the same incorrect answer is far more dangerous.

Composability – Methods such as Monte Carlo are ambivalent to how the samples are to be used, creating a scalable ability to modularize, compose and reuse. Non-composable systems, on the other hand, are doomed to fail for large frameworks through the curse of repeated nesting of estimators as we discussed in Chapter 10.

Speed and simplicity – Sometimes even if there is the information available to improve a computation, uncovering or incorporating that information in a principled manner may require substantially more computational power than not doing so.

While the latter two arguments are computational, the others are related to the following principle:

> *Once we have imparted all possible information upon a system, we must treat what is left as truly stochastic or introduce bias.*

In other words, imagine we can construct a problem in a manner that utilizes all available prior information, both in terms of the model itself and in how best to draw inferences from that model. Any remaining uncertainty is now, at least for all practical purposes, inherent to the problem and so any further attempt to remove randomness from the system is indirectly adding additional prior information that we did intend to impart. Furthermore, as soon as we make approximations to our model, we lose *calibration* in our estimate: we can estimate the

variance resulting from randomness through repetition, but there is usually no way to estimate the bias resulting from approximation.

As we explained in Section 3.4, frequentist statistics is all about *repeatability* and focuses on the fact that any data we collect is random in a many-worlds point of view: multiple possible datasets *could* have been generated. We demonstrated that there are many times when this frequentist approach is absolutely essential. For example, in medical tasks we need to make sure that our confidence intervals are actually correct when we repeat a procedure – we need guarantees on their calibration. This is not possible in a Bayesian framework as, by its very nature, it presumes the data is fixed and thus permits no concept of repetition. Once we realize that we sometimes need to take a frequentist approach, the need for randomness becomes obvious, as in the frequentist framework probability originates only through random variables. As powerful as the Bayesian framework is, it is inherently optimistic and subjective (e.g. there are always unknown unknowns), and as such, it can never really be the whole story. If we try to do away with randomness completely it is impossible to be objective – we have to choose our approximation, sometimes in an arbitrary fashion – and so we can never provide properly calibrated confidence intervals that are not prone to subjective interpretation.

One of its most powerful consequences of the honesty of Monte Carlo is the composability of estimates: give a sample from a marginal, one can generate a valid sample of the joint by sampling from the conditional. Similarly, if we have samples from a joint, we also have samples from the marginal distributions. This behavior is essential when composing different components into a greater system, as one can ensure each component takes in samples as inputs and outputs corresponding samples that can be used by the next process in the pipeline. If we instead make some deterministic approximation at each stage, we flaunt the flaw of averages and our biases will conflagulate to give a result that might be substantially different to the truth. If we stick with unbiased Monte Carlo estimation methods then although it is, of course, possible that our variance will explode, this can almost always be checked. If we do away with randomness, we might generate large errors without even knowing we have.

This lack of repeatability is actually an essential advantage of Monte Carlo. It is better for a system to give you a different wrong answer each time it is queried, particularly if these answers are unbiased, than to always give the same wrong answer. When we write scientific papers we run our experiments multiple times to show the variability in the results. Systems that always give the same answer and instead return a single subjective uncertainty estimate (e.g. a Gaussian process) are not generally trustworthy because there is no true calibration or sense of whether

the experimenter simply got lucky with a system that will not generalize. Furthermore, such a setting is very open to intentional or unintentional abuse. If we fix our randomness a priori and then adapt our algorithm until it works, we may well simply be over-fitting to what works best for that random number seed or approximation. To maintain scientific integrity, we have to report results on experiments that have not been tested during the design of our algorithm and which are tested with multiple different choices for arbitrary or subjective decisions, to show that those results are actually stable and representative. If we do away with randomness, we lose, or at the very least seriously hamper, this ability to repeat experiments in an objective manner.

To summarize, we need randomness because we do not know everything. In the Bayesian framework, we place distributions on what we are not sure about to reflect this lack of knowledge. A key part of this process is acknowledging that once we have done this, what is left is random. For example, if we have a sampler that converts $u \sim \text{UNIFORM}(0, 1)$ draws to samples from the posterior, then the u corresponding to the ground truth parameters is equally likely to be any value between 0 and 1 and is truly random. Yes, we might be able to impart more knowledge on problems that we currently do, such as by decorrelating our samples, but there always become a point where we have imparted everything we know, after which what is left is truly random.

12.3 Amortizing Inference

An interesting recent advancement in PPSs is the idea of *amortizing* the process of inference [Paige and Wood, 2016; Ritchie et al., 2016a; Le et al., 2017a]. The high-level idea is that we often repeatedly use the same models for multiple different datasets and therefore it can be wasteful to attempt to do inference from scratch each time. Instead, one can, before any data is actually seen, look to learn a regressor from the space of data to, for example, proposals, such that when any particular dataset is actually observed, one can use this regressor to produce an artifact to assist with the resulting inference. To do this, amortization methods exploit the fact that generative approaches specify a distribution over both data and parameters. It is thus theoretically possible to learn the form of the posterior as a function of the dataset by sampling parameter-data pairs from the joint distribution and then using these to train a regressor from data to some variables providing a characterization of posterior. In practice, doing this is can be very challenging – learning a posterior for any possible dataset is inevitably far harder than learning one for a particular dataset – but in applications where the same model is repeatedly used, the additional cost can be easily justified. Current methods share a number of weaknesses, such as being restricted by the fidelity of the underlying regressor, mostly only allowing importance sampling

for the final inference, and being potentially even more sensitive to model misspecification than conventional Bayesian approaches. Nonetheless, they represent a very interesting avenue for future research and offer the potential for substantial long-term impact. In particular, as current methods are already effective at amortizing inference for low-dimensional problems, the utility of such approaches might be dramatically increased if one can develop methods for breaking an overall inference problem down into a number of smaller problems that can be amortized separately and then usefully combined at run time.

12.4 A General Purpose Experimental Design Language

We finish by discussing an exciting and direct possible extension of our work: developing the framework we introduced in Chapter 11 into a general purpose tool for automating sequential design problems. The potential applications for such a system would be huge, ranging from designing clinical trials to smart online surveys and classic autonomous agent problems. In the same way all Bayesian modeling problems can be reduced to a common inference framework, such problems can be reduced to a common Bayesian experimental design framework. This suggests that they are highly suited to a probabilistic programming style approach. However, there are a number of reasons existing systems are not suited to such problems. Firstly, such problems require coincident inference and optimization. Our introduction of BOPP could be of help for overcoming this problem, but its computationally intensive nature means that it is perhaps not suitable for some scenarios, e.g. the psychological trials we considered in Section 11.4, for which the approach we outlined in Section 11.3, or an extension thereof, might be more suitable. If BOPP were to be used, suitable adjustments would also be necessary to account for *sequential* design problems. Secondly, Bayesian experiment design problems, in general, require the calculation of a nested estimation. For discrete outputs, we showed how a superior estimator can be derived and so it would be important to exploit this, when possible, in any hypothetical system. For continuous problems that cannot be rearranged, then as we explained in Section 10.7, we need to be careful to avoid the pitfalls of nested estimation, for which existing systems appear to need suitable adaptation. It may also to be beneficial to develop application specific estimation schemes for this setting, potentially exploiting the known form of the nonlinear mapping to devise methods for bias reduction. Finally, to make such a system user-friendly, it would be important to provide a convenient means of encoding a suitable design space alongside the model. It might, for example, be possible to do this in a similar manner as constraints were enforced in BOPP, noting that each iteration in a sequential design is effectively a maximum marginal likelihood problem.

Bibliography

Emile Aarts and Jan Korst. Simulated annealing and Boltzmann machines. 1988.

Martín Abadi et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

David H Ackley, Geoffrey E Hinton, and Terrence J Sejnowski. A learning algorithm for Boltzmann machines. *Cognitive science*, 9(1):147–169, 1985.

Shipra Agrawal and Navin Goyal. Analysis of Thompson sampling for the multi-armed bandit problem. In *Conference on Learning Theory*, pages 39–1, 2012.

Billy Amzal, Frédéric Y Bois, Eric Parent, and Christian P Robert. Bayesian-optimal design via interacting particle systems. *Journal of the American Statistical Association*, 101(474):773–785, 2006.

Christophe Andrieu and Gareth O Roberts. The pseudo-marginal approach for efficient Monte Carlo computations. *The Annals of Statistics*, pages 697–725, 2009.

Christophe Andrieu and Johannes Thoms. A tutorial on adaptive MCMC. *Statistics and computing*, 18(4): 343–373, 2008.

Christophe Andrieu, Arnaud Doucet, and Roman Holenstein. Particle Markov chain Monte Carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 2010.

Christophe Andrieu, Matti Vihola, et al. Convergence properties of pseudo-marginal Markov chain Monte Carlo algorithms. *The Annals of Applied Probability*, 25(2):1030–1077, 2015.

Andrew W Appel and Trevor Jim. Continuation-passing, closure-passing style. In *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, 1989.

Nachman Aronszajn. Theory of reproducing kernels. *Transactions of the American mathematical society*, pages 337–404, 1950.

Peter Auer. Using confidence bounds for exploitation-exploration trade-offs. *JMLR*, 2002.

Thomas Bäck. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press, 1996.

Renée Baillargeon. The acquisition of physical knowledge in infancy: A summary in eight lessons. *Blackwell handbook of childhood cognitive development*, 1(46-83):1, 2002.

A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. Automatic differentiation in machine learning: a survey. *arXiv preprint arXiv:1502.05767*, 2015.

Mark A Beaumont. Estimation of population growth or decline in genetically monitored populations. *Genetics*, 164(3):1139–1160, 2003.

Richard Bellman. *Dynamic programming*. Courier Corporation, 2013.

Richard E Bellman. *Adaptive control processes: a guided tour*. Princeton university press, 1961.

Denis Belomestny, Anastasia Kolodko, and John Schoenmakers. Regression methods for stochastic control problems and their convergence analysis. *SIAM Journal on Control and Optimization*, 2010.

Yoshua Bengio, Olivier Delalleau, and Nicolas L Roux. The curse of highly variable functions for local kernel machines. In *NIPS*, 2006.

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

James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: A CPU and GPU math compiler in Python. In *Proc. 9th Python in Science Conf*, pages 1–7, 2010.

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

Michael Betancourt. A conceptual introduction to Hamiltonian Monte Carlo. *arXiv preprint arXiv:1701.02434*, 2017.

Joris Bierkens, Paul Fearnhead, and Gareth Roberts. The zig-zag process and super-efficient sampling for Bayesian analysis of big data. *arXiv preprint arXiv:1607.03188*, 2016.

Hildo Bijl, Thomas B Schön, Jan-Willem van Wingerden, and Michel Verhaegen. A sequential Monte Carlo approach to Thompson sampling for Bayesian optimization. *arXiv preprint arXiv:1604.00169*, 2016.

Bingham, Jonathan P. Eli, Chen, Martin Jankowiak, Theofanis Karaletsos, Fritz Obermeyer, Neeraj Pradhan, Rohit Singh, Paul Szerlip, and Noah Goodman. Pyro, 2017. URL `https://github.com/uber/pyro`.

Christopher M Bishop. *Neural networks for pattern recognition*. Oxford university press, 1995.

Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *arXiv preprint arXiv:1601.00670*, 2016.

Benjamin Bloem-Reddy, Emile Mathieu, Adam Foster, Tom Rainforth, Hong Ge, María Lomelí, Zoubin Ghahramani, and Yee Whye Teh. Sampling and inference for discrete random probability measures in probabilistic programs. *NIPS Workshop on Advances in Approximate Bayesian Inference*, 2017.

Alexandre Bouchard-Côté, Sebastian J Vollmer, and Arnaud Doucet. The Bouncy Particle Sampler: A Non-Reversible Rejection-Free Markov Chain Monte Carlo Method. *arXiv preprint arXiv:1510.02451*, 2015.

George EP Box. Robustness in the strategy of scientific model building. *Robustness in statistics*, 1: 201–236, 1979.

George EP Box, William Gordon Hunter, and J Stuart Hunter. *Statistics for experimenters: an introduction to design, data analysis, and model building*, volume 1. John Wiley and Sons, 1979.

Stephen Boyd and Lieven Vandenberghe. Convex optimization, 2004.

Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

Leo Breiman et al. Statistical modeling: The two cultures (with comments and a rejoinder by the author). *Statistical science*, 16(3):199–231, 2001.

François-Xavier Briol, Chris Oates, Mark Girolami, and Michael A Osborne. Frank-Wolfe Bayesian quadrature: Probabilistic integration with theoretical guarantees. In *NIPS 28*. 2015a.

François-Xavier Briol, Chris J Oates, Mark Girolami, Michael A Osborne, and Dino Sejdinovic. Probabilistic integration: A role for statisticians in numerical analysis? *arXiv preprint arXiv:1512.00933*, 2015b.

Mark Broadie, Yiping Du, and Ciamac C Moallemi. Efficient risk estimation via nested sequential simulation. *Management Science*, 2011.

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

Adam D Bull. Convergence rates of efficient global optimization algorithms. *JMLR*, 12, 2011.

Robert Burbidge, Matthew Trotter, B Buxton, and S Holden. Drug design by machine learning: support vector machines for pharmaceutical data analysis. *Computers & chemistry*, 26(1):5–14, 2001.

Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519*, 2015.

Russel E Caflisch. Monte carlo and quasi-Monte Carlo methods. *Acta numerica*, 7:1–49, 1998.

Roberto Calandra, André Seyfarth, Jan Peters, and Marc Peter Deisenroth. Bayesian optimization for learning gaits under uncertainty. *Annals of Mathematics and Artificial Intelligence*, 76(1-2):5–23, 2016.

James V Candy. *Bayesian signal processing: classical, modern, and particle filtering methods*, volume 54. John Wiley & Sons, 2016.

Olivier Cappé, Arnaud Guillin, Jean-Michel Marin, and Christian P Robert. Population Monte Carlo. *Journal of Computational and Graphical Statistics*, 13(4):907–929, 2004.

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

Olivier Cappé, Randal Douc, Arnaud Guillin, Jean-Michel Marin, and Christian P Robert. Adaptive importance sampling in general mixture classes. *Statistics and Computing*, 18(4):447–459, 2008.

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

James Carpenter, Peter Clifford, and Paul Fearnhead. Improved particle filter for nonlinear problems. *IEE Proceedings-Radar, Sonar and Navigation*, 146(1):2–7, 1999.

George Casella and Christian P Robert. Rao-blackwellisation of sampling schemes. *Biometrika*, 83(1): 81–94, 1996.

Daniel R Cavagnaro, G J Aranovich, S M McClure, and M A Pitt. On the functional form of temporal discounting: An optimized adaptive test. 2014.

Kathryn Chaloner and Isabella Verdinelli. Bayesian experimental design: A review. *Statistical Science*, pages 273–304, 1995.

Siddhartha Chib and Ivan Jeliazkov. Marginal likelihood from the Metropolis–Hastings output. *Journal of the American Statistical Association*, 96(453):270–281, 2001.

Hugh A Chipman, Edward I George, and Robert E McCulloch. Bayesian CART model search. *Journal of the American Statistical Association*, 93(443):935–948, 1998.

Nicolas Chopin and Sumeetpal S. Singh. On particle Gibbs sampling. *Bernoulli*, 2015.

Nicolas Chopin, Pierre E Jacob, and Omiros Papaspiliopoulos. SMC2: an efficient algorithm for sequential analysis of state space models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 75(3):397–426, 2013.

Emile Contal, David Buffoni, Alexandre Robicquet, and Nicolas Vayatis. Parallel Gaussian process optimization with upper confidence bound and pure exploration. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 225–240. Springer, 2013.

Emile Contal, Vianney Perchet, and Nicolas Vayatis. Gaussian process optimization with mutual information. In *ICML*, pages 253–261, 2014.

Gregory F Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial intelligence*, 42(2-3):393–405, 1990.

Robert Cornish, Frank Wood, and Hongseok Yang. Efficient exact inference in discrete Anglican programs. 2017.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

Radu V Craiu and Xiao-Li Meng. Perfection within reach: exact MCMC sampling. *Handbook of Markov Chain Monte Carlo*, pages 199–226, 2011.

Drew Creal. A survey of sequential Monte Carlo methods for economics and finance. *Econometric reviews*, 31(3):245–296, 2012.

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

Paul Dagum and Michael Luby. Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial intelligence*, 60(1):141–153, 1993.

Marc Peter Deisenroth, Gerhard Neumann, Jan Peters, et al. A survey on policy search for robotics. *Foundations and Trends® in Robotics*, 2(1–2):1–142, 2013.

P Del Moral. Feynman-Kac formulae: genealogical and interacting particle systems with applications. *Probability and its applications*, 2004.

Pierre Del Moral, Arnaud Doucet, and Ajay Jasra. Sequential Monte Carlo samplers. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(3):411–436, 2006.

Pierre Del Moral, Arnaud Doucet, Ajay Jasra, et al. On adaptive resampling strategies for sequential Monte Carlo methods. *Bernoulli*, 18(1):252–278, 2012.

Thomas Desautels, Andreas Krause, and Joel W Burdick. Parallelizing exploration-exploitation tradeoffs in Gaussian process bandit optimization. *The JMLR*, 15(1):3873–3923, 2014.

Neil Dhir, Yura Perov, Matthew Wijers, Frank Wood, Andrew Markham, Paul Trethowan, Byron du Preez, Andrew Loveridge, and David Macdonald. Tracking african lions with nonparametric hierarchical models using probabilistic programming. In *Proceedings of the International Society of Bayesian Analysis (ISBA) 2016 World Meeting*, 2016.

Neil Dhir, Matthijs Vákár, Matthew Wijers, Andrew Markham, Frank Wood, Paul Trethowan, Byron du Preez, Andrew Loveridge, and David Macdonald. Interpreting lion behaviour with nonparametric probabilistic programs. In *UAI*, 2017.

Pedro M Domingos. Why does bagging work? a Bayesian account and its implications. In *KDD*, pages 155–158, 1997.

Joseph L Doob. Application of the theory of Martingales. *Le calcul des probabilites et ses applications*, pages 23–27, 1949.

Randal Douc and Olivier Cappé. Comparison of resampling schemes for particle filtering. In *Image and Signal Processing and Analysis, 2005. ISPA 2005. Proceedings of the 4th International Symposium on*, pages 64–69. IEEE, 2005.

Arnaud Doucet and Adam M Johansen. A tutorial on particle filtering and smoothing: Fifteen years later. *Handbook of Nonlinear Filtering*, 12:656–704, 2009.

Arnaud Doucet, Nando De Freitas, and Neil Gordon. An introduction to sequential Monte Carlo methods. In *Sequential Monte Carlo methods in practice*, pages 3–14. Springer, 2001a.

Arnaud Doucet, Nando de Freitas, and Neil Gordon. *Sequential Monte Carlo methods in practice*. Springer Science & Business Media, 2001b.

Arnaud Doucet, Mark Briers, and Stéphane Sénécal. Efficient block sampling strategies for sequential Monte Carlo methods. *Journal of Computational and Graphical Statistics*, 15(3):693–711, 2006.

Arnaud Doucet, Michael Pitt, George Deligiannidis, and Robert Kohn. Efficient implementation of Markov chain Monte Carlo when using an unbiased likelihood estimator. *Biometrika*, page asu075, 2015.

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

Richard Durbin, Sean R Eddy, Anders Krogh, and Graeme Mitchison. *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge university press, 1998.

Rick Durrett. *Probability: theory and examples*. Cambridge university press, 2010.

David Duvenaud. *Automatic model construction with Gaussian processes*. PhD thesis, University of Cambridge, 2014.

David Duvenaud, James Robert Lloyd, Roger Grosse, Joshua B Tenenbaum, and Zoubin Ghahramani. Structure discovery in nonparametric regression through compositional kernel search. *arXiv preprint arXiv:1302.4922*, 2013.

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

Pierre Etoré and Benjamin Jourdain. Adaptive optimal allocation in stratified sampling methods. *Methodology and Computing in Applied Probability*, 12(3):335–360, 2010.

Owain Evans, Andreas Stuhlmüller, John Salvatier, and Daniel Filan. Modeling Agents with Probabilistic Programs. `http://agentmodels.org`, 2017. Accessed: 2017-9-21.

Geir Evensen. Sequential data assimilation with a nonlinear quasi-geostrophic model using Monte Carlo methods to forecast error statistics. *Journal of Geophysical Research: Oceans*, 99, 1994.

Richard G. Everitt. Bayesian parameter estimation for latent Markov random fields and social networks. *Journal of Computational and Graphical Statistics*, 21(4):940–960, 2012.

Paul Feliot, Julien Bect, and Emmanuel Vazquez. A Bayesian approach to constrained single-and multi-objective optimization. *Journal of Global Optimization*, 67(1-2):97–133, 2017.

William Feller. An introduction to probability theory and its applications. vol. i. 1950.

Thomas S Ferguson. A Bayesian analysis of some nonparametric problems. *The annals of statistics*, pages

209–230, 1973.

Gersende Fort, Emmanuel Gobet, and Eric Moulines. MCMC design-based non-parametric regression for rare-event. application to nested risk computations. *Monte Carlo Methods Appl*, 2017.

David A Freedman. On the asymptotic behavior of Bayes' estimates in the discrete case. *The Annals of Mathematical Statistics*, pages 1386–1403, 1963.

D Frenkel. Waste-recycling monte carlo. *Computer Simulations in Condensed Matter Systems: From Materials to Chemical Biology Volume 1*, pages 127–137, 2006.

Charles C J Frye, Ann Galizio, Jonathan E Friedel, W Brady DeHart, and Amy L Odum. Measuring Delay Discounting in Humans Using an Adjusting Amount Task. *Journal of Visualized Experiments*, 2016.

Yarin Gal. *Uncertainty in deep learning*. PhD thesis, 2017.

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

Roman Garnett, Michael A Osborne, and Stephen J Roberts. Bayesian optimization for sensor set selection. In *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pages 209–219. ACM, 2010.

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

Andrew Gelman and Christian P Robert. "Not only defended but also applied": The perceived absurdity of Bayesian inference. *The American Statistician*, 67(1):1–5, 2013.

Andrew Gelman, John B Carlin, Hal S Stern, David B Dunson, Aki Vehtari, and Donald B Rubin. *Bayesian data analysis*, volume 2. CRC press Boca Raton, FL, 2014.

Andrew Gelman et al. Objections to Bayesian statistics. *Bayesian Analysis*, 2008.

Andrew Gelman et al. Induction and deduction in Bayesian data analysis. *Rationality, Markets and Morals*, 2(67-78):1999, 2011.

Zoubin Ghahramani. Probabilistic machine learning and artificial intelligence. *Nature*, 2015.

Zoubin Ghahramani and Carl E Rasmussen. Bayesian Monte Carlo. In *NIPS*, pages 505–512, 2003.

Walter R Gilks and Pascal Wild. Adaptive rejection sampling for Gibbs sampling. *Applied Statistics*, pages 337–348, 1992.

Walter R Gilks, Sylvia Richardson, and David Spiegelhalter. *Markov chain Monte Carlo in practice*. CRC press, 1995.

Peter W Glynn and Chang-han Rhee. Exact estimation for Markov chain equilibrium expectations. *Journal of Applied Probability*, 51(A):377–389, 2014.

Takashi Goda. Computing the variance of a conditional expectation via non-nested Monte Carlo. *Operations Research Letters*, 2016.

Javier González, Zhenwen Dai, Philipp Hennig, and Neil Lawrence. Batch Bayesian optimization via local penalization. In *AISTATS*, pages 648–657, 2016a.

Javier González, Michael Osborne, and Neil Lawrence. Glasses: Relieving the myopia of Bayesian optimisation. In *AISTATS*, pages 790–799, 2016b.

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

Noah D Goodman. The principles and practice of probabilistic programming. *ACM SIGPLAN Notices*, 48 (1):399–402, 2013.

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

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

Steven N Goodman. Toward evidence-based medical statistics. 1: The p value fallacy. *Annals of internal medicine*, 130(12):995–1004, 1999.

Andrew D Gordon, Thomas A Henzinger, Aditya V Nori, and Sriram K Rajamani. Probabilistic programming. In *Proceedings of the on Future of Software Engineering*. ACM, 2014.

Neil J Gordon, David J Salmond, and Adrian FM Smith. Novel approach to nonlinear/non-Gaussian Bayesian state estimation. *IEE Proceedings F (Radar and Signal Processing)*, 140(2):107–113, 1993.

Michael B Gordy and Sandeep Juneja. Nested simulation in portfolio risk measurement. *Management Science*, 2010.

Robert B Gramacy and Herbert KH Lee. Optimization under unknown constraints. *arXiv preprint arXiv:1004.4027*, 2010.

Shixiang Gu, Zoubin Ghahramani, and Richard E Turner. Neural adaptive sequential Monte Carlo. In *NIPS*, pages 2629–2637, 2015.

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

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., 2001.

W Keith Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

Stefan Heinrich. Multilevel Monte Carlo methods. *LSSC*, 1:58–67, 2001.

Philipp Hennig and Christian J Schuler. Entropy search for information-efficient global optimization. *JMLR*, 13(Jun):1809–1837, 2012.

Philipp Hennig, Michael A Osborne, and Mark Girolami. Probabilistic numerics and uncertainty in computations. In *Proc. R. Soc. A*, volume 471, page 20150142. The Royal Society, 2015.

James Hensman, Nicolo Fusi, and Neil D Lawrence. Gaussian processes for big data. In *UAI*, page 282. Citeseer, 2013.

James Hensman, Alexander G Matthews, Maurizio Filippone, and Zoubin Ghahramani. MCMC for variationally sparse Gaussian processes. In *NIPS*, pages 1648–1656, 2015.

Daniel Hernández-Lobato, Jose Hernandez-Lobato, Amar Shah, and Ryan Adams. Predictive entropy search for multi-objective Bayesian optimization. In *ICML*, pages 1492–1501, 2016a.

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

José Miguel Hernández-Lobato, Michael A Gelbart, Ryan P Adams, Matthew W Hoffman, and Zoubin Ghahramani. A general framework for constrained bayesian optimization using information-based search. 2016b.

Timothy Classen Hesterberg. *Advances in importance sampling*. PhD thesis, Stanford University, 1988.

Chris Heunen, Ohad Kammar, Sam Staton, and Hongseok Yang. A convenient category for higher-order probability theory. *arXiv preprint arXiv:1701.02547*, 2017.

Rich Hickey. The clojure programming language. In *Proceedings of the 2008 symposium on Dynamic languages*, page 1. ACM, 2008.

David M Higdon. Auxiliary variable methods for Markov chain Monte Carlo with applications. *Journal of the American Statistical Association*, 93(442):585–595, 1998.

Matthew Hoffman and David Blei. Stochastic structured variational inference. In *AISTATS*, 2015.

Matthew D Hoffman and Andrew Gelman. The No-U-turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo. *JMLR*, 15(1):1593–1623, 2014.

Thomas Hofmann, Bernhard Schölkopf, and Alexander J Smola. Kernel methods in machine learning. *The annals of statistics*, pages 1171–1220, 2008.

Roman Holenstein. *Particle Markov chain Monte Carlo*. PhD thesis, University Of British Columbia, 2009.

L Jeff Hong and Sandeep Juneja. Estimating the mean of a non-linear function of conditional expectation. In *Winter Simulation Conference*, 2009.

Jonathan H. Huggins and Daniel M. Roy. Convergence of sequential Monte Carlo-based sampling methods. *ArXiv e-prints, arXiv:1503.00966v1*, March 2015.

Chung-Kil Hur, Aditya V Nori, Sriram K Rajamani, and Selva Samuel. A provably correct sampler for probabilistic programs. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 45. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

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

John PA Ioannidis. Why most published research findings are false. *PLoS medicine*, 2(8):e124, 2005.

Peter Jäckel. *Monte Carlo methods in finance*. J. Wiley, 2002.

Pierre E Jacob, Alexandre H Thiery, et al. On nonnegative unbiased estimators. *The Annals of Statistics*, 43(2):769–784, 2015.

Pierre E Jacob, Fredrik Lindsten, and Thomas B Schön. Smoothing with couplings of conditional particle filters. *arXiv preprint arXiv:1701.02002*, 2017.

David Janz, Brooks Paige, Tom Rainforth, Jan-Willem van de Meent, and Frank Wood. Probabilistic structure discovery in time series data. *NIPS Workshop on Artificial Intelligence for Data Science*, 2016.

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

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

Galin L Jones, Gareth O Roberts, and Jeffrey S Rosenthal. Convergence of conditional Metropolis-Hastings samplers. *Advances in Applied Probability*, 46(2):422–445, 2014.

Galin L Jones et al. On the Markov chain central limit theorem. *Probability surveys*, 2004.

Michael I Jordan. Are you a Bayesian or a frequentist? 2009. URL `http://videolectures.net/mlss09uk_jordan_bfway/`.

Dan Jurafsky and James H Martin. *Speech and language processing*, volume 3. Pearson London, 2014.

Herman Kahn and Andy W Marshall. Methods of reducing sample size in Monte Carlo computations. *Journal of the Operations Research Society of America*, 1(5):263–278, 1953.

Rudolph Emil Kalman et al. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

Kirthevasan Kandasamy, Akshay Krishnamurthy, Jeff Schneider, and Barnabas Poczos. Asynchronous parallel Bayesian optimisation via Thompson sampling. *arXiv preprint arXiv:1705.09236*, 2017.

Tarun Kathuria, Amit Deshpande, and Pushmeet Kohli. Batched Gaussian process bandit optimization via determinantal point processes. In *NIPS*, pages 4206–4214, 2016.

Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. In *ICLR*, 2014.

Claude Kipnis and SR Srinivasa Varadhan. Central limit theorem for additive functionals of reversible Markov processes and applications to simple exclusions. *Communications in Mathematical Physics*, 104(1):1–19, 1986.

Kris N Kirby. One-year temporal stability of delay-discount rates. *Psychonomic Bulletin & Review*, 16(3): 457–462, June 2009.

Oleg Kiselyov. Problems of the lightweight implementation of probabilistic programming. In *Proceedings of Workshop on Probabilistic Programming Semantics*, 2016.

Genshiro Kitagawa. Monte Carlo filter and smoother for non-Gaussian nonlinear state space models. *Journal of computational and graphical statistics*, 5(1):1–25, 1996.

BJK Kleijn, AW Van der Vaart, et al. The Bernstein-von-Mises theorem under misspecification. *Electronic Journal of Statistics*, 6:354–381, 2012.

L L Kontsevich and C W Tyler. Bayesian adaptive estimation of psychometric slope and threshold. *Vision research*, 39(16):2729–2737, August 1999.

Alp Kucukelbir, Rajesh Ranganath, Andrew Gelman, and David Blei. Automatic variational inference in

Stan. In *NIPS*, pages 568–576, 2015.

Tejas Dattatraya Kulkarni, Pushmeet Kohli, Joshua B Tenenbaum, and Vikash Kumar Mansinghka. Picture: a probabilistic programming language for scene perception. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4390–4399, 2015.

Harold J Kushner. A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. *Journal of Basic Engineering*, 86(1):97–106, 1964.

Malte Kuss and Carl Edward Rasmussen. Assessing approximate inference for binary Gaussian process classification. *JMLR*, 6(Oct):1679–1704, 2005.

Tze Leung Lai and Herbert Robbins. Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, 6(1):4–22, 1985.

Balaji Lakshminarayanan, Daniel Roy, and Yee Whye Teh. Top-down particle filtering for Bayesian decision trees. In *ICML*, pages 280–288, 2013.

David P Landau and Kurt Binder. *A guide to Monte Carlo simulations in statistical physics*. Cambridge university press, 2014.

Steffen L Lauritzen and David J Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 157–224, 1988.

Neil D Lawrence. Gaussian process latent variable models for visualisation of high dimensional data. In *NIPS*, pages 329–336, 2004.

Tuan Anh Le. Inference for higher order probabilistic programs. *Masters thesis, University of Oxford*, 2015.

Tuan Anh Le, Atılım Güneş Baydin, and Frank Wood. Nested compiled inference for hierarchical reinforcement learning. In *NIPS Workshop on Bayesian Deep Learning*, 2016.

Tuan Anh Le, Atılım Güneş Baydin, and Frank Wood. Inference compilation and universal probabilistic programming. In *20th AISTATS*, 2017a.

Tuan Anh Le, Atilim Gunes Baydin, Robert Zinkov, and Frank Wood. Using synthetic data to train neural networks is model-based reasoning. *arXiv preprint arXiv:1703.00868*, 2017b.

Tuan Anh Le, Maximilian Igl, Tom Jin, Tom Rainforth, and Frank Wood. Auto-encoding sequential Monte Carlo. *arXiv preprint arXiv:1705.10306*, 2017c.

Stéphanie Lefèvre, Dizan Vasquez, and Christian Laugier. A survey on motion prediction and risk assessment for intelligent vehicles. *Robomech Journal*, 1(1):1, 2014.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *arXiv preprint arXiv:1603.06560*, 2016.

Faming Liang. A double Metropolis–Hastings sampler for spatial models with intractable normalizing constants. *Journal of Statistical Computation and Simulation*, 80(9):1007–1022, 2010.

Fredrik Lindsten and Thomas B Schön. Backward simulation methods for Monte Carlo statistical inference. *Foundations and Trends in Machine Learning*, 6(1):1–143, 2013.

Fredrik Lindsten, Michael I. Jordan, and Thomas B. Schön. Particle Gibbs with ancestor sampling. *JMLR*, 15:2145–2184, june 2014.

Fredrik Lindsten, Randal Douc, and Eric Moulines. Uniform ergodicity of the particle Gibbs sampler. *Scandinavian Journal of Statistics*, 42(3):775–797, 2015.

Jun S Liu and Rong Chen. Blind deconvolution via sequential imputations. *Journal of the american statistical association*, 90(430):567–576, 1995.

Jun S Liu and Rong Chen. Sequential Monte Carlo methods for dynamic systems. *Journal of the American statistical association*, 93(443):1032–1044, 1998.

Daniel James Lizotte. *Practical Bayesian optimization*. University of Alberta, 2008.

James Robert Lloyd, David K Duvenaud, Roger B Grosse, Joshua B Tenenbaum, and Zoubin Ghahramani. Automatic construction and natural-language description of nonparametric regression models. In *AAAI*, pages 1242–1250, 2014.

Francis A Longstaff and Eduardo S Schwartz. Valuing American options by simulation: a simple least-squares approach. *Review of Financial studies*, 2001.

David J Lunn, Andrew Thomas, Nicky Best, and David Spiegelhalter. Winbugs-a Bayesian modelling framework: concepts, structure, and extensibility. *Statistics and computing*, 10(4):325–337, 2000.

Anne-Marie Lyne, Mark Girolami, Yves Atchade, Heiko Strathmann, Daniel Simpson, et al. On Russian roulette estimates for Bayesian inference with doubly-intractable likelihoods. *Statistical science*, 30(4): 443–467, 2015.

Pierre L'Ecuyer and Christiane Lemieux. Recent advances in randomized quasi-Monte Carlo methods. *Modeling uncertainty*, pages 419–474, 2005.

Lars Maaløe, Casper Kaae Sønderby, Søren Kaae Sønderby, and Ole Winther. Auxiliary deep generative models. *arXiv preprint arXiv:1602.05473*, 2016.

Chris J Maddison, Dieterich Lawson, George Tucker, Nicolas Heess, Mohammad Norouzi, Andriy Mnih, Arnaud Doucet, and Yee Whye Teh. Filtering variational objectives. *arXiv preprint arXiv:1705.09279*, 2017.

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

Vikash K Mansinghka, Tejas D Kulkarni, Yura N Perov, and Josh Tenenbaum. Approximate Bayesian image interpretation using generative probabilistic graphics programs. In *NIPS*, pages 1520–1528, 2013.

Theofrastos Mantadelis and Gerda Janssens. Nesting probabilistic inference. *arXiv preprint arXiv:1112.3785*, 2011.

George Marsaglia, Wai Wan Tsang, et al. The ziggurat method for generating random variables. *Journal of statistical software*, 5(8):1–7, 2000.

David McAllester. A pac-Bayesian tutorial with a dropout bound. *arXiv preprint arXiv:1307.2118*, 2013.

A McCallum, K Schultz, and S Singh. Factorie: Probabilistic programming via imperatively defined factor graphs. In *NIPS*, volume 22, 2009.

Nicholas Metropolis and Stanislaw Ulam. The Monte Carlo method. *Journal of the American statistical association*, 44(247):335–341, 1949.

Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21 (6):1087–1092, 1953.

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

Thomas P Minka. Bayesian model averaging is not model combination. *Available electronically at http://www. stat. cmu. edu/minka/papers/bma. html*, 2000.

J Mockus. On Bayesian methods for seeking the extremum. In *Optimization Techniques IFIP Technical Conference*, pages 400–404. Springer, 1975.

Jesper Møller, Anthony N Pettitt, R Reeves, and Kasper K Berthelsen. An efficient Markov chain Monte Carlo method for distributions with intractable normalising constants. *Biometrika*, 93(2):451–458, 2006.

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

Iain Murray and Ryan P Adams. Slice sampling covariance hyperparameters of latent Gaussian models. In *NIPS*, 2010.

Iain Murray and Zoubin Ghahramani. Bayesian learning in undirected graphical models: approximate MCMC algorithms. In *UAI*, pages 392–399. AUAI Press, 2004.

Iain Murray, Zoubin Ghahramani, and David JC MacKay. MCMC for doubly-intractable distributions. In *UAI*, pages 359–366. AUAI Press, 2006.

Lawrence M Murray. Bayesian state-space modelling on high-performance hardware using libbi. *arXiv preprint arXiv:1306.3277*, 2013.

Jay I Myung, Daniel R Cavagnaro, and Mark A Pitt. A tutorial on adaptive design optimization. *Journal of mathematical psychology*, 57(3):53–67, 2013.

Christian A Naesseth, Fredrik Lindsten, and Thomas B Schön. Sequential Monte Carlo for graphical models. In *NIPS 27*, pages 1862–1870. Curran Associates, Inc., 2014.

Christian A. Naesseth, Fredrik Lindsten, and Thomas B Schön. Nested sequential Monte Carlo methods. In *ICML*, volume 37, 2015.

Christian A Naesseth, Scott W Linderman, Rajesh Ranganath, and David M Blei. Variational sequential Monte Carlo. *arXiv preprint arXiv:1705.11140*, 2017.

Praveen Narayanan, Jacques Carette, Wren Romano, Chung-chieh Shan, and Robert Zinkov. Probabilistic inference by program transformation in hakaru (system description). In *International Symposium on Functional and Logic Programming*. Springer, 2016.

Radford M Neal. Probabilistic inference using Markov chain Monte Carlo methods. 1993.

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

Radford M Neal. MCMC using Hamiltonian dynamics. *Handbook of MCMC*, 2, 2011.

James Neufeld, András György, Dale Schuurmans, and Csaba Szepesvári. Adaptive Monte Carlo via bandit allocation. *arXiv preprint arXiv:1405.3318*, 2014.

Andrew Y Ng and Michael I Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *NIPS*, pages 841–848, 2002.

Anthony O'Hagan. Monte Carlo is fundamentally unsound. *The Statistician*, pages 247–249, 1987.

Anthony O'Hagan. Bayes–hermite quadrature. *Journal of statistical planning and inference*, 1991.

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

Michael Osborne, Roman Garnett, Zoubin Ghahramani, David K Duvenaud, Stephen J Roberts, and Carl E Rasmussen. Active learning of model evidence using Bayesian quadrature. In *NIPS*, 2012.

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

Long Ouyang, Michael Henry Tessler, Daniel Ly, and Noah Goodman. Practical optimal experiment design with probabilistic programs. *arXiv preprint arXiv:1608.05046*, 2016.

Art B. Owen. *Monte Carlo theory, methods and examples*. 2013.

Brooks Paige. *Automatic inference for higher-order probabilistic programs*. PhD thesis, PhD thesis, University of Oxford, 2016.

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

Brooks Paige and Frank Wood. Inference networks for sequential Monte Carlo in graphical models. In *ICML*, pages 3040–3049, 2016.

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

Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1982.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. *NIPS Workshop on AutoDiff*, 2017.

Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. 2014.

Giacomo Perantoni and David JN Limebeer. Optimal control for a formula one car with variable parameters. *Vehicle System Dynamics*, 52(5):653–678, 2014.

Kaare Brandt Petersen, Michael Syskind Pedersen, et al. The matrix cookbook. *Technical University of Denmark*, 7:15, 2008.

Michael K Pitt, Ralph dos Santos Silva, Paolo Giordani, and Robert Kohn. On some properties of Markov chain Monte Carlo simulation methods based on the particle filter. *Journal of Econometrics*, 171(2): 134–151, 2012.

Martyn Plummer et al. Jags: A program for analysis of Bayesian graphical models using Gibbs sampling. In *Proceedings of the 3rd international workshop on distributed statistical computing*, 2003.

N Prins. The psi-marginal adaptive method: How to give nuisance parameters the attention they deserve (no more, no less). *Journal of Vision*, 13(7):3–3, June 2013.

Lawrence Rabiner and B Juang. An introduction to hidden Markov models. *IEEE ASSP magazine*, 3(1): 4–16, 1986.

Tom Rainforth. Nesting probabilistic programs. *arXiv preprint arXiv:1803.06328*, 2018.

Tom Rainforth and Frank Wood. Canonical correlation forests. *arXiv preprint arXiv:1507.05444*, 2015.

Tom Rainforth, Jan-Willem van de Meent, Michael A Osborne, and Frank Wood. Bayesian optimization for probabilistic programs. *1st NIPS Workshop on Black Box Learning and Inference*, 2015.

Tom Rainforth, Robert Cornish, Hongseok Yang, and Frank Wood. On the pitfalls of nested Monte Carlo. *NIPS Workshop on Advances in Approximate Bayesian Inference*, 2016a.

Tom Rainforth, Tuan Anh Le, Jan-Willem van de Meent, Michael A Osborne, and Frank Wood. Bayesian Optimization for Probabilistic Programs. In *NIPS*, pages 280–288, 2016b.

Tom Rainforth, Christian A Naesseth, Fredrik Lindsten, Brooks Paige, Jan-Willem van de Meent, Arnaud Doucet, and Frank Wood. Interacting particle Markov chain Monte Carlo. *ICML*, 48, 2016c.

Tom Rainforth, Robert Cornish, Hongseok Yang, Andrew Warrington, and Frank Wood. On the opportunities and pitfalls of nesting Monte Carlo estimators. *arXiv preprint arXiv:1709.06181*, 2017a.

Tom Rainforth, Tuan Anh Le, Jan Willem van de Meent, Michael A. Osborne, and Frank Wood. Bayesian optimization for probabilistic programs. *arXiv e-prints, arXiv:1707.04314*, 2017b.

Rajesh Ranganath, Sean Gerrish, and David M Blei. Black Box Variational Inference. In *AISTATS*, 2014.

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

Herbert E Rauch, CT Striebel, and F Tung. Maximum likelihood estimates of linear dynamic systems. *AIAA journal*, 3(8):1445–1450, 1965.

Erzsébet Ravasz and Albert-László Barabási. Hierarchical organization in complex networks. *Physical Review E*, 67(2):026112, 2003.

Daniel Ritchie, Paul Horsfall, and Noah D Goodman. Deep amortized inference for probabilistic programs. *arXiv preprint arXiv:1610.05735*, 2016a.

Daniel Ritchie, Andreas Stuhlmüller, and Noah Goodman. C3: Lightweight incrementalized MCMC for probabilistic programs using continuations and callsite caching. In *AISTATS*, pages 28–37, 2016b.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

Christian Robert. *The Bayesian choice: from decision-theoretic foundations to computational implementation*. Springer Science & Business Media, 2007.

Christian P Robert. *Monte Carlo methods*. Wiley Online Library, 2004.

Gareth O Roberts and Adrian FM Smith. Simple conditions for the convergence of the Gibbs sampler and Metropolis-Hastings algorithms. *Stochastic processes and their applications*, 49(2):207–216, 1994.

Reuven Y Rubinstein and Dirk P Kroese. *Simulation and the Monte Carlo method*, volume 10. 2016.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

John Salvatier, Thomas V Wiecki, and Christopher Fonnesbeck. Probabilistic programming in python using pymc3. *PeerJ Computer Science*, 2:e55, 2016.

Thomas C Schelling. *The strategy of conflict*. Harvard university press, 1980.

Michael Schober. *Practical probabilistic ordinary differential equation solvers—theory and applications [In Preparation]*. PhD thesis, Eberhard Karls University of Tübingen, 2017.

Bernhard Schölkopf and Alexander J Smola. Learning with kernels: support vector machines, regularization, optimization, and beyond, 2002.

Paola Sebastiani and Henry P Wynn. Maximum entropy sampling and optimal Bayesian experimental

design. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 62(1), 2000.

Amar Shah and Zoubin Ghahramani. Pareto frontier learning with expensive correlated objectives. In *ICML*, pages 1919–1927, 2016.

Bobak Shahriari, Ziyu Wang, Matthew W Hoffman, Alexandre Bouchard-Côté, and Nando de Freitas. An entropy search portfolio for Bayesian optimization. *arXiv preprint arXiv:1406.4625*, 2014.

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

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

N Siddharth, Brooks Paige, Jan-Willem van de Meent, Alban Desmaison, Frank Wood, Noah D Goodman, Pushmeet Kohli, Philip HS Torr, et al. Learning disentangled representations with semi-supervised deep generative models. *arXiv preprint arXiv:1706.00400*, 2017.

John Skilling. Nested sampling. In *AIP Conference Proceedings*, volume 735, pages 395–405. AIP, 2004.

Edward Snelson and Zoubin Ghahramani. Sparse Gaussian processes using pseudo-inputs. In *NIPS*, 2006.

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

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

David Spiegelhalter, Andrew Thomas, Nicky Best, and Wally Gilks. Bugs 0.5: Bayesian inference using Gibbs sampling manual (version ii). *MRC Biostatistics Unit, Cambridge*, 1996.

Niranjan Srinivas, Andreas Krause, Sham M Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. *arXiv preprint arXiv:0912.3995*, 2009.

Sam Staton, Hongseok Yang, Frank Wood, Chris Heunen, and Ohad Kammar. Semantics for probabilistic programming: higher-order functions, continuous distributions, and soft constraints. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 525–534. ACM, 2016.

Jacob Steinhardt. Beyond Bayesians and frequentists. 2012.

Andreas Stuhlmüller and Noah D Goodman. A dynamic programming algorithm for inference in recursive probabilistic programs. In *Second Statistical Relational AI workshop at UAI 2012 (StaRAI-12)*, 2012.

Andreas Stuhlmüller and Noah D Goodman. Reasoning about reasoning by nested conditioning: Modeling theory of mind with probabilistic programs. *Cognitive Systems Research*, 28:80–99, 2014.

Kevin Swersky, Jasper Snoek, and Ryan P Adams. Multi-task Bayesian optimization. In *NIPS*, 2013.

Yee Whye Teh. Dirichlet process. In *Encyclopedia of machine learning*, pages 280–287. Springer, 2011.

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

Luke Tierney. Markov chains for exploring posterior distributions. *The Annals of Statistics*, 1994.

Adrien Todeschini, François Caron, Marc Fuentes, Pierrick Legrand, and Pierre Del Moral. Biips: software for Bayesian inference with interacting particle systems. *arXiv preprint arXiv:1412.3779*, 2014.

David Tolpin and Frank Wood. Maximum a posteriori estimation by search in probabilistic programs. In *Proceedings of the Eighth Annual Symposium on Combinatorial Search*, pages 201–205, 2015.

David Tolpin, Jan Willem van de Meent, Brooks Paige, and Frank Wood. Output-sensitive adaptive Metropolis-Hastings for probabilistic programs. *arXiv preprint arXiv:1501.05677*, 2015a.

David Tolpin, Jan-Willem van de Meent, and Frank Wood. Probabilistic programming in Anglican. Springer International Publishing, 2015b.

David Tolpin, Jan-Willem van de Meent, Hongseok Yang, and Frank Wood. Design and implementation of probabilistic programming language Anglican. In *Proceedings of the 28th Symposium on the Implementation and Application of Functional Programming Languages*, page 6. ACM, 2016.

Dustin Tran, Alp Kucukelbir, Adji B. Dieng, Maja Rudolph, Dawen Liang, and David M. Blei. Edward: A

library for probabilistic modeling, inference, and criticism. *arXiv preprint arXiv:1610.09787*, 2016.

Nilesh Tripuraneni, Shixiang Gu, Hong Ge, and Zoubin Ghahramani. Particle Gibbs for infinite hidden Markov Models. In *NIPS*, pages 2386–2394. Curran Associates, Inc., 2015.

Isabel Valera, Fran Francisco, Lennart Svensson, and Fernando Perez-Cruz. Infinite factorial dynamical model. In *NIPS*, pages 1657–1665. Curran Associates, Inc., 2015.

Jan-Willem van de Meent, Hongseok Yang, Vikash Mansinghka, and Frank Wood. Particle Gibbs with ancestor sampling for probabilistic programs. In *AISTATS*, pages 986–994, 2015.

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

David A Van Dyk and Taeyoung Park. Partially collapsed Gibbs samplers: Theory and methods. *Journal of the American Statistical Association*, 103(482):790–796, 2008.

Joep Vanlier, Christian A Tiemann, Peter AJ Hilbers, and Natal AW van Riel. A Bayesian approach to targeted experiment design. *Bioinformatics*, 28(8):1136–1142, 2012.

Vladimir Naumovich Vapnik. *Statistical learning theory*, volume 1. Wiley New York, 1998.

Henk Kaarle Versteeg and Weeratunge Malalasekera. *An introduction to computational fluid dynamics: the finite volume method*. Pearson Education, 2007.

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

Benjamin T Vincent. Hierarchical Bayesian estimation and hypothesis testing for delay discounting tasks. *Behavior research methods*, 48(4):1608–1620, 2016.

Benjamin T Vincent and Tom Rainforth. The DARC Toolbox: automated, flexible, and efficient delayed and risky choice experiments using bayesian adaptive design. *PsyArXiv*, 2017.

Luis Von Ahn, Benjamin Maurer, Colin McMillen, David Abraham, and Manuel Blum. Recaptcha: Human-based character recognition via web security measures. *Science*, 321(5895):1465–1468, 2008.

Zi Wang, Bolei Zhou, and Stefanie Jegelka. Optimization as estimation with Gaussian processes in bandit settings. In *International Conf. on Artificial and Statistics (AISTATS)*, 2016.

Nick Whiteley, Christophe Andrieu, and Arnaud Doucet. Efficient Bayesian inference for switching state-space models using discrete particle Markov chain Monte Carlo methods. *ArXiv e-prints, arXiv:1011.2437*, 2010.

Nick Whiteley, Anthony Lee, and Kari Heine. On the role of interaction in sequential Monte Carlo algorithms. *Bernoulli*, 22(1):494–529, 02 2016.

Darrell Whitley. A genetic algorithm tutorial. *Statistics and computing*, 4(2):65–85, 1994.

David Williams. *Probability with Martingales*. Cambridge university press, 1991.

Andrew Wilson, Elad Gilboa, John P Cunningham, and Arye Nehorai. Fast kernel learning for multidimensional pattern extrapolation. In *NIPS*, pages 3626–3634, 2014.

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

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

Charles Xie. Interactive heat transfer simulations for everyone. *The Physics Teacher*, 2012.

Lingfeng Yang, Patrick Hanrahan, and Noah Goodman. Generating efficient MCMC kernels from probabilistic programs. In *AISTATS*, pages 1068–1076, 2014.

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