Large-Scale Bayesian Computation Using Stochastic Gradient Markov Chain Monte Carlo

Jack Baker, B.Sc.(Hons.), M.Res

Submitted for the degree of Doctor of Philosophy at Lancaster University.

December 2018

STOR-i
excellence with impact

Abstract

Markov chain Monte Carlo (MCMC), one of the most popular methods for inference on Bayesian models, scales poorly with dataset size. This is because it requires one or more calculations over the full dataset at each iteration. Stochastic gradient Markov chain Monte Carlo (SGMCMC) has become a popular MCMC method that aims to be more scalable at large datasets. It only requires a subset of the full data at each iteration. This thesis builds upon the SGMCMC literature by providing contributions that improve the efficiency of SGMCMC; providing software that improves its ease-of-use; and removes large biases in the method for an important class of model.

While SGMCMC has improved per-iteration computational cost over traditional MCMC, there have been empirical results suggesting that its overall computational cost (i.e. the cost for the algorithm to reach an arbitrary level of accuracy) is still $O(N)$, where N is the dataset size. In light of this, we show how control variates can be used to develop an SGMCMC algorithm of $O(1)$, subject to two one-off preprocessing steps which each require a single pass through the dataset.

While SGMCMC has gained significant popularity in the machine learning community, uptake among the statistics community has been slower. We suggest this may

be due to lack of software, so as part of the contributions in this thesis we provide an R software package that automates much of the procedures required to build SGMCMC algorithms. Finally, we show that current algorithms for sampling from the simplex space using SGMCMC have inherent biases, especially when some of the parameter components are close to zero. To get around this, we develop an algorithm that is provably asymptotically unbiased. We empirically demonstrate its performance on a latent Dirichlet allocation model and a Dirichlet process model.

Acknowledgements

First, I'd like to thank the staff, students and management at the STOR-i Centre for Doctoral Training. This is such an enjoyable and stimulating atmosphere to do research in, and I hope the centre stays for a long while to come. I'd like to mention the directors of STOR-i especially: Jon Tawn, Kevin Glazebrook and Idris Eckley; whose tireless work has helped build such a great atmosphere at STOR-i. Thanks for giving me the opportunity to be part of this centre. I'd also like to thank the admin staff: Kim Wilson, Jennifer Bull and Wendy Shimmin; who make the department run so smoothly, and for putting up with me. I am very grateful for the financial support provided by EPSRC.

This work could not have happened without my supervisors: Paul Fearnhead, Christopher Nemeth and Emily Fox; thank you for all the time and effort you have put into this PhD project. They have taught me so much, both technical and not.

STOR-i obviously would not be the same without its students, and I'm really grateful to all of them, both past and present. I'll remember the laughs, discussions and advice for a long time to come. There's no doubt I'll stay in touch with many of you. The CSML group members were also invaluable for their wealth of knowledge,

discussions and idea sharing.

This work would not have happened without Sorcha, who has offered me so much support and guidance during the PhD; as well as some much needed hilarity. I am also very lucky to have such supportive and entertaining friends, who have kept me sane throughout. Finally I'd like to thank my family. Without their support, guidance and laughs I wouldn't have managed to finish school, let alone undertake a PhD.

Dedicated to Scampi my wonderful cat.

Declaration

I declare that the work in this thesis has been done by myself and has not been submitted elsewhere for the award of any other degree.

Jack Baker

Contents

List of Figures

List of Tables

List of Abbreviations

MCMC	Markov Chain Monte Carlo
SGMCMC	Stochastic Gradient Markov Chain Monte Carlo
a.s.	Almost Surely
a.e.	Almost Everywhere
MH	Metropolis–Hastings
LD	Langevin Diffusion
SDE	Stochastic Differential Equation
TV	Total Variation
ULA	Unadjusted Langevin Algorithm
MALA	Metropolis–Adjusted Langevin Algorithm
MSE	Mean Square Error
RMSE	Root Mean Square Error
HMC	Hamiltonian Monte Carlo
SGLD	Stochastic Gradient Langevin Dynamics
SGD	Stochastic Gradient Descent
SGHMC	Stochastic Gradient Hamiltonian Monte Carlo

PDP	Piecewise Deterministic Process
CIR	Cox-Ingersoll-Ross Process
SCIR	Stochastic Cox-Ingersoll-Ross Process

Chapter 1

Introduction

Markov chain Monte Carlo (MCMC), one of the most popular methods for inference in Bayesian models, is known to scale poorly with dataset size. This has become a problem due to the growing complexity of practical models in both statistics and machine learning. This thesis provides contributions for stochastic gradient Markov chain Monte Carlo, a popular class of MCMC which aims to mitigate this problem.

In this chapter we set up the problem by providing a brief introduction to Bayesian inference and outlining the inherent scalability problems. This is elaborated on in Chapter 2, which provides a literature review of Monte Carlo methods and scalability. We then outline the contributions of this thesis, as well as the structure of the chapters.

1.1 Bayesian Inference

In most statistical and machine learning problems, interest is in an unknown parameter θ. For simplicity, for now we suppose that θ takes values in $\mathbb{R}^d$; but this is relaxed

in Chapter 2. Suppose relevant data is collected $\mathbf{x} = \{x_i\}_{i=1}^N$, with $x_i \in \mathbb{R}^d$. Then Bayesian inference assumes that θ is a random variable, and aims to calculate the distribution of θ given this new information $\mathbf{x}$, i.e. the distribution of $\theta \,|\, \mathbf{x}$. We refer to this distribution as π. Treating θ as a random variable rather than a fixed quantity can alleviate overfitting, which is important for the complex models currently in popular use.

Suppose the data $\mathbf{x}$ depend on a random parameter θ through the density $p_i(\theta) := p(x_i|\theta)$, here we assume that θ takes values in $\mathbb{R}^d$. We assign θ a prior density $p_0(\theta)$. Then, the posterior density $p(\theta) := p(\theta|\mathbf{x})$ (i.e. the density of π) is given by

$$p(\theta) = \frac{\prod_{i=0}^N p_i(\theta)}{Z}, \qquad Z = \int_{\mathbb{R}^d} \prod_{i=0}^N p_i(\theta) d\theta, \tag{1.1.1}$$

where Z is referred to as the normalising constant.

If Z can be calculated, then $p(\theta)$ can be calculated analytically, giving a closed form expression detailing $\theta \,|\, \mathbf{x}$ (though further integration would be required to obtain the distribution function itself). However, a fundamental problem in Bayesian inference is that the integration to find Z is rarely tractable. This means typically we only know the posterior up to the unnormalised density $h(\theta) := \prod_{i=0}^N p_i(\theta)$. MCMC gets around this issue by constructing an algorithm that will converge to sampling from π; while only needing to evaluate the unnormalised density h (for exact details see Section 2.2). Most quantities of interest can be written in the form $\mathbb{E}_\pi[\psi(\theta)]$. This quantity can then be estimated using the MCMC sample θ_m, $m = 1, \ldots, M$ by using the Monte Carlo estimate

$$\mathbb{E}_\pi[\psi(\theta)] \approx \frac{1}{M} \sum_{m=1}^M \psi(\theta_m).$$

In many modern statistics and machine learning problems, the dataset sizes N are very large. However, MCMC requires the calculation of h at each iteration. Since h is a product of $N + 1$ terms, this is an $O(N)$ calculation and can cause MCMC to be prohibitively slow for large datasets. This has sparked interest in improving the computational efficiency of MCMC. One of the most popular methods for doing so is stochastic gradient MCMC (SGMCMC), which uses a subset of the data at each iteration of size n. This enables an algorithm to be implemented with $O(n)$ calculations at each iteration. The main cost for the improved efficiency is that SGMCMC samples are no longer guaranteed to converge to π.

1.2 Contributions and Thesis Outline

This thesis has focussed on developing three aspects of SGMCMC: efficiency, ease-of-use, and performance on an important class of problems. Contributions include: providing a detailed review of SGMCMC, including details of underlying theory and a comparison to an alternative popular class of scalable MCMC; establishing a framework for SGMCMC which provably improves its overall computational cost; developing a software package for SGMCMC which enhances its ease of implementation; and improving the performance of SGMCMC when the method is used to sample from simplex spaces, an important class of problem.

The material for this thesis is presented in four chapters. Chapter 2 contains a review of scalable Monte Carlo methods, and Chapters 3, 4 and 5 contain new research that has been accepted for publication. We now give a brief outline of each chapter.

Chapter 2: Monte Carlo Methods and Scalability

This Chapter provides a review of SGMCMC. The chapter first outlines standard MCMC methods. Then useful background material for SGMCMC is detailed, including continuous-time Markov processes and Itô processes. Important methodology in the SGMCMC literature is outlined based on the background material. Comparisons between SGMCMC and divide-and-conquer MCMC, an alternative popular class of scalable MCMC methods, are provided.

Chapter 3: Control Variates for Stochastic Gradient MCMC

This chapter is a journal contribution with co-authors Paul Fearnhead, Emily B. Fox and Christopher Nemeth. The manuscript has been accepted by the journal "Statistics and Computing." The abstract of the publication is given below.

It is well known that Markov chain Monte Carlo (MCMC) methods scale poorly with dataset size. A popular class of methods for solving this issue is stochastic gradient MCMC (SGMCMC). These methods use a noisy estimate of the gradient of the log-posterior, which reduces the per iteration computational cost of the algorithm. Despite this, there are a number of results suggesting that stochastic gradient Langevin dynamics (SGLD), probably the most popular of these methods, still has computational cost proportional to the dataset size. We suggest an alternative log-posterior gradient estimate for stochastic gradient MCMC which uses control variates to reduce the variance. We analyse SGLD using this gradient estimate, and show that, under log-concavity assumptions on the target distribution, the computational cost required for a given level of accuracy is independent of the dataset size. Next we show that

a different control variate technique, known as zero variance control variates, can be applied to SGMCMC algorithms for free. This post-processing step improves the inference of the algorithm by reducing the variance of the MCMC output. Zero variance control variates rely on the gradient of the log-posterior; we explore how the variance reduction is affected by replacing this with the noisy gradient estimate calculated by SGMCMC.

Chapter 4: sgmcmc: An R Package for Stochastic Gradient Markov Chain Monte Carlo

This chapter is a journal contribution with co-authors Paul Fearnhead, Emily B. Fox and Christopher Nemeth. The manuscript has been accepted by the journal "Journal of Statistical Software." The abstract of the publication is given below.

This paper introduces the R package sgmcmc; which can be used for Bayesian inference on problems with large datasets using stochastic gradient Markov chain Monte Carlo (SGMCMC). Traditional Markov chain Monte Carlo (MCMC) methods, such as Metropolis–Hastings, are known to run prohibitively slowly as the dataset size increases. SGMCMC solves this issue by only using a subset of data at each iteration. SGMCMC requires calculating gradients of the log likelihood and log priors, which can be time consuming and error prone to perform by hand. The sgmcmc package calculates these gradients itself using automatic differentiation, making the implementation of these methods much easier. To do this, the package uses the software library TensorFlow, which has a variety of statistical distributions and mathematical operations as standard, meaning a wide class of models can be built using this framework.

SGMCMC has become widely adopted in the machine learning literature, but less so in the statistics community. We believe this may be partly due to lack of software; this package aims to bridge this gap.

Chapter 5: Large-Scale Stochastic Sampling from the Probability Simplex

This chapter is conference proceedings appearing in "Advances in Neural Information Processing Systems" in 2018, with co-authors Paul Fearnhead, Emily B. Fox and Christopher Nemeth. The abstract of the publication is given below.

Stochastic gradient Markov chain Monte Carlo (SGMCMC) has become a popular method for scalable Bayesian inference. These methods are based on sampling a discrete-time approximation to a continuous-time process, such as the Langevin diffusion. When applied to distributions defined on a constrained space, such as the simplex, the time-discretisation error can dominate when we are near the boundary of the space. We demonstrate that while current SGMCMC methods for the simplex perform well in certain cases, they struggle with *sparse simplex spaces*; when many of the components are close to zero. However, most popular large-scale applications of Bayesian inference on simplex spaces, such as network or topic models, are sparse. We argue that this poor performance is due to the biases of SGMCMC caused by the discretisation error. To get around this, we propose the stochastic CIR process, which removes all discretisation error, and we prove that samples from the stochastic CIR process are asymptotically unbiased. Use of the stochastic CIR process within an SGMCMC algorithm is shown to give substantially better performance for a topic model and a Dirichlet process mixture model than existing SGMCMC approaches.

Chapter 2

Monte Carlo Methods and SGMCMC

Many statistical and machine learning problems can be reduced to the calculation of an expectation with respect to a probability distribution. The main problem that then needs to be overcome is that these expectations can rarely be calculated analytically. Monte Carlo methods use the fact that these expectations can be simply approximated when the probability distribution can be simulated from. In Section 2.1, we explain the Monte Carlo procedure. While this simplifies the problem, often the underlying probability distribution is difficult to simulate from, especially in the Bayesian paradigm. In light of this, Section 2.2 details Markov chain Monte Carlo methods (MCMC), which can be used to simulate from a large class of probability distributions. The most popular scalable MCMC methods, stochastic gradient Markov chain Monte Carlo (SGMCMC), are based on continuous-time Itô processes, so in Section 2.3.2 we provide an introduction to these processes, as well as the numerical

approximation procedure which forms the basis for many of these algorithms. Finally in Section 2.4 we detail SGMCMC methods, which form the basis for the rest of this thesis, and are some of the most popular scalable MCMC samplers. We also provide a comparison of some popular SGMCMC methods to a class of competitor algorithms known as divide-and-conquer MCMC. This forms the first contribution of this thesis. Monte Carlo is a large and varied topic, so only the topics necessary for this thesis are presented here. For a more thorough treatment of standard MCMC, please see Robert and Casella (2004); Meyn and Tweedie (1993a); for a more thorough treatment of Itô processes and their approximation we refer the reader to Øksendal (2003); Kloeden and Platen (1992); Khasminskii (2011).

2.1 Monte Carlo

Many statistical and machine learning problems can be reduced to the calculation of the expectation of a function. Let θ be a random variable taking values in some topological space Θ with distribution π (i.e. $\mathbb{P}(\theta \in A) = \pi(A)$). Denote the Borel σ-algebra for Θ by $\mathcal{B}(\Theta)$. Note we use some simple measure-theoretic concepts (see e.g. Williams, 1991) to make notation clearer, and this allows us to avoid multiple definitions on different classes of Θ; but this thesis aims to be light on measure theory.

Most statistical quantities of interest can be reduced to

$$\bar{\psi} := \mathbb{E}_\pi \left[\psi(\theta)\right] = \int_\Theta \psi(\theta)\pi(d\theta), \tag{2.1.1}$$

where $\psi : \Theta \to \mathbb{R}^p$ is some $\mathcal{B}(\Theta)$-measurable function, referred to as the test function. Typically $\bar{\psi}$ cannot be calculated analytically, and standard numerical approximation

methods, such as quadrature, suffer from the curse of dimensionality. Monte Carlo methods get around this issue by assuming we can simulate from π. Let $\theta_1, \ldots, \theta_M$ be a sequence of independent, identically distributed simulations from π. Then the Monte Carlo estimate of $\bar{\psi}$ is defined by

$$\hat{\psi}_M = \frac{1}{M} \sum_{m=1}^{M} \psi(\theta_m). \tag{2.1.2}$$

This estimate has a number of desirable statistical properties. The strong law of large numbers can be immediately applied to show that as $M \to \infty$, $\hat{\psi}_M$ converges almost surely (a.s.) to $\bar{\psi}$, i.e.

$$\hat{\psi}_M \xrightarrow{\text{a.s.}} \bar{\psi}, \qquad \text{as } M \to \infty.$$

Similarly, suppose $\mathbb{V}\text{ar}[\psi(\theta)] = \sigma^2 < \infty$, then the central limit theorem can be applied to show that,

$$\sqrt{M}(\hat{\psi}_M - \bar{\psi}) \xrightarrow{\mathcal{D}} N(0, \sigma^2), \qquad \text{as } M \to \infty \tag{2.1.3}$$

where $\xrightarrow{\mathcal{D}}$ denotes convergence in distribution.

2.2 Markov Chain Monte Carlo

The Monte Carlo method assumes that π can simulated from, but often this is not possible, especially when π is multivariate. Markov chain Monte Carlo (MCMC) aims to counteract this by introducing a way to produce a stochastic process that converges to π. The main disadvantage of this method is that the draws from this stochastic process are no longer independent, but alternative convergence results exist for these methods (see Meyn and Tweedie, 1993a; Robert and Casella, 2004). Before

we introduce specific MCMC algorithms, we first need to introduce some results for Markov chains and stochastic stability.

2.2.1 Markov Chains and Stochastic Stability

Let θ_m, $m = 1, \ldots, M$, be a discrete-time stochastic process taking values in Θ. Then this stochastic process is a Markov chain if it satisfies the Markov property; namely the future state θ_{m+1} is independent of previous states given the value of the current state $\theta_m = \vartheta$. For notational convenience it is common to define a quantity known as the Markov kernel $K : (\Theta, \mathcal{B}(\Theta)) \to [0, 1]$ as the following conditional probability

$$K(\vartheta, A) = \mathbb{P}(\theta_{m+1} \in A \,|\, \theta_m = \vartheta).$$

Then the Markov property can be stated as follows

$$K(\vartheta_m, A) = \mathbb{P}(\theta_{m+1} \in A \,|\, \theta_m = \vartheta_m) = \mathbb{P}(\theta_{m+1} \in A \,|\, \theta_m = \vartheta_m, \ldots, \theta_1 = \vartheta_1).$$

We will also use the shorthand that $K^m(\vartheta, A) = \underbrace{K \circ \cdots \circ K}_{m}(\vartheta, A)$; and that for some function ψ taking inputs in Θ, $K\psi(\vartheta) = \int_\Theta K(\vartheta, d\theta)\psi(\theta)d\theta$.

Since we eventually wish to construct Markov chains that converge to the desired π, we need some way of assessing this. Before we can do this, we need some definitions. A Markov chain is defined to be stationary if the distribution of θ_m does not depend on m, i.e. the Markov chain is drawn from a single distribution. An invariant distribution π of a Markov chain has the property that $\theta_{m-1} \sim \pi \implies \theta_m \sim \pi$. A Markov chain with kernel K has invariant distribution π if the following condition holds (Meyn and Tweedie, 1993a; Geyer, 2005) referred to as detailed balance or reversibility

$$\int_B \pi(d\vartheta)K(\vartheta, A) = \int_A \pi(d\vartheta)K(\vartheta, B), \qquad \text{for all } A, B \in \mathcal{B}(\Theta).$$

Now suppose we have a desired π, and wish to construct a Markov chain with kernel K that converges to sampling from π. Then we need to check two things: that the Markov chain converges to stationarity, that the stationary distribution of this Markov chain is uniquely π. If we know that the Markov chain leaves π invariant, then there are two further properties that ensure this is the case: Harris recurrence and aperiodicity. If Harris recurrence holds then this ensures the invariant distribution is unique. A Markov chain is Harris recurrent if there exists a non-zero, σ-finite measure φ on $\mathcal{B}(\Theta)$, such that for all for all $A \in \mathcal{B}(\Theta)$, with $\varphi(A) > 0$; and for all $\vartheta \in \Theta$, a chain starting from ϑ will eventually reach A with probability one (Meyn and Tweedie, 1993a; Geyer, 2005). A Harris recurrent Markov chain is aperiodic if there does not exist an integer $b > 1$, and disjoint subsets $B_1, \dots B_b \in \mathcal{B}(\Theta)$ such that, for all $i = 1, \dots, b$ we have $\varphi(B_i) > 0$ and $K(\vartheta, B_i) = 1$, when $\vartheta \in B_j$ for $j = i - 1 \mod b$ (Meyn and Tweedie, 1993a; Geyer, 2005).

Once it is established that a Markov chain is Harris recurrent and aperiodic, then desirable properties similar to the results for Monte Carlo presented in the previous section can be established. Let $\theta_1, \dots, \theta_M$ be a Harris recurrent, aperiodic Markov chain. Let $\hat{\psi}_M$ be as defined in (2.1.2). Then

$$\hat{\psi}_M \xrightarrow{\text{a.s.}} \bar{\psi}, \qquad \text{as } M \to \infty, \tag{2.2.1}$$

for any starting distribution λ. A central limit result for MCMC, similar to standard Monte Carlo, can also be derived (Robert and Casella, 2004; Geyer, 2005).

It can also be shown that the distribution defined by the Markov chain converges to π. These results are important for deriving convergence bounds for SGMCMC

methods to the target π, so we will outline these results. First we need to describe the total variation metric, used to calculate the distance between two probability measures. A measure can be decomposed into its positive and negative parts for any set $A \in \Theta$ as $\mu(A) = \mu^+(A) - \mu^-(A)$, where $\mu^+(A), \mu^-(A)$ are positive measures with disjoint support. The total variation norm of some measure μ, can then be defined by

$$\|\mu\|_{TV} = \mu^+(A) + \mu^-(A).$$

If the distribution defined by θ_m converges in total variation to π given any initial distribution λ, then it is said to be ergodic; i.e.

$$\left\| \int \lambda(d\vartheta) K^M(\vartheta, \cdot) - \pi \right\|_{TV} \xrightarrow{M \to \infty} 0.$$

This property holds if θ_m is Harris recurrent and aperiodic (Meyn and Tweedie, 1993a; Geyer, 2005).

Results on the convergence of SGMCMC methods require a stronger condition though, known as geometric ergodicity (Meyn and Tweedie, 1993a; Geyer, 2005). Geometric ergodicity bounds the non-asymptotic total variation distance. A Markov chain θ_m is said to be geometrically ergodic if there exists a function $\beta : \Theta \to \mathbb{R}_+$, with $\beta(\theta) < \infty$ π-a.e.[1], and a constant $\rho < 1$, such that

$$\|K^m(\vartheta, \cdot) - \pi\|_{TV} \leq \beta(\vartheta)\rho^m, \qquad \vartheta \in \Theta.$$

Geometric ergodicity can be verified using a 'drift condition,' or Lyapunov–Foster condition. This relies on the existence of a norm-like function V and a petite set

[1] Given a measure π on $\mathcal{B}(\Theta)$, a property holds almost everywhere (π-a.e.) if there exists $N \in \mathcal{B}(\Theta)$ such that $\pi(N) = 0$ and the property holds for all $\vartheta \in \Theta \setminus N$.

C. The norm-like function V has the properties that $V(\theta) \geq 1$ and $V(\theta) \to \infty$ as $\|\theta\| \to \infty$. It plays a similar role to Lyapunov functions, introduced in Section 2.3; which are useful for deriving convergence results for SGMCMC. A set C is petite if there exists a probability distribution a, defined over $\mathbb{N}$; a constant $\delta > 0$; and a probability measure Q, defined over θ; such that

$$\sum_{m=0}^{\infty} a(m) K^m(\vartheta, A) \geq \delta Q(A), \qquad \vartheta \in \Theta.$$

To show a Markov chain is geometrically ergodic we then need a norm-like function V, a petite set C and constants $\lambda < 1$ and $b < \infty$ such that

$$KV(\vartheta) \leq \lambda V(\vartheta) + b\mathbf{1}_C, \qquad \vartheta \in \Theta.$$

This is known as a geometric drift condition. Drift conditions also exist to ensure a variety of properties of the Markov chain, including Harris recurrence (see e.g. Meyn and Tweedie, 1992)

Geometric ergodicity can be used to ensure a central limit theorem (CLT) holds for the Markov chain, similar to the CLT for Monte Carlo (2.1.3). In particular, let $\psi : \Theta \to \mathbb{R}$ be some test function of interest, and assume that $\mathbb{E}_\pi[(\psi(\theta))^{2+\delta}] < \infty$ for some $\delta > 0$. If a Markov chain θ_m with stationary distribution π is geometrically ergodic, as usual define $\bar{\psi} = \mathbb{E}_\pi[\psi(\theta)]$, then

$$\frac{1}{\sqrt{M}} \sum_{m=1}^{M} \left(\psi(\theta_m) - \bar{\psi}\right) \xrightarrow{\mathcal{D}} N(0, \sigma^2), \qquad \text{as } M \to \infty; \tag{2.2.2}$$

for some $\sigma^2 < \infty$ (see e.g. Meyn and Tweedie, 1992; Roberts and Rosenthal, 2004).

2.2.2 Gibbs Update

Now that we have covered the Markov chain background required, we introduce some popular transition kernels K used to sample from a given π. The Gibbs sampler (Geman and Geman, 1984) is a particularly simple Markov kernel used for multiple parameter problems. Suppose we are able to divide a multivariate $\theta \in \Theta$ into components $j = 1, \ldots, d$, such that $\theta = (\theta_1, \ldots, \theta_d)$. For example, if we have interest in the target $\pi(\mu, \sigma) = N(\mu, \sigma\mathbf{I})$, where $\mathbf{I}$ is the identity matrix; then we might divide θ into two parameters $\theta = (\mu \in \mathbb{R}^2, \sigma \in \mathbb{R})$ (notice that θ_j need not necessarily be a scalar).

Then for each component of the partition, j, the Gibbs sampler updates θ_j assuming the rest $\theta_{-j} = (\theta_1, \ldots, \theta_{j-1}, \theta_{j+1}, \ldots, \theta_d)$ is fixed at the previous state ϑ. To do this it uses the conditional distribution of the desired target, $\pi(\cdot \,|\, \theta_{-j} = \vartheta_{-j})$. For each component j, the Gibbs update kernel can be defined as follows

$$K_j(\vartheta, A) = \mathbf{1}_{\vartheta_{-j} \in A_{-j}} \pi(A_j \,|\, \theta_{-j} = \vartheta_{-j}),$$

where $A = (A_1, \ldots, A_d)$ and $A_{-j} = (A_1, \ldots, A_{j-1}, A_{j+1}, \ldots, A_d)$.

To show this update leaves π invariant we can use properties of the conditional expectation (Geyer, 2005). First notice that $K_j(\vartheta, A) = \mathbf{1}_{\vartheta_{-j} \in A_{-j}} \mathbb{E}[\mathbf{1}_{\theta_j \in A_j} \,|\, \theta_{-j} = \vartheta_{-j}] = \mathbb{E}[\mathbf{1}_{\vartheta_j \in A_{-j}} \mathbf{1}_{\theta_j \in A_j} \,|\, \theta_{-j} = \vartheta_{-j}]$, so that by the law of total expectation

$$\int_\Theta \pi(d\vartheta) K(\vartheta \,|\, A) = \mathbb{E}[\mathbb{E}[\mathbf{1}_{\vartheta_{-j} \in A_{-j}} \mathbf{1}_{\theta_j \in A_j} \,|\, \theta_{-j} = \vartheta_{-j}]] = \pi(A)$$

We conclude the Gibbs sampler leaves π invariant. A necessary condition for the Gibbs sampler to be Harris recurrent is that each component j is updated frequently enough. The two most popular ways of doing this are either to update every j at

each iteration, but in any order; or to pick a j with probability $1/d$, and update using kernel K_j. Additional conditions on the state space Θ and K ensure the Gibbs sampler is Harris recurrent (see Meyn and Tweedie, 1993a; Robert and Casella, 2004; Geyer, 2005, for details). A Harris recurrent Gibbs sampler is always aperiodic.

Gibbs updates require no user tuning, and can make large moves since updating component j does not depend on θ_j, just θ_{-j}. However, the Gibbs sampler can mix slowly when components are highly dependent. Another major disadvantage is that it requires the calculation of the conditional distributions $\pi(\cdot|\theta_{-j})$. While there are many important machine learning and statistical problems where this is possible, there are also many problems where it is not.

2.2.3 Metropolis–Hastings Update

Commonly the only information we have about π is an unnormalised density. We say a function $h : \Theta \to \mathbb{R}$ is an unnormalised density if it has the following properties: h is nonnegative; and $0 < \int_\Theta h(\theta)d\mu < \infty$, where μ is defined to be the Lebesgue measure.

An important example where the only information we have about π is its unnormalised density is in Bayesian inference. Suppose we have data $\mathbf{x} = \{x_i\}_{i=1}^N$ which depends on a parameter θ through the density $p_i(\theta) := p(x_i|\theta)$. We assign θ a prior density $p_0(\theta)$. Then, defining the Lebesgue measure by μ, the posterior density $p(\theta)$ is given by

$$p(\theta) = \frac{\prod_{i=0}^N p_i(\theta)}{Z}, \qquad Z = \int_\Theta \prod_{i=0}^N p_i(\theta)d\mu, \tag{2.2.3}$$

where Z is referred to as the normalising constant. A fundamental problem in Bayesian inference is that the integration to find Z is rarely tractable. This means typically we only know the posterior up to the unnormalised density $h(\theta) = \prod_{i=0}^{N} p_i(\theta)$.

The Metropolis–Hastings algorithm aims to get around this issue by defining a Markov chain that converges to sampling from π and only relies on being able to evaluate h, where h is an unnormalised density of π. The Metropolis–Hastings algorithm (MH) was first developed by Metropolis et al. (1953), with an important later development by Hastings (1970), as well as Green (1995).

The main idea behind MH is to find a distribution Q, that is easy to simulate from, referred to as a proposal distribution. Then to correct simulations from this distribution so that the resulting process θ_m converges to sampling from π. More formally, suppose the Markov chain is currently at a state θ. A proposal distribution, $Q(\cdot|\theta)$, is used in order to simulate a new proposal state θ' given the current state θ. Suppose this proposal distribution admits a density $q(\theta'|\theta)$ with respect to the Lebesgue measure μ; then the Metropolis–Hastings algorithm proceeds as follows: a candidate state θ' is simulated from Q, this candidate state is then accepted with probability

$$\alpha(\theta', \theta) = 1 \wedge \frac{h(\theta')q(\theta|\theta')}{h(\theta)q(\theta'|\theta)},$$

where $c_1 \wedge c_2$ denotes the minimum between numbers c_1 and c_2. If the candidate value θ' is accepted, then the next state in the Markov chain is defined to be θ'. Otherwise it is discarded and the next state is defined to be θ, the same as the current state of the chain. Notice that α is invariant to changing h by a multiplicative constant, since

these constants cancel in the ratio of terms; this is why any unnormalised density of π can be used to implement this algorithm.

To check the MH algorithm leaves π invariant we will show that the MH kernel satisfies detailed balance. Because the accept-reject step can lead to the algorithm staying in the current state, the Markov kernel for the MH algorithm is in the form of a sum

$$K(\theta, A) = \left[1 - \int_A q(\theta'|\theta)\alpha(\theta, \theta')\mu(d\theta')\right] \mathbf{1}(\theta \in A) + \int_A q(\theta'|\theta)\alpha(\theta, \theta')\mu(d\theta'),$$

where $\mathbf{1}(\theta \in A)$ is 1 if $\theta \in A$ and 0 otherwise. We demonstrate detailed balance informally by showing $p(\theta)K(\theta, d\theta') = p(\theta')K(\theta', d\theta)$, for a more formal proof see Robert and Casella (2004). We make use of the following identities, which can easily be checked: $p(\theta)q(\theta'|\theta)\alpha(\theta, \theta') = p(\theta')q(\theta|\theta')\alpha(\theta', \theta)$, and $p(\theta)\delta_{\theta'}(\theta) = p(\theta')\delta_{\theta}(\theta')$. We can apply these identities to check detailed balance as follows

$$\begin{aligned} p(d\theta)K(\theta, d\theta') &= p(d\theta)\left[1 - q(\theta'|\theta)\alpha(\theta, \theta')\right]\delta_{\theta'}(\theta) + p(d\theta)q(\theta'|\theta)\alpha(\theta, \theta') \\ &= p(d\theta')\left[1 - q(\theta|\theta')\alpha(\theta', \theta)\right]\delta_{\theta}(\theta') + p(d\theta')q(\theta|\theta')\alpha(\theta', \theta) \\ &= p(d\theta')K(\theta', d\theta). \end{aligned}$$

Further properties on the state space Θ and the proposal distribution ensure that the Markov chain is Harris recurrent and aperiodic (Meyn and Tweedie, 1993a; Robert and Casella, 2004; Geyer, 2005). The Gibbs sampler can be seen to be a special case of the Metropolis–Hastings method. We can see this by implementing the MH algorithm for each component j of the partition of Θ in turn. If the proposal distribution for this component Q_j is set to be $\pi(\cdot|\theta_{-j})$, it can be shown that the corresponding

Metropolis–Hastings update has acceptance probability 1 (Robert and Casella, 2004; Geyer, 2005).

If our proposal and state space ensure Harris recurrence and aperiodicity then the MH sample is guaranteed to satisfy the strong law of large numbers result (2.2.1). For practical purposes though, we only simulate from our chain for a finite amount of time, so to ensure good properties of the chain we need to choose a good proposal distribution. The MSE of the chain tends to be controlled by the autocovariance, so the best proposals lead to chains with low autocovariance. In the next section we discuss how to construct efficient proposals for the MH algorithm using continuous-time Markov processes.

2.3 Itô Processes for MCMC

Many MCMC algorithms, including SGMCMC, rely on the theory of continuous-time Markov processes, in particular Itô diffusions. In this section we review results about these processes, so that the necessary grounding has been discussed when we summarise SGMCMC.

2.3.1 Markov Processes and Stochastic Stability

We refer to Markov processes as the analog to Markov chains in continuous-time. Let $\{\theta_t, t \in \mathbb{R}_+\}$ be a continuous-time stochastic process taking values in $\mathbb{R}^d$. Define $\mathcal{F}_t$ to be the σ-algebra generated by $\{\theta_s, 0 \leq s \leq t\}$ (this can simply be thought of as all the possible paths θ_t could take up to time t); then θ_t is Markov (see e.g. Khasminskii,

2011) if, for all $A \in \mathcal{B}(\mathbb{R}^d)$, $0 \leq s \leq t$,

$$\mathbb{P}(\theta_t \in A \,|\, \mathcal{F}_s) = \mathbb{P}(\theta_t \in A \,|\, \theta_s). \tag{2.3.1}$$

Our interest will be in Markov processes that are time-homogeneous, meaning $\mathbb{P}(\theta_t \in A \,|\, \theta_s) = \mathbb{P}(\theta_{t-s} \in A \,|\, \theta_0)$. This allows us to use the following shorthand for the transition probability $P(\theta_t \in A \,|\, \theta_0 = \vartheta) = K_t(\vartheta, A)$. Provided it exists, we can define the transition density $p_t(\varphi \,|\, \vartheta)$ of the Markov process by

$$K_t(\vartheta, A) = \int_A p_t(\varphi \,|\, \vartheta) d\varphi.$$

As in discrete-time Markov chains, we are often interested in the behaviour of a test function ψ under the dynamics of θ_t. We define

$$K_t\psi(\vartheta) = \int K_t(\vartheta, dy)\psi(y).$$

This allows us to define the operator known as the *generator* $\mathcal{A}$ (see e.g. Khasminskii, 2011) of the process, applied to a function ψ (provided the limit exists), as

$$\mathcal{A}\psi(\vartheta) = \lim_{t \to +0} \frac{K_t\psi(\vartheta) - \psi(\vartheta)}{t}.$$

It can be shown the generator fully defines the Markov process (see e.g. Khasminskii, 2011). It can be visualised as describing the infinitesimal evolution of the process.

Similar to discrete-time Markov chains, we are interested in convergence of θ_t to a stationary distribution π. Necessary conditions for θ_t to be stationary are: for $A, B \in \mathcal{B}(\mathbb{R}^d)$, and for all $h > 0$, the events $\{\theta_t \in A\}$ and $\{\theta_t \in A, \theta_{t+h} \in B\}$ are independent of t; that the initial distribution π_0 is invariant (see e.g. Khasminskii, 2011); i.e. for every $s > 0$,

$$\pi_0(A) = \int \pi_0(d\vartheta) K_s(\vartheta, A), \qquad \theta_0 \sim \pi_0.$$

The idea behind finding stationary distributions for Markov processes is to again derive a law of large numbers for θ_t. Specifically, given some test function ψ, we desire results of the form

$$\frac{1}{T}\int_0^T \psi(\theta_t)dt \xrightarrow{a.s.} \bar{\psi}, \qquad T \to \infty.$$

Similarly to the law of large numbers for Markov chains, this relies on existence and uniqueness of the stationary solution to the chain (see e.g. Khasminskii, 2011). Conditions for this to be the case are investigated for Markov processes in Meyn and Tweedie (1993b,c). Similarly to Markov chains, a sufficient condition for the existence and uniqueness of a stationary solution π is Harris recurrence. The definition of Harris recurrence is the same as for Markov chains, i.e. there exists a measure ν on Θ such that the probability a chain θ_t ever hits a set A is one, for all $\vartheta \in \Theta$ and $A \in \mathcal{B}(\Theta)$ with $\nu(A) > 0$ (Meyn and Tweedie, 1993c). Moreover, if a Markov process θ_t is Harris recurrent and time points $t_1, \dots t_M$ can be chosen such that the Markov chain θ_{t_m} is also Harris recurrent, then θ_t is ergodic, i.e. $\lim_{t\to\infty} \|K_t(\vartheta, \cdot) - \pi\|_{TV} = 0$ for all $\vartheta \in \Theta$.

In Meyn and Tweedie (1993c), sufficient conditions are derived for desirable Markov process properties, such as Harris recurrence and geometric ergodicity, using drift conditions; similar to the geometric drift condition for Markov chains outlined in Section 2.2.1. As for Markov chains, the drift conditions rely on the existence of a norm-like or Lyapunov function V, satisfying the usual $V(\vartheta) \geq 1$, for all $\vartheta \in \Theta$, and $\lim_{\|\vartheta\|\to\infty} V(\vartheta) = \infty$. The generator $\mathcal{A}$ then acts on this function to obtain the drift conditions[2].

[2]Meyn and Tweedie (1993c) consider an extended generator, but we omit this for brevity.

Many results explored later rely on Markov processes that are geometrically ergodic; so we outline geometric drift conditions for Markov processes. Meyn and Tweedie (1993c) show that a Markov process is ergodic (i.e. converges in TV distance) if the following conditions hold: it is Harris recurrent; there exists time points $t_1, \ldots, t_M$, such that all compact sets are petite for the Markov chain θ_{t_m}; and the stationary solution is finite. Geometric ergodicity for Markov processes requires a stronger norm than the total variation norm, known as the ψ-norm, defined by $\|\mu\|_\psi = \sup_{|g|\leq\psi} |\mu(g)|$, where μ is some measure over $\mathcal{B}(\Theta)$. This ensures that $\mathbb{E}[\psi(\theta_t)]$ converges to $\bar{\psi}$ and is bounded. A Markov process is ψ-geometrically ergodic if there exists $\rho < 1$ and a function $\beta : \Theta \to \mathbb{R}_+$ bounded π-a.e., such that

$$\|K_t(\vartheta, \cdot) - \pi\|_\psi \leq \beta(\vartheta)\rho^t.$$

Meyn and Tweedie (1993c) show that, given a Markov process θ_t, if the conditions for ergodicity hold, and there is a norm-like function V and constants $d < \infty$ and $c > 0$ such that

$$\mathcal{A}V(\vartheta) \leq -cV(\vartheta) + d, \qquad \vartheta \in \Theta;$$

then θ_t is ψ-geometrically ergodic, with $\psi = V + 1$.

2.3.2 Itô Processes and the Langevin Diffusion

Many efficient proposals for the MH algorithm rely on Itô processes (see e.g. Roberts and Rosenthal, 1998; Neal, 2010), which are Markov processes defined as a solution to a stochastic differential equation. A stochastic differential equation (SDE) is a differential equation which has at least one term that is a stochastic process (see e.g.

Kloeden and Platen, 1992; Øksendal, 2003). Itô processes are based on a particular continuous-time stochastic process referred to as a Wiener process. Let $\{W_t, t \in \mathbb{R}_+\}$ be a Wiener process, then the following properties hold:

- $W_0 = 0$ with probability 1;
- (independent increments) $W_{t+s} - W_t$ is independent of W_u for $0 < u < t$;
- $W_{t+s} - W_t \sim N(0, s)$;
- W_t has continuous paths with t (a.s.).

To setup a differential equation based on this process, we need to be able to integrate with respect to its derivative. A difficulty of this is that the process is differentiable nowhere, which leads traditional integration procedures, such as the Riemann–Stieltjes integral, to fail. The Itô integral gets around this by defining an alternative integral with respect to the Wiener process (see e.g. Kloeden and Platen, 1992; Øksendal, 2003). Other integrals with respect to the Wiener process exist, for example the Stranovich integral; but we focus on the Itô integral as the most common in the MCMC literature. Let $\{\theta_t, t \in \mathbb{R}_+\}$ be a continuous-time stochastic process, then the Itô integral of θ_t with respect to W_t is defined by

$$\int_0^t \theta_s dW_s = \lim_{M \to \infty} \sum_{m=1}^{M} \theta_{t_{m-1}} \left[W_{t_m} - W_{t_m - 1} \right],$$

where $0 = t_0 < \cdots < t_M = t$ is a partition of $[0, t]$, such that as $M \to \infty$ the gap between any two consecutive partition points goes to 0. Commonly, interest is in a d-dimensional Wiener process; this is simply defined as a vector of independent scalar Wiener processes; the integral is then performed coordinate wise.

We are now able to define the differential form of an Itô process (see e.g. Kloeden and Platen, 1992; Øksendal, 2003). Define two functions $b : \mathbb{R}^d \to \mathbb{R}^d$ and $\sigma : \mathbb{R}^d \to \mathbb{R}^{d\times d}$, referred to as the drift and diffusion terms respectively. An Itô process is a continuous-time stochastic process $\{\theta_t, t \in \mathbb{R}^d\}$ that takes the following form

$$\theta_t = \theta_0 + \int_0^t b(\theta_s)ds + \int_0^t \sigma(\theta_s)dW_s. \tag{2.3.2}$$

This means the Itô process is fully specified by three things: the starting point θ_0; the 'deterministic' drift term determined by b; and the stochastic diffusion term, determined by σ. Notice that in an Itô process, the terms b and σ do not depend directly on the time t. A solution to (2.3.2) does not necessarily exist, so normally conditions are imposed on b and σ to ensure the solution exists, and that it is unique (see e.g. Kloeden and Platen, 1992; Øksendal, 2003). Sufficient conditions for an Itô process to have a unique solution are that there exists a constant $C \in \mathbb{R}_+$ such that the following holds:

- (Lipschitz) $\|b(\theta) - b(\theta')\| + \|\sigma(\theta) - \sigma(\theta')\|_{L_{21}} \leq C \|\theta - \theta'\|$;
- (Linear Growth) $\|b(\theta)\| + \|\sigma(\theta)\|_{L_{21}} \leq C(1 + \|\theta\|)$;

where $\|\cdot\|$ is the Euclidean norm; and $\|\cdot\|_{L_{21}}$ is the L_{21} matrix norm. Given a matrix A, we define the L_{21} norm as $\|A\|_{L_{21}} := \sum_{j=1}^d \left[\sum_{i=1}^d a_{ij}^2\right]^{\frac{1}{2}}$, i.e. the sum of the Euclidean norms of the columns. An Itô process that satisfies these conditions is often referred to as an Itô diffusion (Øksendal, 2003). However, these conditions are quite restrictive in practice, so often these assumptions are relaxed; as a result we will consider general Itô processes, assuming the solution exists. The SDE whose solution is the Itô process of

interest (2.3.2) is often written in a shorthand similar to that for ordinary differential equations,

$$d\theta_t = b(\theta_t)dt + \sigma(\theta_t)dW_t.$$

We now explore some of the properties of the Itô process. An Itô process satisfies the Markov property (2.3.1), so is a Markov process (see e.g. Øksendal, 2003; Khasminskii, 2011); it is also time homogeneous. Due to the alternative integration procedure for W_t, an Itô process has its own version of the chain rule. This is referred to as Itô's Lemma (see e.g. Øksendal, 2003; Khasminskii, 2011), and we use it repeatedly.

Lemma 2.3.1. *(Itô's Lemma) Let θ_t be a 1-dimensional Itô process of the form (2.3.2). Let $\psi : \mathbb{R} \to \mathbb{R}$ be a twice differentiable function; then $\psi_t := \psi(\theta_t)$ is also an Itô process, defined by the following equation*

$$d\psi_t = \left[b(\theta_t)\frac{d\psi}{d\theta}(\theta_t) + \frac{\sigma^2(\theta_t)}{2}\frac{d^2\psi}{d\theta^2}(\theta_t) \right] dt + \sigma(\theta_t)\frac{d\psi}{d\theta}(\theta_t)dW_t. \tag{2.3.3}$$

Equivalent versions exist for multi-dimensional diffusions (see e.g. Øksendal, 2003; Khasminskii, 2011). Even if θ_t is an Itô diffusion, ψ_t is only guaranteed to be an Itô process, not an Itô diffusion (see e.g. Øksendal, 2003). Itô's Lemma can be used to derive the generator $\mathcal{A}$ for an Itô diffusion in terms of the coefficients b and σ (see e.g. Øksendal, 2003; Khasminskii, 2011, for details). For a twice differentiable function $\psi : \mathbb{R}^d \to \mathbb{R}^d$, the generator has the following form

$$\mathcal{A}\psi(\vartheta) = \sum_{i=1}^{d} b_i(\vartheta)\frac{\partial\psi}{\partial\vartheta_i} + \frac{1}{2}\sum_{i=1}^{d}\sum_{j=1}^{d}(\sigma\sigma^T)_{ij}(\vartheta)\frac{\partial^2\psi}{\partial\vartheta_i\partial\vartheta_j}. \tag{2.3.4}$$

The transition density $p_t(\varphi \,|\, \vartheta)$ of an Itô process, assuming it exists, can be found

by solving the Fokker-Planck equation (see e.g. Khasminskii, 2011), a partial differential equation as follows

$$\frac{\partial}{\partial t}p_t(\varphi \mid \vartheta) = -\sum_{i=1}^{d}\frac{\partial}{\partial \varphi_i}[b_i(\varphi)p_t(\varphi \mid \vartheta)] + \frac{1}{2}\sum_{i=1}^{d}\sum_{j=1}^{d}\frac{\partial^2}{\partial \varphi_i \partial \varphi_j}\left[(\sigma\sigma^T)_{ij}(\varphi)p_t(\varphi \mid \vartheta)\right]. \tag{2.3.5}$$

If a unique stationary distribution π exists, then the Fokker-Planck equation (2.3.5) can be used to calculate its density exactly, by solving it assuming $p_t(\varphi|\vartheta) := p(\varphi)$; i.e. assuming the transition density is independent of time, so that $\partial_t p(\varphi) = 0$.

Unfortunately, the Fokker-Planck equation is rarely solvable, though there are some important cases where it can which we shall detail later. In these cases, stochastic stability results, such as those detailed in Section 2.3.1, can be used to investigate existence and uniqueness of a stationary solution. Stochastic stability results specifically for Itô processes are well studied. Similar to the general results of Section 2.3.1, the results generally rely on the existence of norm-like functions V referred to as Lyapunov functions (see e.g. Khasminskii, 2011). Khasminskii (2011) detail sufficient conditions to ensure existence and uniqueness of the stationary distribution, and show how to verify them using Lyapunov functions. The conditions are as follows: suppose there exists a bounded, open domain $B \subset \mathbb{R}^d$ with regular boundary[3] Γ, then the conditions of Khasminskii (2011) are as follows:

- In the domain, and some neighbourhood of B, the smallest eigenvalue of the diffusion matrix $\sigma\sigma^T(\theta)$ is bounded away from 0.

- If $\vartheta \in \mathbb{R}^d \setminus B$, the mean time τ for a path from ϑ to the set B is finite and

[3]A regular boundary is a standard concept in the study of PDEs (see e.g. Petrovskii, 1954).

$\sup_{\theta_0 \in A} \mathbb{E}\left[\tau \,|\, \theta_0 = \vartheta\right] < \infty$ for all compact subsets $A \subset \mathbb{R}^d$.

The Langevin Diffusion and Other Important Itô Processes

In this section, we detail important examples of Itô processes which have a known stationary distribution. An important diffusion in the MCMC literature is the Langevin diffusion (LD), which forms the basis of one of the most popular SGMCMC samplers (stochastic gradient Langevin dynamics), as well as numerous other MCMC algorithms (Roberts and Tweedie, 1996). Given a target distribution π, a LD is guaranteed to have stationary solution π. This means, provided the process is ergodic, simulating from the LD will target π. Suppose π admits a density p with respect to the Lebesgue measure, and define $f(\vartheta) = -\log p(\vartheta)$. Then the Langevin diffusion is defined by the SDE

$$d\theta_t = -\nabla f(\theta_t)dt + \sqrt{2}dW_t. \tag{2.3.6}$$

Because of the form of f, this means the density p only needs to be known up to a normalising constant, which is one of the reasons why LD underlies so many MCMC algorithms.

We can demonstrate π is a solution of (2.3.6) using the Fokker-Planck equation. Note that for the Langevin diffusion $\sigma\sigma^T = 2I$, where I is the identity matrix, so that

$$\begin{aligned}\sum_{i=1}^{d} \partial_{\theta_i}\left[b_i(\theta)p(\theta)\right] &= \sum_{i=1}^{d} \partial_{\theta_i}\left[(\partial_{\theta_i} f(\theta))e^{-f(\theta)}\right] \\ &= \sum_{i=1}^{d} \partial^2_{\theta_i}[e^{-f(\theta)}] = \frac{1}{2}\sum_{i=1}^{d} \partial^2_{\theta_i}\left[\sum_{j=1}^{d}(\sigma\sigma^T)_{ij}(\theta)p(\theta)\right],\end{aligned}$$

which shows that the density $p(\theta) = e^{-f(\theta)}$ is a solution of the Fokker-Planck equation. Unfortunately, while it can be easily shown that a stationary solution is π, and

sufficient conditions can be found to show uniqueness and convergence (see Roberts and Tweedie, 1996); the transition density cannot be found in general. The lack of transition density complicates simulating from this process. As a result, there is a vast literature on approximate simulation of Itô processes, with particular emphasis on simulating from the Langevin diffusion (see Section 2.3.3).

There are other Itô processes which admit the general distribution π as a stationary solution. An important example is Hamiltonian dynamics, or underdamped Langevin dynamics (Wang and Uhlenbeck, 1945). Hamiltonian dynamics augments the state space by introducing a term ν taking values in $\mathbb{R}^d$, referred to as the momentum term. This enables Hamiltonian dynamics to incorporate more information about the geometry of the space which improves the mixing of Hamiltonian based MCMC algorithms over Langevin based MCMC. Since Hamiltonian dynamics is based on two parameters ν and θ, it is the solution to a system of SDEs rather than a single SDE (see e.g. Horowitz, 1991; Chen et al., 2014; Leimkuhler and Shang, 2016). Typically the density of the augmented target is set to be $p(\theta, \nu) = e^{-f(\theta)-\frac{1}{2}\nu^T M^{-1}\nu}$, so that marginally $\nu \sim N(0, M)$. Here M is a user-specified matrix known as the mass matrix. The Hamiltonian dynamics are then defined as follows

$$d\theta_t = M^{-1}\nu_t dt \tag{2.3.7}$$

$$d\nu_t = -\nabla f(\theta_t)dt - \beta\nu_t dt + \sqrt{2\beta}M^{\frac{1}{2}}dW_t, \tag{2.3.8}$$

where β is a user-specified constant. Once again, Hamiltonian dynamics cannot be simulated in general, so there is a lot of interest in approximating these dynamics. There is an alternative version of Hamiltonian dynamics that is defined by an ordi-

nary differential equation, rather than a stochastic differential equation (see e.g. Neal, 2010); and it can be shown that these two versions are related (see e.g. Horowitz, 1991; Leimkuhler and Shang, 2016). This version underlies many efficient samplers known collectively as Hamiltonian Monte Carlo (see e.g. Neal, 2010). This includes the sampler NUTS (Hoffman and Gelman, 2014), one of the most popular samplers implemented in the probabilistic programming language STAN (Carpenter et al., 2017).

Apart from Itô processes to simulate from general π, there are also diffusions which simulate from specific distributions, some of which have known transition densities meaning they can be simulated exactly. In fact there exist diffusions with known transition densities for all exponential family distributions (Bibby et al., 2005). Possibly the most common diffusion in this class is the Ornstein-Uhlenbeck process, which admits a normal distribution as its stationary distribution (Øksendal, 2003). Another process in this class, commonly used in the mathematical finance literature is the Cox-Ingersoll-Ross process (Cox et al., 1985), which me make use of in Chapter 5. The stationary distribution of this process is the Gamma distribution.

2.3.3 The Euler–Maruyama Method and ULA

As mentioned in the previous section, there are many Itô processes that have a stationary distribution of π, but cannot be simulated in general; such as the Langevin and Hamiltonian diffusions. This means there is a lot of interest in approximate simulation of Itô processes. In this section, we detail how Euler's method for ODEs can be adapted to generate an approximate sample path from an Itô process (see e.g. Kloeden and Platen, 1992). This method forms the basis of most SGMCMC samplers.

Given an Itô process of the form (2.3.2), the Euler–Maruyama method suggests linearising both the drift and diffusion functions for a small time period h. We will label the Euler–Maruyama approximation to the process θ_t at time $t = mh$ to be θ_m, for $m \in \{1, \ldots, M\}$. Using that $\int_0^h W_s ds = N(0, h)$, this leads to the Euler–Maruyama approximation having the following form

$$\theta_{m+1} = \theta_m + hb(\theta_m) + \sigma(\theta_m)\sqrt{h}\zeta_m, \qquad \zeta_m \sim N(0, 1).$$

The Euler–Maruyama approximation can be easily implemented provided b and σ can be evaluated. Provided h is not too large compared to the typical magnitude of b, the approximation will not diverge to infinity (though this does not guarantee a good approximation).

Applying the Euler–Maruyama method in the case of the Langevin diffusion leads to the unadjusted Langevin algorithm (ULA) as follows

$$\theta_{m+1} = \theta_m + h\nabla f(\theta_m) + \sqrt{2h}\zeta_m. \tag{2.3.9}$$

The stationary distribution of this algorithm, π_h, will be an approximation to the desired posterior π. Roberts and Tweedie (1996) investigate the ergodicity of the Langevin diffusion and ULA. They show, using the results of Meyn and Tweedie (1993a,b), that even in cases when the diffusion is ergodic, the numerical approximation need not be. This means the algorithm will not even converge to the stationary solution π_h, or that π_h may not even exist. This led the statistics community to favour the Metropolis-adjusted Langevin algorithm (MALA), which uses (2.3.9) as a proposal to a Metropolis–Hastings algorithm. This algorithm both has π as its

stationary distribution, and is ergodic whenever the Langevin diffusion is ergodic (Roberts and Tweedie, 1996).

Despite the results of Roberts and Tweedie (1996), there has still been interest quantifying the error of the Euler–Maruyama method, as it forms the basis of more sophisticated Euler type approximations. We will detail these results as they are relevant for SGMCMC methods. Kloeden and Platen (1992), detail a number of well known results on the error of the approximation compared to the diffusion θ_t. These results are known as the strong and weak error. However in MCMC, generally more interest is in the error between $\hat{\psi}_M := \frac{1}{M}\sum_{m=1}^{M}\psi(\theta_m)$ and $\bar{\psi} = \int_{\mathbb{R}^d}\psi(\theta)\pi(d\theta)$, where ψ is some test function; as well as ergodicity results. For this reason we focus on outlining results of this form.

Talay and Tubaro (1990), define sufficient conditions for the Euler–Maruyama scheme to be ergodic and, based on these assumptions, quantify the asymptotic bias of the Euler–Maruyama method, $\lim_{M\to\infty}|\hat{\psi}_M - \bar{\psi}|$. They find this bias to be $O(h)$. Mattingly et al. (2002) use Lyapunov–Foster drift conditions detailed in Sections 2.2.1 and 2.3.1 to find when the Euler–Maruyama approximation will be geometrically ergodic.

Lamberton and Pagès (2002) investigate a Euler–Maruyama scheme with a stepsize that decreases to 0, making use of the Lyapunov ideas for Itô processes. They show that when h is decreased to 0, $\hat{\psi}_M$ converges to $\bar{\psi}$ in the limit $M \to \infty$. They also find that when h_m is set to decrease with $O(m^{-1/3})$, then the bias and RMSE (root mean square error) of $\hat{\psi}_M$ are both $O(M^{-1/3})$. In comparison, standard MCMC methods such as the MH and MALA algorithms have unbiased $\hat{\psi}_M$ and RMSE of $O(M^{-1/2})$.

Let γ be a solution to the Poisson equation, defined by $\mathcal{A}\gamma = \psi - \bar{\psi}$, where $\mathcal{A}$ is the generator of the Itô process of interest. Mattingly et al. (2010) used this equation to study the non-asymptotic bias and MSE of $\hat{\psi}_M$ when the stepsize h is fixed, provided the Euler–Maruyama scheme is ergodic and a solution to the Poisson equation exists. They find the bias of $\hat{\psi}_M$ to be $O(h+\frac{1}{Mh})$ and the MSE to be $O(h^2+\frac{1}{Mh})$. Interestingly, this leads Mattingly et al. (2010) to suggest it is optimal to set h to be $O(M^{-1/3})$, leading to both bias and RMSE $O(M^{-1/3})$; similar to the results of Lamberton and Pagès (2002).

Despite the results of Roberts and Tweedie (1996), there has been renewed interest in the ergodicity of ULA. This is possibly because ULA forms the basis for one of the most popular SGMCMC samplers. In particular, there has been interest in the non-asymptotic convergence of ULA to the target π. Central to this work is the assumption that f strongly convex (i.e. the density, p, of π is strongly log-concave) and smooth. This enables the authors to ensure that ULA is not transient, and derive geometric ergodicity results for the method. More formally, it is assumed that there exists constants $l, L > 0$ such that, for $\theta, \theta' \in \mathbb{R}^d$,

$$f(\theta) - f(\theta') - \nabla f(\theta')^T(\theta - \theta') \geq \frac{l}{2}\left\|\theta - \theta'\right\|^2, \tag{2.3.10}$$

$$\left\|\nabla f(\theta) - \nabla f(\theta')\right\| \leq L\left\|\theta - \theta'\right\|. \tag{2.3.11}$$

The line (2.3.10) corresponds to the assumption that f is l-strongly-convex, while (2.3.11) corresponds to the assumption f is L-smooth. Dalalyan (2016) derives a non-asymptotic bound in the TV distance of the distribution defined by a fixed stepsize ULA and the desired target π; using these assumptions as well as a spectral gap

argument. Durmus and Moulines (2017a) extend these results by considering both decreasing and fixed stepsize schemes; as well as deriving tighter bounds on the TV distance using Foster–Lyapunov conditions detailed in Sections 2.2.1 and 2.3.1. They also consider the case where f is a sum of two functions $f_1 + f_2$, where f_1 is strongly convex and f_2 has bounded L_∞ norm.

Durmus and Moulines (2017b) consider alternative bounds on the Wasserstein distance of order 2, which improve dramatically on previous TV bounds. The Wasserstein distance of order $\alpha \geq 1$, W_α, between two measures μ and ν defined on the probability space $(\mathbb{R}^d, \mathcal{B}(\mathbb{R}^d))$ is defined by

$$W_\alpha(\mu, \nu) = \left[\inf_{\gamma \in \Gamma(\mu,\nu)} \int_{\mathbb{R}^d \times \mathbb{R}^d} \|\theta - \theta'\|^\alpha \, d\gamma(\theta, \theta') \right]^{\frac{1}{\alpha}},$$

where the infimum is with respect to all joint distributions Γ having μ and ν as marginals. Cheng et al. (2018) relaxes the strongly log-concave assumption to the assumption that f is smooth and *locally* strongly convex (i.e. strongly convex outside a ball of finite radius); they analyse the W_1 distance between the distribution defined by the ULA approximation and the target distribution under these assumptions.

2.4 Stochastic Gradient Markov Chain Monte Carlo

In this section we detail stochastic gradient Markov chain Monte carlo (SGMCMC), a popular class of methods which aim to make MCMC methods more scalable to the large data setting. The rest of this thesis provides various contributions to the SGMCMC literature. In this section we also provide a comparison of popular SGMCMC methods to divide-and-conquer MCMC, another popular scalable MCMC method;

these comparisons form the first original contribution of this thesis. The comparisons generally show SGMCMC methods to be more robust than divide-and-conquer MCMC; which forms the motivation for us to contribute to SGMCMC methods for the rest of the thesis.

2.4.1 Background

The basis for SGMCMC is the Euler–Maruyama method. Commonly when inferring an unknown parameter θ using MCMC, the cost of evaluating p (or f) will be $O(N)$, where N is the dataset size. For example, consider the setup of Bayesian inference detailed in Section 2.2, (2.2.3). The unnormalised density of π, $h(\theta) = \prod_{i=0}^{N} p_i(\theta)$ is a product of $N+1$ terms. Similarly, defining $f_i(\theta) := -\nabla \log p_i(\theta)$, then $\nabla f(\theta) = \sum_{i=0}^{N} \nabla f_i(\theta)$; a sum of $N+1$ terms.

The consequence of this is that when implementing a modern MCMC sampler, such as MALA or HMC, there are two steps that are $O(N)$: calculating the proposal, which requires calculating f at the current state; and calculating the acceptance step, which requires calculating p at the proposed state. This leads to MCMC being prohibitively slow for large dataset sizes. To reduce this cost in the big data setting, SGMCMC methods do not calculate an acceptance step, which leads the updates to be more similar to Euler–Maruyama updates; and replace f with a cheap estimate, $\hat{f}$, which can be evaluated at cost $O(n)$, for $n \ll N$. This leads to an algorithm with per-iteration computational cost of $O(n)$.

SGMCMC algorithms were first introduced Welling and Teh (2011), who considered using the Langevin diffusion as the underlying process. They referred to the

algorithm as stochastic gradient Langevin dynamics (SGLD). The SGLD can be derived simply from the ULA algorithm, by replacing f with an unbiased, cheap estimate $\hat{f}$. Let θ_m, $m = 0, \ldots, M$ denote iterates from the SGLD algorithm; then they are updated by the following algorithm

$$\theta_{m+1} = \theta_m - h\nabla\hat{f}(\theta_m) + \sqrt{2h}\zeta_m, \qquad \zeta_m \sim N(0,1); \tag{2.4.1}$$

Welling and Teh (2011) suggested the following estimate of f,

$$\hat{f}(\theta) = f_0(\theta) + \frac{N}{n}\sum_{i\in S} f_i(\theta), \tag{2.4.2}$$

where S is a random sample from $\{1, \ldots, N\}$ of size n. This algorithm therefore has two error contributions: bias due to the discretisation by h; and the estimate of $\hat{f}$. As a result, the stationary distribution is just an approximation to the desired target π. The SGLD algorithm is also highly related to the popular scalable optimisation aglorithm stochastic gradient descent (SGD, Robbins and Monro, 1951); whose update is given by $\theta_{m+1} = -h\nabla\hat{f}(\theta_m)$.

A natural extension to SGLD would be to consider SGMCMC algorithms based on higher order dynamics such as Hamiltonian dynamics introduced earlier. A difficulty is that replacing ∇f with $\nabla\hat{f}$ in more popular HMC dynamics based on ordinary differential equations (see e.g. Neal, 2010) leads to the resulting underlying process having a stationary distribution that is a poor approximation to π. Chen et al. (2014) get around this by using the alternative SDE dynamics introduced in (2.3.8). This sampler enables the more efficient Hamiltonian dynamics to be used to sample approximately from π in the big data setting. The downside is that in order for the more efficient dynamics to be taken advantage of, either an estimate of the Fisher

information is required, or the stepsize needs to be set small; Ding et al. (2014) aim to get around this issue by adaptively estimating the Fisher information within the dynamics.

Designing SGMCMC algorithms requires two main ingredients: identifying an underlying Itô process which has π uniquely as its stationary distribution; replacing f with $\hat{f}$ in a way that the algorithm still provides a reasonable approximation to π. To make this process more principled, Ma et al. (2015) provide a general Itô process which targets π and shows it is complete (i.e. any Itô process that has π as its stationary distribution can be written in this form). The reason that replacing f with $\hat{f}$ can make the approximation poor is that $\hat{f}$ adds additional, unwanted noise to the process. In light of this, Ma et al. (2015) also develop an approximate correction term to counteract this effect.

Theoretical Results

Most of the theoretical results for SGMCMC samplers have focussed on the most popular algorithm, SGLD. Much of this builds on previous work for Euler–Maruyama schemes. Sato and Nakagawa (2014) investigate the error of the SGLD algorithm compared with the underlying Langevin diffusion. Teh et al. (2016) analysed SGLD with a decreasing stepsize scheme in a similar way to the analysis of Lamberton and Pagès (2002) on the Euler–Maruyama method. Teh et al. (2016) showed, similar to the Euler–Maruyama method, the optimal decreasing stepsize scheme is to set h_m to decrease at $O(m^{-1/3})$. When h_m decreases optimally, the bias and RMSE of $\hat{\psi}$ versus $\bar{\psi}$ is $O(M^{-1/3})$. Similarly, Vollmer et al. (2016) build on the work of Mattingly et al.

(2010), to investigate the non-asymptotic error of $\hat{\psi}$ when SGLD is implemented with a fixed stepsize. As in Teh et al. (2016), they find SGLD recovers the same bias and MSE of the Euler–Maruyama scheme, i.e. a bias of $O(h + 1/(Mh))$ and MSE of $O(h^2 + 1/(Mh))$. Vollmer et al. (2016) also builds on the work of Talay and Tubaro (1990) to find that the asymptotic bias of SGLD is $O(h)$. Chen et al. (2015) extend the work of Vollmer et al. (2016) to more complex algorithms such as SGHMC. Finally, Dalalyan and Karagulyan (2017) build on the work of Durmus and Moulines (2017b) in order to develop W_2 bounds for the distance of the distribution defined by SGLD to π in the strongly log-concave setting.

There has also been considerable interest in SGMCMC algorithms in the optimisation literature. Suppose we have access to i.i.d data $\mathbf{x} = (x_1, \ldots, x_N)^T$, where each data point is a random element from the unknown distribution P; and interest is in some function $h(\theta, x)$, where $\theta \in \mathbb{R}^d$. Raginsky et al. (2017) investigate using SGLD to approximate

$$H^* = \min_{\theta \in \mathbb{R}^d} H(\theta) = \min_{\theta \in \mathbb{R}^d} \mathbb{E}_P[h(\theta, X)].$$

Define $H_{\mathbf{x}}(\theta) = \frac{1}{N} \sum_{i=1}^{N} h(\theta, x_i)$. To approximate H^* Raginsky et al. (2017) implement an SGLD algorithm that targets the distribution $\pi_{\mathbf{x}}$, with density $p_{\mathbf{x}}(\theta) \propto e^{-\beta H_{\mathbf{x}}(\theta)}$, with update as follows

$$\theta_{m+1} = \theta_m + h\nabla \hat{H}_{\mathbf{x}}(\theta) + \underbrace{\sqrt{2\beta^{-1}}\zeta_m}_{\text{injected noise}}, \qquad \zeta_m \sim N(0, 1), \tag{2.4.3}$$

where $\nabla \hat{H}_{\mathbf{x}} := \frac{1}{|S|} \sum_{i \in S} \nabla h(\theta, x_i)$, with $S \subset \{1, \ldots, N\}$, is the standard stochastic estimate of $\nabla H_{\mathbf{x}}$. Here β is a user-specified constant known as the temperature. The idea is that β is set large enough for $e^{-\beta H_{\mathbf{x}}(\theta)}$ to concentrate around the minima of

$H_{\mathbf{x}}$; but that β is small enough for the injected noise term to escape local modes, and so that the algorithm does not overfit to $H_{\mathbf{x}}$ rather than approximating the desired H^*. Central to the analysis by Raginsky et al. (2017) are the assumptions that $h(\cdot, x)$ is M-smooth and m-dissipative for all $x \in \mathcal{X}$. The dissipative property is a common assumption in the study of dynamical systems, it states that there exists constants $m > 0$, $b \geq 0$ such that

$$\langle \theta, \nabla h(\theta, x) \rangle \geq m \|\theta\|^2 - b, \qquad \theta \in \mathbb{R}^d.$$

The dissipative assumption means that within a ball of radius $\sqrt{b/m}$, the function can be arbitrarily complicated, with multiple stationary points. As we move outside this ball though, the gradient points back towards the ball with increasing magnitude. Under these assumptions, Raginsky et al. (2017) investigates the *population risk* of θ_m. More formally, they non-asymptotically bound $|\mathbb{E}[H(\theta_m)] - H^*|$, where the expectation is with respect to the data $\mathbf{x}$ and any additional randomness used by the SGLD algorithm to generate θ_m. Part of the proof relies on bounding the 2-Wasserstein distance between SGLD and the underlying Langevin diffusion. Similarly, Xu et al. (2018) investigate the non-asymptotic bounds on the *empirical risk* i.e. $|\mathbb{E}[H_{\mathbf{x}}(\theta)] - \min_{\theta \in \mathbb{R}^d} H_{\mathbf{x}}(\theta)|$ under this setting. These are important results for optimisation. Most results for stochastic optimisation methods in the non-convex setting before now have only been able to guarantee local minimisation. Both works also provide interesting theoretical tools for extending the analysis of the convergence of SGLD to the target distribution in terms of 2-Wasserstein distance to the non-convex setting.

Theoretical analysis of SGLD, especially the work of Dalalyan and Karagulyan (2017), makes it clear that the error in SGLD is dominated by the error in the estimate $\hat{f}$. In light of this, recent work has considered variance reduction for $\hat{f}$ in order to improve the convergence of SGLD. Dubey et al. (2016) adapt variance reduction methods for SGD to SGLD, and show that this improves the MSE of the algorithm using the work of Chen et al. (2015). Nagapetyan et al. (2017), and independently Baker et al. (2018) contained in Chapter 3 of this thesis, consider improvements to the computational cost by implementing SGLD with control variates (both in the strongly log-concave setting). There are a number of empirical results suggesting that, despite the per iteration computational savings of SGLD, the cost of implementing SGLD in order to reach an arbitrary level of accuracy for the given metric (for example W_2) is still $O(N)$. Nagapetyan et al. (2017) analyse SGLD with control variates (SGLD-CV) in order to reach a desired level of accuracy in terms of the MSE of the average of θ_M over K independent samples from SGLD. They show that the resulting implementation has $O(\log(N))$ computational cost. In comparison Baker et al. (2018) extend the results of Dalalyan and Karagulyan (2017) to derive a W_2 distance bound for SGLD-CV, and find there exists an implementation with arbitrary W_2 accuracy that has $O(1)$ computational cost. There is currently no result proving that the computational cost of SGLD is $O(N)$. Such a result would require showing that there is no implementation with sublinear computational cost in N such that SGLD reaches arbitrary accuracy in the desired distance measure. This requires upper bounds on the distance measure, which have not yet been derived for SGLD. Chatterji et al. (2018), again extend the work of Dalalyan and Karagulyan (2017) to derive computational

cost results and bounds in W_2 for the variance reduction methods of Dubey et al. (2016).

Scalable Samplers Using Piecewise Deterministic Processes

More recently, a number of big data MCMC samplers have been introduced based on a class of Markov processes known as piecewise deterministic processes (PDP; Davis, 1984). PDPs cannot be written in terms of Itô diffusions; rather they move in a deterministic direction until an event occurs determined by an inhomogeneous Poisson process. When the event occurs the direction of motion is switched. Similarly to the Hamiltonian dynamics of (2.3.8), PDPs augment the state space with a velocity parameter $\nu \in \mathcal{V} \subset \mathbb{R}^d$, which determines the current direction of motion. Let (θ_t, ν_t) be a PDP, then the process can be constructed so that θ_t has marginal stationary solution π, for general π (Davis, 1984; Fearnhead et al., 2018). To do this requires switching direction using a Poisson process with inhomogeneous rate $\max(0, \nu_t \cdot \nabla f(\theta_t))$. Remarkably, provided the Poisson process can be simulated from, the PDP can be simulated exactly. Also, ∇f can be replaced by $\nabla \hat{f}$ and π remains the stationary solution (Fearnhead et al., 2018).

These results have been used to develop a number of scalable samplers (Bouchard-Côté et al., 2018; Bierkens et al., 2018a; Pollock et al., 2016), which target π exactly. The most popular of these methods are the bouncy particle sampler (BPS) (Bouchard-Côté et al., 2018) and the zig-zag sampler (ZZ) (Bierkens et al., 2018a). The main difference between these methods is the way the direction ν is chosen: at each event time, the direction of BPS is chosen by reflecting the velocity on the hyperplane

tangential to $\nabla f(\theta_t)$; while the direction of ZZ is ± 1 in each dimensional component. As well as the advantage of targeting the posterior exactly, geometric ergodicity results have been derived for both BPS and ZZ (Deligiannidis et al., 2018; Bierkens et al., 2018b). It has been suggested to use control variates in a similar way to Chapter 3, in order to reduce the variance of the gradient estimate $\nabla \hat{f}$ when using PDP samplers (see e.g. Fearnhead et al., 2018). Specifically, it has been shown by Bierkens et al. (2018a) that this can improve the efficiency of the ZZ sampler.

The main difficulty in implementing these samplers is in simulating from the inhomogeneous Poisson process with rate $\max(0, \nu_t \cdot \nabla f(\theta_t))$. In general, the suggested procedure to simulate from this is known as thinning (see e.g. Lewis and Shedler); but this requires a local upper bound of $\partial_j f(\theta)$, for $j = 1, \ldots, d$, to be calculated. This upper bound is problem specific, and a significant overhead for these samplers.

2.4.2 Comparison to Divide-and-Conquer MCMC

In this section we present the first contribution of this thesis. The section compares the two most popular SGMCMC algorithms – SGLD and SGHMC – to some popular alternative methods known as divide-and-conquer methods. Divide-and-conquer methods aim to also improve the scalability of MCMC. They achieve this by splitting the dataset into subsets, and running separate MCMC chains in parallel. The main challenge is then in combining the information from each of these samples to produce an MCMC chain that targets an approximation to π. This early work demonstrated that SGMCMC methods seem more robust than the divide-and-conquer counterparts, and formed the motivation to develop SGMCMC methods further for the remainder of

the thesis. However it is worth mentioning that alternative divide-and-conquer methods have since become more popular (Minsker et al., 2014; Xu et al., 2014; Srivastava et al., 2015; Minsker et al., 2017; Li et al., 2017; Nemeth and Sherlock, 2018).

Divide-and-conquer methods

We assume the Bayesian setup of Section 2.2, (2.2.3). The divide-and-conquer methods divide the data into disjoint subsets $B_s \subset \{1, \ldots, N\}$ for $s = 1, \ldots, S$. Defining $h_{B_s}(\theta) := p_0^{1/S} \prod_{B_s} p_i(\theta)$, referred to as the (unnormalised) subposterior, the unnormalised posterior is given by

$$h(\theta) = \prod_{s=1}^{S} h_{B_s}(\theta) \tag{2.4.4}$$

This leads to the idea that MCMC can be run to target each subposterior in parallel, then the chains can be combined to get a chain that approximately targets π. We let $\theta_{s1}, \ldots, \theta_{sM}$, $s = 1, \ldots, S$ denote the MCMC sample from subposterior s. In reality, this recombination step is challenging. We compare SGLD and SGHMC to three divide-and-conquer methods: Consensus Monte Carlo (Scott et al., 2016), kernel density estimation Monte Carlo (KDEMC) (Neiswanger et al., 2014) and the Weierstrass sampler (Wang and Dunson, 2013).

Consensus Monte Carlo

The simplest way to recombine the samples from the subposteriors is to approximate each subposterior as a Gaussian distribution. The samples can be used to estimate the mean and variance of each of the subposteriors. Then, conditionally on these estimates

we can analytically calculate a Gaussian approximation to the full posterior. This idea was first proposed by Neiswanger et al. (2014). The motivation is that as N gets large the Bernstein-von Mises theorem states that the posterior will be approximately Gaussian (Le Cam, 2012).

The consensus Monte Carlo algorithm of Scott et al. (2016) aims to improve on this. It works by approximating the full posterior as a weighted average of the subposterior samples. The idea behind consensus Monte Carlo is that, if the subposteriors were Gaussian then this method of combining samples would give us draws from the true posterior; but if the subposteriors are not Gaussian, Scott et al. (2016) argue that the weighted averaging procedure is more likely to inherit properties of the subposterior samples themselves, rather than forcing the approximation to be Gaussian.

Scott et al. (2016) propose estimating the full MCMC chain, call this $\hat{\theta}_i$, as a weighted average of the subposterior samples

$$\hat{\theta}_i = \left(\sum_{s=1}^{S} W_s \right)^{-1} \sum_{s=1}^{S} W_s \theta_{si}, \tag{2.4.5}$$

where $W_s \in \mathbb{R}^{d \times d}$ is a weight matrix for subposterior s. Scott et al. (2016) suggest letting $W_s = \hat{\Sigma}_s^{-1}$, where $\hat{\Sigma}_s$ is the sample covariance matrix for θ_s.

KDEMC

Neiswanger et al. (2014) suggest applying kernel density estimation to each subposterior sample θ_s, in order to estimate the true density of that subposterior. Denote this estimate $\hat{p}_{B_s}(\theta)$. Then by (2.4.4) we can approximate the full posterior by $\hat{p}(\theta) = \prod_{s=1}^{S} \hat{p}_{B_s}(\theta)$.

If Gaussian kernels are used in the approximation, then $\hat{p}(\theta)$ becomes a product of Gaussian mixtures. This product can be expanded to give another Gaussian mixture with $O(SM)$ components, where M is the number of iterations of the MCMC chain that are stored, and S is the number of subposteriors. Neiswanger et al. (2014) suggest sampling from this Gaussian mixture using MCMC. We refer to this algorithm as KDEMC. The number of mixture components increases dramatically with the number of subsets and subposterior samples. This means KDEMC can be computationally expensive and inefficient, but the algorithm should target more complex posterior geometries. Neiswanger et al. (2014) also suggest a similar method based on *semiparametric* density estimation; but we find this method performs similarly to consensus Monte Carlo so omit it in the comparisons.

Weierstrass

The Weierstrass method (Wang and Dunson, 2013) is similar to KDEMC, but uses a Weierstrass transform to approximate the subposterior densities rather than a kernel density estimate. Using the Weierstrass approximation rather than a kernel density is associated with a number of better properties, including an improvement when subposteriors do not overlap, and better scalings with dimensionality. To produce draws from the KDE approximation, Neiswanger et al. (2014) suggest using Metropolis-within-Gibbs procedure. To avoid the inefficiencies of this sampler, Wang and Dunson (2013) develop an approximate, more computationally efficient scheme to produce draws from the Weierstrass approximation based on rejection sampling.

Experiments

As far as we are aware, there has been limited comparison across stochastic gradient and divide and conquer methods, and we aim to bridge this gap in the following experiments. We use simple examples that focus on important scenarios, and hope to build intuition for where methods should be used. The particular scenarios we focus on are: (i) heavy tailed posterior, (ii) multi-modal posterior, (iii) posteriors with complex geometry, and (iv) the impact of parameter dimension.

We compare each method's performance by measuring the KL divergence between the approximate sample and a HMC sample, taken to be the truth, using the R package *FNN* (Li et al., 2013). The HMC sample is simulated using the NUTS sampler (Hoffman and Gelman, 2014) implemented in the probabilistic programming language STAN (Carpenter et al., 2017). The KL divergence is measured over 10 different runs of the algorithm (using the same dataset) and plotted as boxplots. Contour plots for one simulation are also provided to help develop the reader's intuition. The only method which does not require tuning parameters is the Consensus method, which is an advantage as this can take a lot of time. For fairness, the other methods are tuned by minimizing the KL divergence measure which we use to make the boxplots. The Weierstrass algorithm is implemented using the associated R package (Wang and Dunson, 2013).

It is quite difficult to implement the algorithms in a way that makes computational cost similar. For example, KDEMC has computational cost component $O(N/S)$ for each iteration of a parallel chain; but then $O(M)$ for each iteration of the MCMC dur-

ing the recombination step, which to get a good approximation will be non-negligible. As a result, we opt for simulating each algorithm for a fixed number of iterations, since often problems with the methods are quite obvious even despite the difference in computational cost. The computational cost of the parallel methods is $O(N/S)$ for each iteration of parallel MCMC, then the consensus Monte Carlo recombination step has a one-time $O(M)$ cost; KDEMC and Weierstrass has $O(M)$ cost to produce one approximate draw from the subposterior chains (KDEMC produces each draw sequentially, as it uses MCMC; while the Weierstrass method can produce multiple draws in parallel). SGLD has computational cost $O(n)$ for each iteration; while SGHMC has cost $O(Ln)$ for each iteration, where L is a tuning constant known as the trajectory which we set to be 3.

Heavy tailed posterior

To compare the methods on a heavy tailed target, we infer the location θ from data $\mathbf{x}$ simulated from a bivariate t-distribution with known scale Σ and degrees of freedom ν. The density of $\mathbf{x}$ is given by

$$p(\mathbf{x}|\theta) \propto \left[1 + \frac{1}{\nu}(\mathbf{x}-\theta)^T\Sigma^{-1}(\mathbf{x}-\theta)\right]^{-(\nu+2)/2},$$

where we assume an uninformative uniform prior on θ. In order to test the algorithms we use a relatively small dataset size of 800. The number of subposteriors used in the divide and conquer methods is 20. We use a minibatch size of 50 for the stochastic gradient MCMC methods.

Figure 2.4.1 gives an illustrative comparison of the methods. The results show that

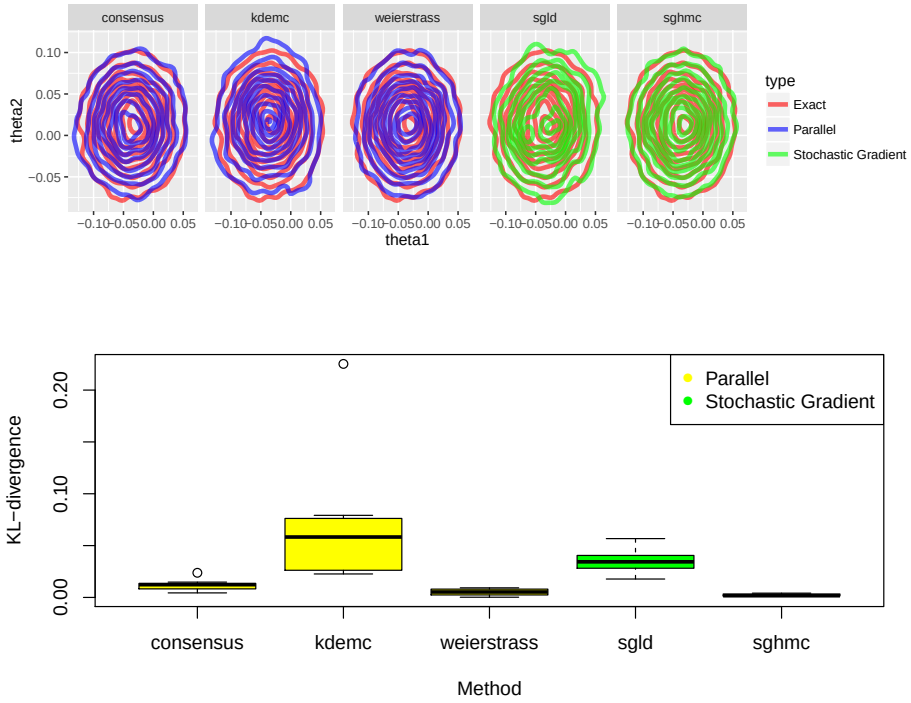


Figure 2.4.1: Comparison of method performance for multivariate-t distribution. Contour plots show empirical densities. Box plots show KL-divergence from the truth.

all methods are equipped to explore this posterior, obtained as a product of heavy tailed densities. The KDEMC and SGLD algorithms have the poorest approximation to the posterior. It has been shown in Teh et al. (2016) that the convergence rate of SGLD is $O(T^{-\frac{1}{3}})$, and therefore slower than the standard Monte Carlo rate of $O(T^{-\frac{1}{2}})$. In this scenario SGHMC performs the best in terms of minimizing KL divergence (though its computational cost is 3 times higher than SGLD), closely followed by the consensus Monte Carlo algorithm and the Weierstrass sampler. The Weierstrass sampler does a good job of improving the convergence speed of KDEMC. There is an additional advantage in using consensus Monte Carlo as it does not require tuning, and its computational cost for this problem was low, so it is arguably the best choice

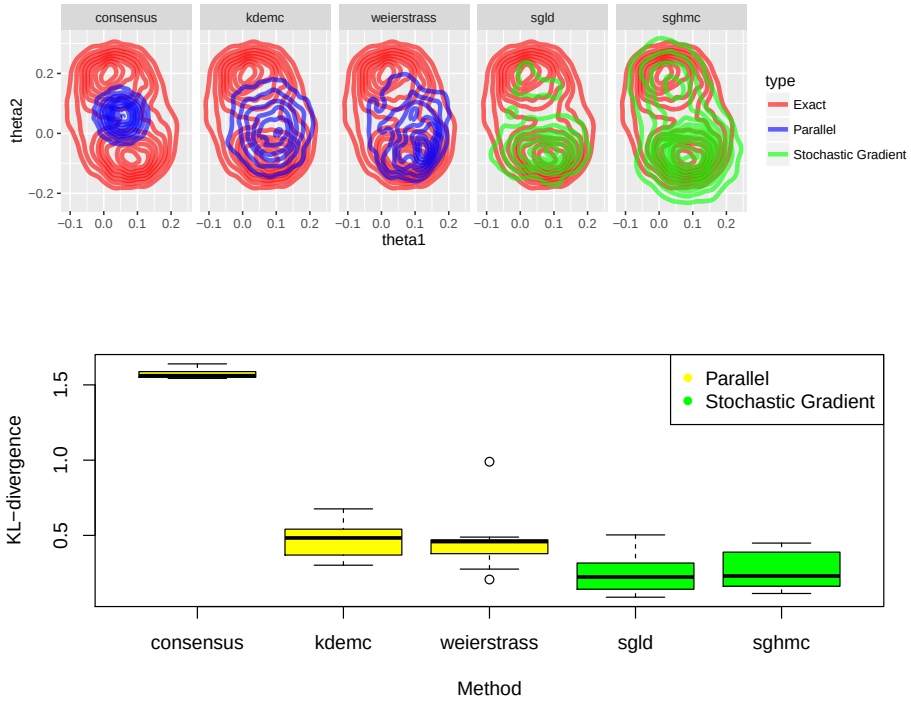


Figure 2.4.2: Comparison of method performance for Gaussian mixture. Contour plots show empirical densities. Box plots show KL-divergence from the truth.

for this problem.

Multi-modal posterior

We compare the methods on a multi-modal target where we infer the locations θ_1, θ_2 from data $\mathbf{x}$ simulated from a bimodal, bivariate Gaussian mixture. We assume the mixture has known common scale Σ, and equal allocation probabilities, leading to the following density of $\mathbf{x}$

$$p(\mathbf{x}|\theta_1, \theta_2) \propto \mathcal{N}(\mathbf{x}|\theta_1, \Sigma) + \mathcal{N}(\mathbf{x}|\theta_2, \Sigma),$$

where $\mathcal{N}(\mathbf{x}|\theta, \Sigma)$ denotes a Gaussian density with mean θ and variance Σ. We assume the priors on θ_i are independent Gaussians with mean 0 and a large variance. We use

a dataset size of 10000. The number of subposteriors used in the divide and conquer methods is 20. We use a minibatch size of 50 for the stochastic gradient MCMC methods.

Results given in Figure 2.4.2 show that the consensus algorithm performs poorly in this setting. The simple weighted average scheme (2.4.5) leads to a unimodal posterior approximation which does not account for the bimodality, suggesting that the posterior mass lies between the subposterior modes. KDEMC offers an improvement over the Consensus algorithm with improved posterior coverage, but still does not capture the bimodality. From investigating the subposteriors it appears that this is a result of the kernel bandwidth being smaller than the width of the posterior, leading to density estimates for each subposterior which tail off rapidly outside of the region where subposterior samples lie. Therefore, the approximation is unimodal in the location where most of the subposteriors overlap. The Weierstrass method seems to encounter similar issues to KDEMC, and its performance is not much better.

The stochastic gradient methods perform better than the divide and conquer approaches, and are able to explore both modes. Note that while the methods are able to explore overlapping modes, if the modes are more separated the stochastic gradient methods will also struggle with exploration. Given a good starting point, SGHMC performs particularly well. When the starting point is further from the posterior mass, the algorithm does not appear to explore both modes. Overall SGLD performs the best at minimising the KL-divergence of this problem, especially considering its lower computational cost compared with SGHMC.

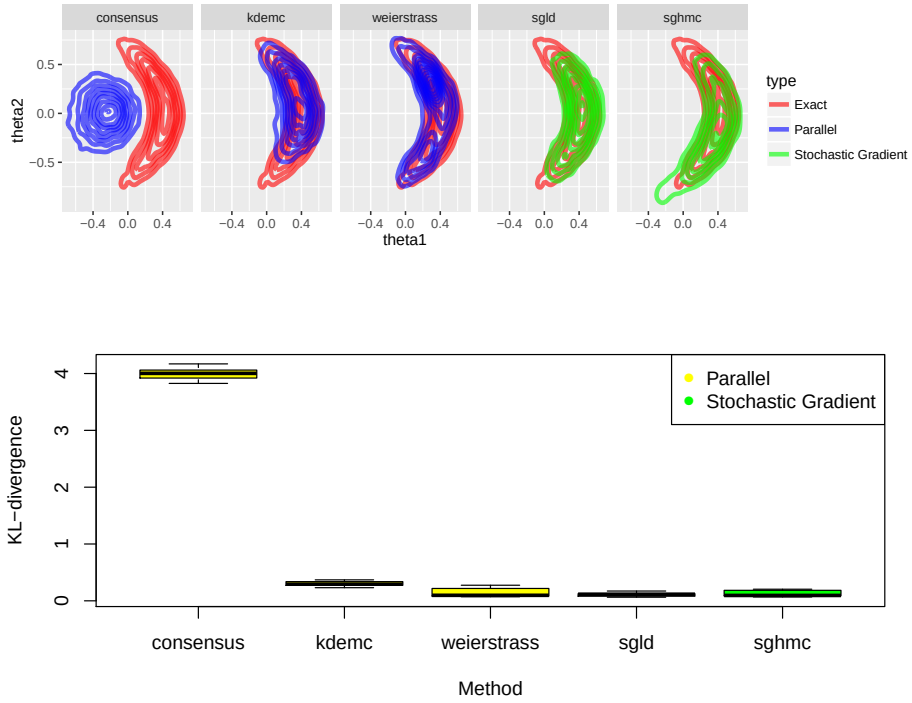


Figure 2.4.3: Comparison of method performance for warped Gaussian. Contour plots show empirical densities. Box plots show KL-divergence from the truth.

Complex geometry: warped Gaussian

We consider a target with complex geometry known as the warped Gaussian. In this case, locations θ_1 and θ_2 are inferred from data x with density

$$p(x|\theta_1, \theta_2) = \mathcal{N}(x|\theta_1 + \theta_2^2, \sigma_x),$$

where σ_x is a known scale parameter. We assume the prior for each θ_i are independent with density $p(\theta_i) = \mathcal{N}(\theta_i|0, \sigma_\theta)$, where σ_θ is some known scale parameter. We use a small dataset size of 800, and the number of subposteriors used in the divide and conquer methods is 20. We use a minibatch size of 50 for the stochastic gradient MCMC methods.

The results given in Figure 2.4.3 show that again the consensus algorithm struggles to approximate the full posterior. The consensus approach uses an average of the subposterior samples, re-scaled by their covariance. One way of understanding why the consensus performs poorly in this example is to consider the situation where there are only two subposteriors, each with approximately the correct warped Gaussian shape and location as the full posterior. Averaging samples from each subposterior would lead to some samples located in the the lower tail of subposterior one being averaged with samples from the upper tail of subposterior two, thus producing an approximation to the full posterior which lies in the centre, as shown in Figure 2.4.3. The KDEMC works reasonably well on this example, but underestimates the tails for the same reason as discussed for the mixture example (Section 2.4.2). The Weierstrass shows some improvement over KDEMC, though does not perform as well as the stochastic gradient methods.

Finally, the stochastic gradient methods perform better than the divide and conquer algorithms and once again SGHMC is more sensitive to the starting point than SGLD.

Dimensionality: multivariate Gaussian

The examples considered so far have been in low dimensional parameter spaces. In this section we explore how these big data MCMC algorithms scale with increasing the dimension of the posterior. We consider the posterior for θ given $\mathbf{x}$, where $\mathbf{x}$ follows a multivariate Gaussian with known scale Σ. We assume an uninformative uniform prior for θ. We use a dataset size of 800; the number of subposteriors used

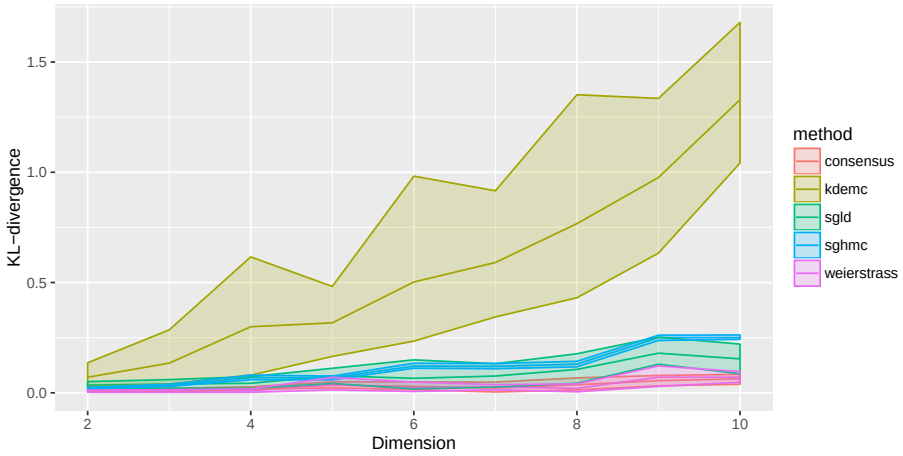


Figure 2.4.4: Comparison of method performance for Gaussian. Plot of KL-divergence against dimension for each method.

in the divide and conquer methods is 20; and we use a minibatch size of 50 for the stochastic gradient MCMC methods.

Figure 2.4.4 gives the KL divergence between the full posterior and the approximate posterior resulting from each of the considered algorithms. Kernel density methods are well known to scale poorly with dimension and this is shown here. The Consensus algorithm performs particularly well. This is unsurprising as the consensus algorithm is exact when each subposterior is Normally distributed. The Weierstrass method scales much better with dimensionality than its KDEMC counterpart, this is due to its sequential rejection sampling procedure, which ensures that error accumulates linearly in dimensionality, as opposed to exponentially in dimensionality as is the case for KDEMC. Both minibatch methods work well, but the trend in Figure 2.4.4 implies that these algorithms may lose some performance in significantly larger posterior spaces.

Discussion

When considering unimodal posteriors which do not exhibit complex geometry, the consensus algorithm performs best; as the algorithm does not require any tuning and scales well in high-dimensional parameter spaces. KDEMC is a natural extension to the consensus algorithm, which merges the subposterior densities rather than the subposterior samples. We found through experimentation that, as with the consensus algorithm, the KDEMC approach tends to underestimate the tails of the full posterior density, which is particularly an issue when the subposterior densities do not overlap. The KDEMC algorithm also scales very poorly with dimension. The Weierstrass sampler, which extends ideas of the KDEMC algorithm, fixes many of these issues. The algorithm scales well with dimensionality; copes better with posteriors that do not overlap and converges faster than KDEMC. However the algorithm still struggles with multimodality, and is not quite up to the standard of the SGMCMC methods when it comes to more complex geometry. The algorithm requires tuning of an acceptance step, but the results are not too sensitive to this choice.

Stochastic gradient methods were found to be robust to the geometry of the posterior, and (in relative terms) were more effective for multimodal posteriors. A major disadvantage of these algorithms is how sensitive they are to the choice of stepsize, though some work has been done to improve this (Giles et al., 2016; Gorham et al., 2016). We found in general the extra computational cost of SGHMC did not lead to vastly improved performance over SGLD. This could be due to sensitivity to the Fisher information estimate, or to the choice of starting point.

It was hard to compare computational cost directly between the two methods, but each algorithm was run for the same number of MCMC iterations. For most of the comparisons, the dataset size was chosen to be about 800, so quite small, designed to test the methods. In these cases the computational cost between SGLD and the divide and conquer methods is similar since the divide and conquer methods have a per iteration cost of 40 (as the number of batches is 20), and the SGLD algorithm has a per iteration cost of 50 since that was its minibatch size. So the comparisons certainly demonstrate the slow convergence rate of SGLD for simple examples. But SGLD demonstrated strength in the more complex geometry of the warped Gaussian. In the Gaussian mixture example, SGLD has the lowest computational cost by far, but still performs the best. It is worth noting that the combining step adds to the computational cost, and for KDEMC this is a particularly slow process.

On the other hand, the trajectory L for SGHMC was chosen to be 3, so that the momentum parameter was not refreshed at every step. This means the per iteration computational cost was about 3 times higher than the other methods. For simple examples, SGHMC did not warrant this extra tuning and computational cost, as it did not perform much better than consensus Monte Carlo. For more complex examples, the method again did not warrant the extra cost over SGLD.

Chapter 3

Control Variates for Stochastic Gradient MCMC

3.1 Introduction

Markov chain Monte Carlo (MCMC), one of the most popular methods for Bayesian inference, scales poorly with dataset size. This is because standard methods require the whole dataset to be evaluated at each iteration. Stochastic gradient MCMC (SGMCMC) is a class of MCMC algorithms that aim to overcome this issue. The algorithms have recently gained popularity in the machine learning literature. These methods use efficient MCMC proposals based on discretised dynamics that use gradients of the log-posterior. They reduce the computational cost by replacing the gradients with an unbiased estimate which uses only a subset of the data, referred to as a minibatch. They also bypass the acceptance step by making small discretisation steps (Welling and Teh, 2011; Chen et al., 2014; Ding et al., 2014; Dubey et al., 2016).

These new algorithms have been successfully applied to a range of state of the art machine learning problems (e.g., Patterson and Teh, 2013; Li et al., 2016). There is now software available to implement these methods (Tran et al., 2016; Baker et al., 2018). In particular, Baker et al. (2018) implements the control variate methodology we discuss in this article.

This paper investigates stochastic gradient Langevin dynamics (SGLD), a popular SGMCMC algorithm that discretises the Langevin diffusion. There are a number of results suggesting that while SGLD has a lower per-iteration computational cost compared with MCMC, its overall computational cost is proportional to the dataset size (Welling and Teh, 2011; Nagapetyan et al., 2017). This motivates improving the computational cost of SGLD, which can be done by using control variates (Ripley, 2009). Control variates can be applied to reduce the Monte Carlo variance of the gradient estimate in stochastic gradient MCMC algorithms. We refer to SGLD using this new control variate gradient estimate as SGLD-CV.

We analyse the algorithm using the Wasserstein distance between the distribution defined by SGLD-CV and the true posterior distribution, by adapting recent results by Dalalyan and Karagulyan (2017). Central to this analysis are the assumptions that the log-posterior is m-strongly-concave and M-smooth (defined formally in Assumption 3.3.1). This is quite a strong assumption, but has become common for the analysis of these methods (see e.g., Durmus and Moulines, 2017a; Dalalyan and Karagulyan, 2017; Nagapetyan et al., 2017) since it ensures that errors do not accumulate from one iteration to the next. We provide an empirical comparison on a wide variety of models that do not necessarily satisfy these conditions in order to fully explore the results.

If the concave and smoothness conditions do not hold, then provided the posterior contracts with the number of observations, there should still be some benefit from the variance reduction; but we leave a full analysis in the nonconvex case for future work.

We get bounds on the Wasserstein distance between the target distribution and the distribution we sample from at a given step of SGLD-CV. These bounds are in terms of the tuning constants chosen when implementing SGLD-CV. Under assumptions on how the posterior changes as we get more data, we are able to show that, after an initialisation step, the cost of running SGLD-CV to a required level of accuracy does not grow with the number of data points. The initialistion step involves finding a centring value $\hat{\theta}$ using optimisation and evaluating the gradient of the log posterior at this value. Both these are $O(N)$ calculations, where N is the dataset size, but we show in Section 3.3.4 these each require just a single pass through the dataset. We also suggest starting the algorithm from the centring value, essentially replacing the burn-in step of SGLD with the optimisation step. The experiments show this optimisation step is often more efficient than the burn-in step of SGLD. Our results in Section 3.3.3 quantify the impact on performance of obtaining a poor centring value.

The use of control variates has also been shown to be important for other Monte Carlo algorithms for simulating from a posterior with a cost that is sub-linear in the number of data points (Bardenet et al., 2017; Bierkens et al., 2018a; Pollock et al., 2016; Nagapetyan et al., 2017). For previous work that suggests using control variates within SGLD, see Dubey et al. (2016); Chen et al. (2017). These latter papers, whilst showing benefits of using control variates, do not show that the resulting algorithm

can have sub-linear cost in the number of data points. A recent paper, Nagapetyan et al. (2017), does investigate how SGLD-CV performs in the data limit under similar log-concavity assumptions on the posterior distribution. They have results that are qualitatively similar to ours, including the sub-linear computational cost of SGLD-CV. Though they measure accuracy of the algorithm through the mean squared error of Monte Carlo averages rather than through the Wasserstein distance.

Not only can control variates be used to speed up stochastic gradient MCMC by enabling smaller minibatches to be used; we show also that they can be used to improve the inferences made from the MCMC output. In particular, we can use post-processing control variates (Mira et al., 2013; Papamarkou et al., 2014; Friel et al., 2016) to produce MCMC samples with a reduced variance. The post-processing methods rely on the MCMC output as well as gradient information. Since stochastic gradient MCMC methods already compute estimates of the gradient, we explore replacing the true gradient in the post-processing step with these free estimates. We also show theoretically how this affects the variance reduction factor; and empirically demonstrate the variance reduction that can be achieved from using these post-processing methods.

3.2 Stochastic Gradient MCMC

Throughout this paper we aim to make inference on a vector of parameters $\theta \in \mathbb{R}^d$, with data $\mathbf{x} = \{x_i\}_{i=1}^N$. We denote the probability density of x_i as $p(x_i|\theta)$ and assign a prior density $p(\theta)$. The resulting posterior density is then $p(\theta|\mathbf{x}) \propto p(\theta)\prod_{i=1}^N p(x_i|\theta)$, which defines the posterior distribution π. For brevity we write $f_i(\theta) = -\log p(x_i|\theta)$

for $i = 1, \dots N$, $f_0(\theta) = -\log p(\theta)$ and $f(\theta) = -\log p(\theta|\mathbf{x})$.

Many MCMC algorithms are based upon discrete-time approximations to continuous-time dynamics, such as the Langevin diffusion, that are known to have the posterior as their invariant distribution. The approximate discrete-time dynamics are then used as a proposal distribution within a Metropolis-Hastings algorithm. The accept-reject step within such an algorithm corrects for any errors in the discrete-time dynamics. Examples of such an approach include the Metropolis-adjusted Langevin algorithm (MALA; see e.g., Roberts and Tweedie, 1996) and Hamiltonian Monte Carlo (HMC; see e.g., Neal, 2010).

3.2.1 Stochastic Gradient Langevin Dynamics

SGLD, first introduced by Welling and Teh (2011), is a minibatch version of the unadjusted Langevin algorithm (Roberts and Tweedie, 1996). At each iteration it creates an approximation of the true gradient of the log-posterior by using a small sample of data.

The SGLD algorithm is based upon the discretisation of a stochastic differential equation known as the Langevin diffusion. A Langevin diffusion for a parameter vector θ with posterior $p(\theta|\mathbf{x}) \propto \exp(-f(\theta))$ is given by

$$\theta_t = \theta_0 - \int_0^t \nabla f(\theta_s) ds + \sqrt{2} dB_t, \tag{3.2.1}$$

where B_t is a d-dimensional Wiener process. The stationary distribution of this diffusion is π. This means that it will target the posterior exactly, but in practice we need to discretise the dynamics to simulate from it, which introduces error. A bottleneck

for this simulation is that calculating $\nabla f(\theta)$ is an $O(N)$ operation. To get around this, Welling and Teh (2011) replace the log-posterior gradient with the following unbiased estimate

$$\nabla \hat{f}(\theta) := \nabla f_0(\theta) + \frac{N}{n} \sum_{i \in S_k} \nabla f_i(\theta) \tag{3.2.2}$$

for some subsample S_k of $\{1, \ldots, N\}$, with $|S_k| = n$ representing the minibatch size. A single update of SGLD is then

$$\theta_{k+1} = \theta_k - \frac{h_k}{2} \nabla \hat{f}(\theta_k) + \zeta_k, \tag{3.2.3}$$

where $\zeta_k \sim N(0, h_k)$.

MALA uses a Metropolis-Hastings accept-reject step to correct for the discretisation of the Langevin process. Welling and Teh (2011) bypass this acceptance step, as it requires calculating $p(\theta|\mathbf{x})$ using the full dataset, and instead use an adaptive rather than fixed stepsize, where $h_k \to 0$ as $k \to \infty$. The motivation is that the noise in the gradient estimate disappears faster than the process noise, so eventually, the algorithm will sample the posterior approximately. In practice, we found the algorithm does not mix well when the stepsize is decreased to zero, so in general a fixed small stepsize h is used in practice, as suggested by Vollmer et al. (2016).

3.3 Control Variates for SGLD Efficiency

The SGLD algorithm has a reduced per iteration computational cost compared to traditional MCMC algorithms. However, there have been a number of results suggesting that the overall computational cost of SGLD is still $O(N)$ (Welling and Teh, 2011;

Nagapetyan et al., 2017). What we mean by this is that in order for the algorithm to reach a desired distance from the true posterior, the minibatch size n, and the total number of iterations K, need to be set so that the computational cost Kn is $O(N)$. The main reason for this result is that in order to control the variance in the gradient estimate, $\nabla \hat{f}(\theta)$, we need n to increase linearly with N. Therefore, we would assume that reducing this variance would lead to an improved computational cost of the algorithm. A natural choice is to reduce this variance through control variates (Ripley, 2009).

Control variates applied to SGLD have also been investigated by Dubey et al. (2016); Chen et al. (2017), who show that the convergence bound of SGLD is reduced when they are used. Theoretical results, similar and independent to ours, show how control variates can improve the computational cost of SGLD (Nagapetyan et al., 2017).

In Section 3.3.1, we show how control variates can be used to reduce the variance in the gradient estimate in SGLD, leading to the algorithm SGLD-CV. Then in Section 3.3.3 we analyse the Wasserstein distance between the distribution defined by SGLD-CV and the true posterior. There are a number of quantities that affect the performance of SGLD-CV, including the stepsize h, the number of iterations K and the minibatch size n. We provide sufficient conditions on h, K and n in order to bound the Wasserstein distance. We show under certain assumptions, the computational cost, measured as Kn, required to bound the Wasserstein distance is independent of N.

3.3.1 Control Variates for SGMCMC

Let $\hat{\theta}$ be a fixed value of the parameter, chosen to be close to the mode of the posterior $p(\theta|\mathbf{x})$. The log-posterior gradient can then be re-written as

$$\nabla f(\theta) = \nabla f(\hat{\theta}) + [\nabla f(\theta) - \nabla f(\hat{\theta})],$$

where the first term on the right-hand side is a constant and the bracketed term on the right-hand side can be unbiasedly estimated by

$$\left[\nabla \hat{f}(\theta) - \nabla \hat{f}(\hat{\theta})\right] = \nabla f_0(\theta) - \nabla f_0(\hat{\theta}) + \frac{1}{n}\sum_{i \in S} \frac{1}{p_i}\left[\nabla f_i(\theta) - \nabla f_i(\hat{\theta})\right]$$

where $p_1, \ldots, p_N$ are user-chosen, strictly positive probabilities, S is a random sample from $\{1, \ldots, N\}$ such that $|S| = n$ and the expected number of times i is sampled is np_i. The standard implementation of control variates would set $p_i = 1/N$ for all i. Yet we show below that there can be advantages in having these probabilities vary with i; for example to give higher probabilities to sampling data points for which $\nabla f_i(\theta) - \nabla f_i(\hat{\theta})$ has higher variability with respect to θ.

If the gradient of the likelihood for a single observation is smooth in θ then we will have

$$\nabla f_i(\theta) \approx \nabla f_i(\hat{\theta}) \text{ if } \theta \approx \hat{\theta}.$$

Hence for $\theta \approx \hat{\theta}$ we would expect the unbiased estimator

$$\nabla \tilde{f}(\theta) = \nabla f(\hat{\theta}) + [\nabla \hat{f}(\theta) - \nabla \hat{f}(\hat{\theta})], \tag{3.3.1}$$

to have a lower variance than the simpler unbiased estimator (3.2.2). This is because when θ is close to $\hat{\theta}$ we would expect the terms $\nabla \hat{f}(\theta)$ and $\nabla \hat{f}(\hat{\theta})$ to be positively

Algorithm 1: SGLD-CV

Input: $\hat{\theta}$, $\nabla f(\hat{\theta})$, h.

Set $\theta_0 \leftarrow \hat{\theta}$.

for $k = 0, \ldots, K-1$ **do**

Update $\nabla \tilde{f}(\theta_k)$ using (3.3.1)

Draw $\zeta_k \sim N(0, hI)$

$\theta_{k+1} \leftarrow \theta_k - \frac{h}{2}\nabla \tilde{f}(\theta_k) + \zeta_k$

end

correlated. This reduction in variance is shown formally in Lemma 1, stated in Section 3.3.2.

The gradient estimate (3.3.1) can be substituted into any stochastic gradient MCMC algorithm in place of $\nabla \hat{f}(\theta)$. We refer to SGLD using this alternative gradient estimate as SGLD-CV. The full procedure is outlined in Algorithm 1.

Implementing this in practice means finding a suitable $\hat{\theta}$, which we refer to as the *centring value*. We show below, under a strongly log-concave assumption, that the Wasserstein distance between the distribution defined by SGLD-CV and the posterior distribution can be bounded by some arbitrary constant; and that this can be done such that the computational cost Kn is $O(1)$. For this to be the case though, we require both $\hat{\theta}$ and the starting point of SGLD-CV, θ_0, to be $O(N^{-\frac{1}{2}})$ distance from the posterior mean. This requires some pre-processing steps: an optimisation step to find $\hat{\theta}$, and calculating $f(\hat{\theta})$. These steps are both $O(N)$, but we suggest starting the algorithm from $\hat{\theta}$, meaning the optimisation step essentially replaces the burn-in of the chain. We find in the experiments that these initial steps are often faster than

the burn-in of SGLD.

In practice, we find $\hat{\theta}$ using *stochastic optimisation* (Robbins and Monro, 1951), and then calculate the full log-posterior gradient at this point $\nabla f(\hat{\theta})$. We then start the algorithm from $\hat{\theta}$. In our implementations we use a simple stochastic optimisation method, known as *stochastic gradient descent* (SGD, see e.g. Bottou, 2010). The method works similarly to the standard optimisation method gradient descent, but at each iteration replaces the true gradient of the function with an unbiased estimate. A single update of the algorithm is as follows

$$\theta_{k+1} = \theta_k - h_k \nabla \hat{f}(\theta), \tag{3.3.2}$$

where $\nabla \hat{f}(\theta)$ is as defined in (3.2.2) and $h_k > 0$ is a small tuning constant referred to as the stepsize. Provided the stepsizes h_k satisfy the following conditions $\sum_k h_k^2 < \infty$ and $\sum_k h_k = \infty$ then this algorithm will converge to a local maximum (Robbins and Monro, 1951).

We show in Section 3.3.4, under our assumptions of log-concavity of the posterior, that finding $\hat{\theta}$ using SGD has a computational cost that is linear in N, and we can achieve the required accuracy with just a single pass through the data. As we then start SGLD-CV with this value for θ, we can view finding the centring value as a replacement for the burn-in phase of the algorithm, and we find, in practice, that the time to find a good $\hat{\theta}$ is often quicker than the time it takes for SGLD to burn-in. One downside of this procedure is that the SGD algorithm, as well as the SGLD-CV algorithm itself needs to be tuned, which adds to the tuning burden.

In comparison to SGLD-CV, the SAGA algorithm by Dubey et al. (2016) also uses

control variates to reduce the variance in the gradient estimate of SGLD. They show that this reduces the MSE of SGLD. The main difference is that their algorithm uses a previous state in the chain as the control variate, rather than an estimate of the mode. This means that SAGA does not require the additional optimisation step, so tuning should be easier. However we show in the experiments of Section 3.5, that the algorithm gets more easily stuck in local stationary points, especially during burn-in. For more complex examples, the algorithm was prohibitively slow to burn-in because of this tendency to get trapped. Dubey et al. (2016) also do not show that SAGA has favourable computational cost results.

3.3.2 Variance Reduction

The improvements of using the control variate gradient estimate (3.3.1) over the standard (3.2.2) become apparent when we calculate the variances of each. For our analysis, we make the assumption that the posterior is m-strongly-log-concave and M-smooth, formally defined in Assumption 3.3.1. These assumptions are common when analysing gradient based samplers that do not have an acceptance step (Durmus and Moulines, 2017a; Dalalyan and Karagulyan, 2017). The assumptions imply that $mI \preccurlyeq \nabla^2 f(\theta) \preccurlyeq MI$, for all $\theta \in \mathbb{R}^d$, where I is the identity matrix, and for two matrices $A_1, A_2 \in \mathbb{R}^{d\times d}$, $A_1 \preccurlyeq A_2$ means that $A_2 - A_1$ is positive semi-definite. In all the following analysis we use $\|\cdot\|$ to denote the Euclidean norm.

Assumption 3.3.1. *Strongly log-concave and smooth posterior: there exists positive constants m and M, such that the following conditions hold for the negative log-*

posterior

$$f(\theta) - f(\theta') - \nabla f(\theta')^\top(\theta - \theta') \geq \frac{m}{2}\left\|\theta - \theta'\right\|^2 \tag{3.3.3}$$

$$\left\|\nabla f(\theta) - \nabla f(\theta')\right\| \leq M\left\|\theta - \theta'\right\|. \tag{3.3.4}$$

for all $\theta, \theta' \in \mathbb{R}^d$.

We further need a smoothness condition for each of the likelihood terms in order to bound the variance of our control-variate estimator of the gradient.

Assumption 3.3.2. *Smoothness: there exists constants* $L_0, \ldots, L_N$ *such that*

$$\left\|\nabla f_i(\theta) - \nabla f_i(\theta')\right\| \leq L_i\left\|\theta - \theta'\right\|, \text{ for } i = 0, \ldots, N.$$

Using Assumption 3.3.2 we are able to derive a bound on the variance of the gradient estimate of SGLD-CV. This bound is formally stated in Lemma 3.3.3.

Lemma 3.3.3. *Under Assumption 3.3.2. Let* θ_k *be the state of SGLD-CV at the* k^{th} *iteration, with stepsize* h *and centring value* $\hat{\theta}$*. Assume we estimate the gradient using the control variate estimator with* $p_i = L_i / \sum_{j=1}^N L_j$ *for* $i = 1, \ldots, N$*. Define* $\xi_k := \nabla\tilde{f}(\theta_k) - \nabla f(\theta_k)$*, so that* ξ_k *measures the noise in the gradient estimate* $\nabla\tilde{f}$ *and has mean 0. Then for all* $\theta_k, \hat{\theta} \in \mathbb{R}^d$*, and all* $k = 1, \ldots, K$ *we have*

$$\mathbb{E}\left\|\xi_k\right\|^2 \leq \frac{\left(\sum_{i=1}^N L_i\right)^2}{n}\mathbb{E}\left\|\theta_k - \hat{\theta}\right\|^2. \tag{3.3.5}$$

Here the expectation is over both the noise in the minibatch choice, as well as the distribution of θ_k. All proofs are relegated to the Appendix. It is simple to show that Assumption 3.3.2 also implies that the log-posterior is M-smooth, with $M = \sum_{i=0}^N L_i$,

i.e. condition (3.3.4) in Assumption 3.3.1 holds. This allows us to write

$$\mathbb{E}\left\|\xi_k\right\|^2 \leq \frac{M^2}{n}\mathbb{E}\left\|\theta_k - \hat{\theta}\right\|^2.$$

We will use this form of the bound for the rest of the analysis. In many situations, it is easier to work with a global bound on the smoothness constants, as in Assumption 3.3.4 below, and it is natural to choose $p_i = 1/N$. We use $p_i = 1/N$ in all our implementations in the experiments of Section 3.5.

In order to consider how SGLD-CV scales with N we need to make assumptions on the properties of the posterior and how these change with N. To make discussions concrete we will focus on the following, strong, assumption that each likelihood-term in the posterior is L-smooth and l-strongly-log-concave. As we discuss later, our results apply under weaker conditions.

Assumption 3.3.4. *Assume there exists positive constants L and l such that f_i satisfies the following conditions*

$$f_i(\theta) - f_i(\theta') - \nabla f_i(\theta')^\top(\theta - \theta') \geq \frac{l}{2}\left\|\theta - \theta'\right\|^2$$
$$\left\|\nabla f_i(\theta) - \nabla f_i(\theta')\right\| \leq L\left\|\theta - \theta'\right\|.$$

for all $i \in 0, \ldots, N$ and $\theta, \theta' \in \mathbb{R}^d$.

Under this assumption the constants, m and M, of the posterior both increase linearly with N, as shown by the following Lemma.

Lemma 3.3.5. *Suppose Assumption 3.3.4 holds. Then f satisfies the following*

$$f(\theta) - f(\theta') - \nabla f(\theta')^\top(\theta - \theta') \geq \frac{l(N+1)}{2}\left\|\theta - \theta'\right\|^2$$
$$\left\|\nabla f(\theta) - \nabla f(\theta')\right\| \leq L(N+1)\left\|\theta - \theta'\right\|.$$

Thus the log posterior is M-smooth and m-strongly-concave with parameters $M = (N+1)L$ and $m = (N+1)l$.

We can see that the bound on the gradient estimate variance in (3.3.5) depends on the distance between θ_k and $\hat{\theta}$. Appealing to the Bernstein-von Mises theorem (see e.g. Le Cam, 2012), under standard asymptotics, and provided h is small enough (we make this more formal in the analysis to follow, but it must be at most $O(1/N)$), we would expect the distance $\mathbb{E}\left\|\theta_k - \hat{\theta}\right\|^2$ to be $O(1/N)$, if $\hat{\theta}$ is within $O(N^{-1/2})$ of the posterior mean, once the MCMC algorithm has burnt in. As M is $O(N)$, this suggests that $\mathbb{E}\left\|\xi_k\right\|^2$ will be $O(N)$.

To see the potential benefit of using control variates to estimate the gradient in situations where N is large, we now compare this $O(N)$ result for SGLD-CV, with a result on the variance of the simple estimator, $\nabla\hat{f}(\theta)$. If we randomly pick some data point index I and fix some point $\theta = \vartheta$, then define $V_j(\vartheta)$ to be the empirical variance of $\partial_j f_I(\vartheta)$ over the dataset $\mathbf{x}$; and set $V(\vartheta) = \sum_{j=1}^{d} V_j(\vartheta)$. Then, defining $\hat{\xi}(\theta) = \nabla\hat{f}(\theta) - \nabla f(\theta)$, if we assume we are sampling the minibatch without replacement then

$$\mathbb{E}\left\|\hat{\xi}(\vartheta)\right\|^2 = \frac{N(N-n)}{n}V(\vartheta).$$

Now, suppose that as $N \to \infty$, the posterior converges to some point mass at $\theta_0 \in \mathbb{R}^d$. Then we would expect that, for ϑ close to θ_0, $\mathbb{E}\left\|\hat{\xi}(\vartheta)\right\|^2 \approx \frac{N^2}{n}V(\theta_0)$, so that the estimator will be $O(N^2)$. More precisely, as $N \to \infty$, if we assume we can choose $\epsilon > 0$ such that $V(\vartheta) \geq \sigma^2 > 0$ for all ϑ in an epsilon ball around θ_0; then there is a

constant $c > 0$ such that

$$\frac{n}{N^2}\mathbb{E}\left\|\hat{\xi}(\theta)\right\|^2 \to c, \quad \text{as } N \to \infty. \tag{3.3.6}$$

This suggests using the estimate $\nabla\tilde{f}$, rather than $\nabla\hat{f}$, could give an $O(N)$ reduction in variance, and this plays a key part in the computational cost improvements we show in the next section.

3.3.3 Computational Cost of SGLD-CV

In this section, we investigate how applying control variates to the gradient estimate of SGLD reduces the computational cost of the algorithm.

In order to show this, we investigate the Wasserstein-Monge-Kantorovich (Wasserstein) distance W_2 between the distribution defined by the SGLD-CV algorithm at each iteration and the true posterior as N is changed. For two measures μ and ν defined on the probability space $(\mathbb{R}^d, B(\mathbb{R}^d))$, and for a real number $q \geq 1$, the distance W_q is defined by

$$W_q(\mu,\nu) = \left[\inf_{\gamma\in\Gamma(\mu,\nu)}\int_{\mathbb{R}^d\times\mathbb{R}^d}\|\theta-\theta'\|^q\,d\gamma(\theta,\theta')\right]^{\frac{1}{q}},$$

where the infimum is with respect to all joint distributions Γ having μ and ν as marginals. The Wasserstein distance is a natural distance measure to work with for Monte Carlo algorithms, as discussed in Durmus and Moulines (2017a); Dalalyan and Karagulyan (2017).

One issue when working with the Wasserstein distance is that it is not invariant to transformations. For example scaling all entries of θ by a constant will scale the

Wasserstein distance by the same constant. A linear transformation of the parameters will result in a posterior that is still strongly log-concave, but with different constants m and M. To account for this we suggest measuring error by the quantity $\sqrt{m}W_2$, which is invariant to scaling θ by a constant. Theorem 1 of Durmus and Moulines (2017a) bounds the standard deviation of any component of θ by a constant times $1/\sqrt{m}$, so we can view the quantity $\sqrt{m}W_2$ as measuring the error on a scale that is relative to the variability of θ under the posterior distribution.

There are a number of quantities that will affect the performance of SGLD and SGLD-CV. These include the step size h, the minibatch size n and the total number of iterations K. In the analysis to follow we find conditions on h, n and K that ensure the Wasserstein distance between the distribution defined by SGLD-CV and the true posterior distribution π are less than some $\epsilon > 0$. We use these conditions to calculate the computational cost, measured as Kn, required for this algorithm to reach the satisfactory error ϵ.

The first step is to find an upper bound on the Wasserstein distance between SGLD-CV and the posterior distribution in terms of h, n, K and the constants m and M declared in Assumption 3.3.1.

Proposition 3.3.6. *Under Assumptions 3.3.1 and 3.3.2, let θ_K be the state of SGLD-CV at the K^{th} iteration of the algorithm with stepsize h, initial value θ_0, centring value $\hat{\theta}$. Let the distribution of θ_K be ν_K. Denote the expectation of θ under the posterior distribution π by $\bar{\theta}$. If $h < \frac{2m}{2M^2+m^2}$, then for all integers $K \geq 0$,*

$$W_2(\nu_K, \pi) \leq (1-A)^K W_2(\nu_0, \pi) + \frac{C}{A} + \frac{B^2}{C + \sqrt{A}B},$$

where

$$A = 1 - \sqrt{\frac{2h^2M}{n} + (1 - mh)^2},$$
$$B = \sqrt{\frac{2h^2M^2}{n}\left[\mathbb{E}\left\|\hat{\theta} - \bar{\theta}\right\|^2 + \frac{d}{m}\right]},$$
$$C = \alpha M (h^3 d)^{\frac{1}{2}},$$

$\alpha = 7\sqrt{2}/6$ *and* d *is the dimension of* θ_k.

The proof of this proposition is closely related to the proof of Proposition 2 of Dalalyan and Karagulyan (2017). The extra complication comes from our bound on the variance of the estimator of the gradient; which depends on the current state of the SGLD-CV algorithm, rather than being bounded by a global constant.

We can now use Proposition 3.3.6 to find conditions on K, h and n in terms of the constants M and m such that the Wasserstein distance is bounded by some positive constant $\epsilon_0/\sqrt{m}$ at its final iteration K.

Theorem 3.3.7. *Under Assumptions 3.3.1 and 3.3.2, let* θ_K *be the state of SGLD-CV at the* K^{th} *iteration of the algorithm with stepsize* h*, initial value* θ_0*, centring value* $\hat{\theta}$*. Let the distribution of* θ_K *be* ν_K*. Denote the expectation of* θ *under the posterior distribution* π *by* $\bar{\theta}$*. Define* $R := M/m$*. Then for any* $\epsilon_0 > 0$*, if the following conditions hold:*

$$h \leq \frac{1}{m} \max\left\{\frac{n}{2R^2 + n}, \frac{\epsilon_0^2}{64R^2\alpha^2 d}\right\}, \tag{3.3.7}$$
$$Kh \geq \frac{1}{m} \log\left[\frac{4m}{\epsilon_0^2}\left(\mathbb{E}\left\|\theta_0 - \bar{\theta}\right\|_2^2 + d/m\right)\right], \tag{3.3.8}$$
$$n \geq \frac{64R^2\beta}{\epsilon_0^2} m \left[\mathbb{E}\left\|\hat{\theta} - \bar{\theta}\right\|^2 + \frac{d}{m}\right], \tag{3.3.9}$$

where

$$\beta = \max\left\{\frac{1}{2R^2+1}, \frac{\epsilon_0^2}{64R^2\alpha^2 d}\right\},$$

$\alpha = 7\sqrt{2}/6$*, and* d *is the dimension of* θ_k*, then* $W_2(\nu_K, \pi) \leq \epsilon_0/\sqrt{m}$*.*

As a corollary of this result, we have the following, which gives a bound on the computational cost of SGLD, as measured by Kn, to achieve a required bound on the Wasserstein distance.

Corollary 3.3.8. *Assume that Assumptions 3.3.1 and 3.3.4 and the conditions of Theorem 3.3.7 hold. Fix* ϵ_0 *and define*

$$C_1 = \min\left\{2R^2+1, \frac{64R^2\alpha^2 d}{\epsilon_0^2}\right\}.$$

and $C_2 := 64R^2\beta/\epsilon_0^2$*. We can implement an SGLD-CV algorithm with* $W_2(\nu_K, \pi) < \epsilon_0/\sqrt{m}$ *such that*

$$Kn \leq \left[C_1 \log\left[m\mathbb{E}\left\|\theta_0 - \bar{\theta}\right\|^2 + d\right] + C_1 \log\frac{4}{\epsilon_0^2} + 1\right]\left[C_2 m\mathbb{E}\left\|\hat{\theta} - \bar{\theta}\right\|^2 + C_2 d + 1\right].$$

The constants, C_1 and C_2, in the bound on Kn, depend on ϵ_0 and $R = M/m$. It is simple to show that both constants are increasing in R. Under Assumption 3.3.4 we have that R is a constant as N increases. Corollary 3.3.8 suggests that provided $\left\|\theta_0 - \bar{\theta}\right\| < c/\sqrt{m}$ and $\left\|\hat{\theta} - \bar{\theta}\right\| < c/\sqrt{m}$, for some constant c; then the computational cost of SGLD-CV will be bounded by a constant. Since we suggest starting SGLD-CV at $\hat{\theta}$, then under the conditions of Corollary 3.3.8, we just need this to hold for $\left\|\hat{\theta} - \bar{\theta}\right\|$. Under Assumption 3.3.4 we have that m increases linearly with N, so this corresponds to needing $\left\|\hat{\theta} - \bar{\theta}\right\| < c_1/\sqrt{N}$ as N increases. Additionally, by Theorem 1 of Durmus and Moulines (2017a) we have that the variance of the posterior scales

like $1/m = 1/N$ as N increases, so we can interpret the $1/\sqrt{N}$ factor as being a measure of the spread of the posterior as N increases. The form of the corollary makes it clear that a similar argument would apply under weaker assumptions than Assumption 3.3.4. We only need that the ratio of the log-concavity constants, M/m, of the posterior remains bounded as N increases.

This corollary also gives insight into the computational penalty you pay for a poor choice of θ_0 or $\hat{\theta}$. The bound on the computational cost will increase logarithmically with $\left\|\theta_0 - \bar{\theta}\right\|$ and linearly with $\left\|\hat{\theta} - \bar{\theta}\right\|$.

Similar bounds for the computational cost of SGLD have been derived (e.g., Nagapetyan et al., 2017; Chatterji et al., 2018). For example, Chatterji et al. (2018) state the computational cost of SGLD in order to sample from a distribution with W_2 distance that is within ϵ distance of the target is $O(1/\epsilon^2)$. For our required level of accuracy, $\epsilon = \epsilon_0/\sqrt{m}$, this corresponds to a bound on the computational cost that is $O(N)$, as compared to $O(1)$ for SGLD-CV. However, a formal proof that SGLD is $O(N)$ requires showing that any implementation of SGLD with Kn that is sublinear in N cannot reach arbitrary accuracy ϵ. This requires a lower bound on the W_2 distance, rather than an upper bound. Whilst the result of Chatterji et al. (2018) is just an upper bound on the W_2 distance, there are empirical results that suggest a linear computational cost for SGLD, including those in Nagapetyan et al. (2017), as well as our own experiments in Section 3.5.

3.3.4 Setup Costs

There are a number of known results on the convergence of SGD under the strongly log-concave conditions of Assumption 3.3.1. These will allow us to quantify the setup cost of finding the point $\hat{\theta}$ in this setting. More complex cases are explored empirically in the experiments in Section 3.5. Lemma 3.3.9 due to Nemirovski et al. (2009) quantifies the convergence of the final point of SGD.

Lemma 3.3.9. *(Nemirovski et al., 2009) Under Assumption 3.3.1, let $\hat{\theta}$ denote the final state of SGD with stepsizes $h_k = 1/(mk)$ after K iterations. Suppose $\mathbb{E}\left\|\nabla \hat{f}(\theta)\right\|^2 \leq D^2$ and denote the true mode of f by θ^*. Then it holds that*

$$\mathbb{E}\left\|\hat{\theta} - \theta^*\right\|^2 \leq \frac{4D^2}{m^2 K}.$$

By using a similar argument to (3.3.6), we would expect that D^2 is $O(N^2/n)$. This means that under Assumption 3.3.4, we will need to process the full dataset once before the SGD algorithm has converged to an estimate of the mode $\hat{\theta}$ within $O(N^{-\frac{1}{2}})$ of the posterior mean. It follows that, for these cases there are two one off $O(N)$ setup costs, one to find an acceptable mode $\hat{\theta}$ and one to find the full log-posterior gradient at this point $\nabla f(\hat{\theta})$.

3.4 Post-processing Control Variates

Control variates can also be used to improve the inferences made from MCMC by reducing the variance of the output directly. The general aim of MCMC is to estimate expectations of functions, $g(\theta)$, with respect to the posterior π. Given an MCMC

sample $\theta^{(1)}, \ldots, \theta^{(M)}$, from the posterior π, we can estimate $\mathbb{E}[g(\theta)]$ unbiasedly as

$$\mathbb{E}[g(\theta)] \approx \frac{1}{M} \sum_{i=1}^{M} g(\theta^{(i)}).$$

Suppose there exists a function $h(\theta)$, which has expectation 0 under the posterior. We can then introduce an alternative function,

$$\tilde{g}(\theta) = g(\theta) + h(\theta),$$

where $\mathbb{E}[\tilde{g}(\theta)] = \mathbb{E}[g(\theta)]$. If $h(\cdot)$ is chosen so that it is negatively correlated with $g(\theta)$, then the variance of $\tilde{g}(\theta)$ will be reduced considerably.

Mira et al. (2013) introduce a way of choosing $h(\theta)$ almost automatically by using the gradient of the log-posterior. Choosing $h(\cdot)$ in this manner is referred to as a zero-variance (ZV) control variate. Friel et al. (2016) showed that, under mild conditions, we can replace the log-posterior gradient with an unbiased estimate and still have a valid control variate. SGMCMC methods produce unbiased estimates of the log-posterior gradient, and so it follows that these gradient estimates can be applied as ZV control variates. For the rest of this section, we focus our attention on SGLD, but these ideas are easily extendable to other stochastic gradient MCMC algorithms. We refer to SGLD with these post-processing control variates as SGLD-ZV.

Given the setup outlined above, Mira et al. (2013) propose the following form for $h(\theta)$,

$$h(\theta) = \Delta Q(\theta) + \nabla Q(\theta) \cdot \mathbf{z},$$

here $Q(\theta)$ is a polynomial of θ to be chosen and $\mathbf{z} = \nabla f(\theta)/2$. Δ refers to the *Laplace operator* $\frac{\partial^2}{\partial \theta_1^2} + \cdots + \frac{\partial^2}{\partial \theta_d^2}$. This is a good form for h because $\mathbb{E}_\pi[h(\theta)] = 0$. This

can be seen in two ways, the first is simply to use integration by parts. The second is to notice that $h(\theta) = \mathcal{A}Q(\theta)$, where $\mathcal{A}$ is the infinitesimal generator of Langevin diffusion, as defined in (2.3.4). If $\mathcal{A}$ is the generator of a continuous-time Markov process with unique stationary distribution π; then mild conditions on $\mathcal{A}$ and Q ensure that $\mathbb{E}_\pi[\mathcal{A}Q(\theta)] = 0$ (Barbour, 1988). In order to get the best variance reduction, we simply have to optimize the coefficients of the polynomial $Q(\cdot)$. In practice, first or second degree polynomials $Q(\theta)$ often provide good variance reduction (Mira et al., 2013). For the rest of this section we focus on first degree polynomials, so $Q(\theta) = \mathbf{a}^T\theta$, but the ideas are easily extendable to higher orders (Papamarkou et al., 2014).

The SGLD algorithm only calculates an unbiased estimate of $\nabla f(\theta)$, so we propose replacing $h(\theta)$ with the unbiased estimate

$$\hat{h}(\theta) = \Delta Q(\theta) + \nabla Q(\theta) \cdot \hat{\mathbf{z}}, \tag{3.4.1}$$

where $\hat{\mathbf{z}} = \nabla \hat{f}(\theta)/2$. By identical reasoning to Friel et al. (2016), $\hat{h}(\theta)$ is a valid control variate. Note that $\hat{\mathbf{z}}$ can use any unbiased estimate, and as we will show later, the better the gradient estimate, the better this control variate performs. The expectation of $\hat{h}$ is zero because $\mathbb{E}[h(\theta)] = 0$, and $\hat{z}$ is an unbiased estimate of z. See the Appendix for the full calculation.

As $Q(\theta)$ is a linear polynomial $\mathbf{a}^T\theta$, so our SGLD-ZV estimate will take the following form

$$\hat{g}(\theta) = g(\theta) + \mathbf{a}^T\hat{\mathbf{z}}. \tag{3.4.2}$$

Similar to standard control variates (Ripley, 2009), we need to find optimal coefficients $\hat{\mathbf{a}}$ in order to minimise the variance of $\tilde{g}(\cdot)$, defined in (3.4.2). In our case, the optimal

Algorithm 2: SGLD-ZV

Input: SGLD Output: $\{\theta_k, \nabla \hat{f}(\theta_k)\}_{k=1}^K$

Set $\mathbf{z}_k \leftarrow \frac{1}{2}\nabla \hat{f}(\theta_k)$

Estimate $V_{\mathbf{z}} \leftarrow \mathbb{V}\text{ar}(\mathbf{z})$, $C_{g,\mathbf{z}} \leftarrow \text{Cov}(g(\theta), \mathbf{z})$

$\hat{\mathbf{a}}_j \leftarrow [V_{\mathbf{z}}]^{-1} C_{g,\mathbf{z}}$

for $k = 1 \dots K$ **do**
  $\hat{g}(\theta_k) \leftarrow g(\theta_k) + \hat{\mathbf{a}}^T \mathbf{z}_k$

end

coefficients take the following form (Friel et al., 2016)

$$\hat{\mathbf{a}} = \mathbb{V}\text{ar}^{-1}(\hat{\mathbf{z}})\, \mathbb{C}\text{ov}(\hat{\mathbf{z}}, g(\theta)).$$

This means that SGLD already calculates all the necessary terms for these control variates to be applied for free. So the post-processing step can simply be applied once when the SGLD algorithm has finished, provided the full output plus gradient estimates are stored. The full details are given in Algorithm 2. For higher order polynomials, the calculations are much the same, but more coefficients need to be estimated (Papamarkou et al., 2014).

The efficiency of ZV control variates in reducing the variance of our MCMC sample is directly affected by using an estimate of the gradient rather than the truth. For the remainder of this section, we investigate how the choice of the gradient estimate, and the minibatch size n, affects the variance reduction.

Assumption 3.4.1. $\mathbb{V}\text{ar}[g(\theta)] < \infty$ *and* $\mathbb{V}\text{ar}[\hat{h}(\theta)] < \infty$. $\mathbb{E}_\pi \|\nabla f_i(\theta)\|^2$ *is bounded by some constant* σ *for all* $i = 0, \dots N$, $\theta \in \mathbb{R}^d$.

Theorem 3.4.2. *Under Assumption 3.4.1, define the optimal variance reduction for ZV control variates using the full gradient estimate to be R, and the optimal variance reduction using SGLD gradient estimates to be $\hat{R}$. Then we have that*

$$\hat{R} \geq \frac{R}{1 + [\sigma(N+1)]^{-1}\mathbb{E}_{\theta|\mathbf{x}}[\mathbb{E}_S \left\|\xi_S(\theta)\right\|^2]}, \tag{3.4.3}$$

where $\xi_S(\theta)$ is the noise in the log-posterior gradient estimate.

The proof of this result is given in the Appendix. An important consequence of Theorem 3.4.2 is that if we use the standard SGLD gradient estimate, then the denominator of (3.4.3) is $O(n/N)$, so (provided the bound is tight) our variance reduction diminishes as N gets large. However, if we use the SGLD-CV estimate instead, then under standard asymptotics, the denominator of (3.4.3) is $O(n)$, so the variance reduction does not diminish with increasing dataset size. The same holds for SAGA, and other control variate algorithms that reduce the gradient error to $O(N)$. It follows that for best results, we recommend using the ZV post-processing step after running the SGLD-CV algorithm, especially for large N. The ZV post-processing step can be immediately applied in exactly the same way to other stochastic gradient MCMC algorithms, such as SGHMC and SGNHT (Chen et al., 2014; Ding et al., 2014).

It is worth noting that there are some storage constraints for SGLD-ZV. This algorithm requires storing the full MCMC chain, as well as the gradient estimates at each iteration. So the storage cost is twice the storage cost of a standard SGMCMC run. However, in some high dimensional cases, the required SGMCMC test statistic is estimated on the fly using the most recent output of the chain and thus reducing

the storage costs. We suggest that if the dimensionality is not too high, then the additional storage cost of recording the gradients to apply the ZV post-processing step can offer significant variance reduction for free. However, for very high dimensional parameters, the cost associated with storing the gradients may preclude the use of the ZV step.

3.5 Experiments

In the experiments to follow we compare SGLD-CV to SGLD in a number of different scenarios. We also compare to the method SAGA (Dubey et al., 2016), an alternative variance reduction method for SGLD discussed at the end of Section 3.3.1. Performance is measured by plotting the log predictive density of a held out test set at each iteration. Some of our examples are high dimensional, so our performance measure aims to reduce dimensionality while still capturing important quantities such as the variance of the chain. To empirically demonstrate the scalability advantages, for each experiment we fit the models with different proportions of the full dataset. For example, in the logistic regression experiment we run the algorithms with dataset sizes $0.01N$, $0.1N$ and N; where N is the full dataset size. The pre-processing steps for SGLD-CV are included in the plots, as a result we also include the burn-in of SGLD and SAGA in the plots to contrast with this.

We also provide boxplots of the log predictive density of SGLD-CV at each iteration, before and after it has been post-processed using ZV control variates; to demonstrate the variance reduction after ZV control variates are applied.

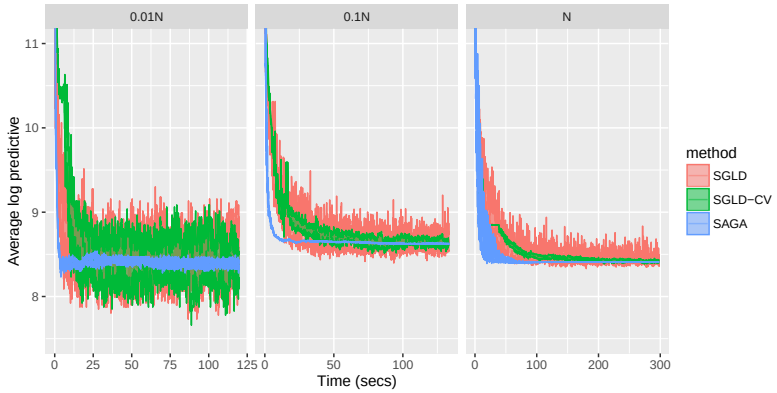


Figure 3.5.1: Log predictive density over a test set every 10 iterations of SGLD, SGLD-CV and SAGA fit to a logistic regression model as the proportion of data used is varied (as compared to the full dataset size N).

A fixed stepsize scheme is used for all methods, and these are tuned using a grid search (for SGLD-CV, both SGD and SGLD steps are tuned using a grid search). Full details of the tuning constants, and alternative results using a decreasing stepsize scheme are given in the Appendix. The minibatch size is fixed across all the dataset sizes. All results are timed when the algorithms are run on four cores of a 2.3 GHz Intel Xeon CPU.

3.5.1 Logistic Regression

We examine our approaches on a Bayesian logistic regression problem. The probability of the i^{th} output $y_i \in \{-1, +1\}$ is given by

$$p(y_i|x_i, \beta) = \frac{1}{1 + \exp(-y_i \beta^T x_i)}.$$

We use a Laplace prior for β with scale 1.

We used the cover type dataset (Blackard and Dean, 1999), which has 581012

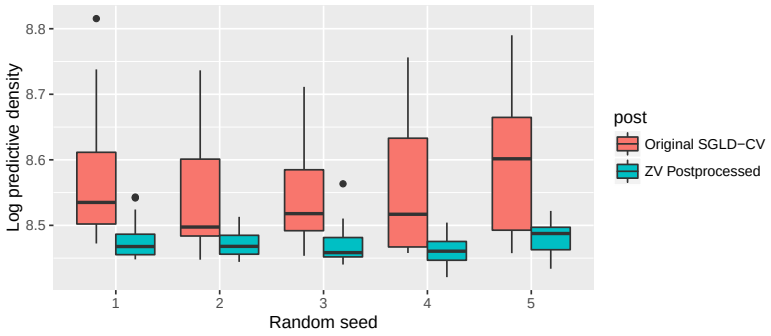


Figure 3.5.2: Plots of the log predictive density of an SGLD-CV chain when ZV post-processing is applied versus when it is not, over 5 random runs. Logistic regression model on the cover type dataset (Blackard and Dean, 1999).

observations, which we split into a training and test set. First we run SGLD, SGLD-CV and SAGA on the dataset, all with minibatch size 500. The model is run with three different dataset sizes, from about 1% of the full dataset to the full dataset size N. The performance is measured by calculating the log predictive density on a held-out test set every 10 iterations.

The results are plotted against time in Figure 3.5.1. The results illustrate the efficiency gains of SGLD-CV over SGLD as the dataset size increases, as expected from Theorem 3.3.8. SAGA outperforms SGLD-CV in this example because SAGA converges quickly in this simple setting. In the more complicated examples to follow, we show that SAGA can get stuck in local stationary points.

We also compare the log predictive density over a test set for SGLD-CV with and without ZV post-processing, averaged over 5 runs at different seeds. We apply the method to SGLD-CV rather than SGLD due to the favourable scaling results as discussed after Theorem 3.4.2. Results are given in Figure 3.5.2. The plot shows

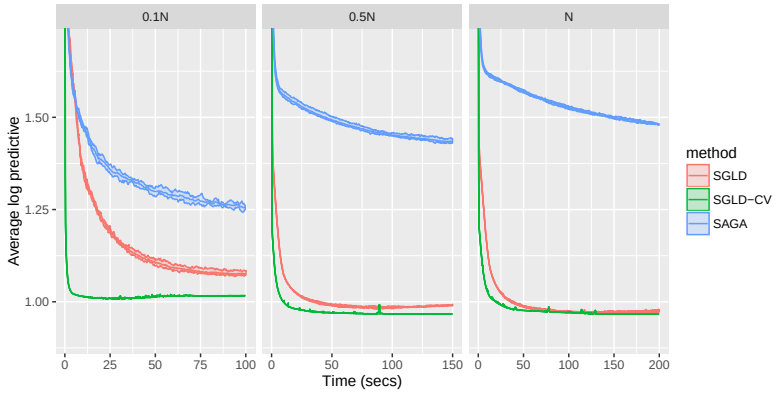


Figure 3.5.3: Log predictive density over a test set of SGLD, SGLD-CV and SAGA fit to a Bayesian probabilistic matrix factorisation model as the number of users is varied, averaged over 5 runs. We used the Movielens ml-100k dataset.

box-plots of the log predictive density of the SGLD sample before and after post-processing using ZV control variates. The plots show good variance reduction of the chain.

3.5.2 Probabilistic Matrix Factorisation

A common recommendation system task is to predict a user's rating of a set of items, given previous ratings and the ratings of other users. The end goal is to recommend new items that the user will rate highly. Probabilistic matrix factorisation (PMF) is a popular method to train these models (Mnih and Salakhutdinov, 2008). As the matrix of ratings is sparse, over-fitting is a common issue in these systems, and Bayesian approaches are a way to account for this (Chen et al., 2014; Ahn et al., 2015).

In this experiment, we apply SGLD, SGLD-CV and SAGA to a Bayesian PMF

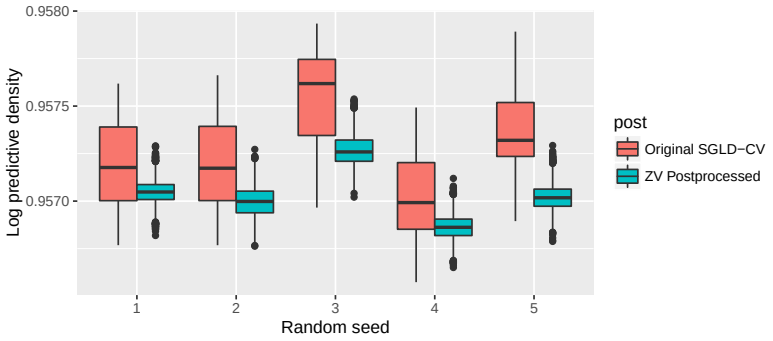


Figure 3.5.4: Plots of the log predictive density of an SGLD-CV chain when ZV post-processing is applied versus when it is not, over 5 random runs. SGLD-CV algorithm applied to a Bayesian probabilistic matrix factorisation problem using the Movielens ml-100k dataset.

(BPMF) problem, using the formulation of Chen et al. (2014). The data for BPMF is a matrix of ratings R, of size $L \times M$, where L is the number of users and M is the number of items. Each entry contains a rating of a particular item by that user, with a lot of missing entries. The aim is to predict the values of the missing entries. This is done by factorising R into two matrices U and V, so that $R \approx U^T V$, where U and V are size $D \times L$ and $D \times M$ respectively. Here D is some user specified parameter, which we choose to be 20. We use the Movielens dataset ml-100k[1], which contains 100,000 ratings from almost 1,000 users and 1,700 movies. We use batch sizes of 5,000, with a larger minibatch size chosen due to the high-dimensional parameter space. As before, we compare performance by calculating the log predictive density on a held out testset every 10 iterations. Full details of chosen hyperparameters are detailed in the Appendix.

[1]https://grouplens.org/datasets/movielens/100k/

In this case, we vary the dataset size by limiting the number of users in the dataset, ranging from 100 users to the full 943. The results are given in Figure 3.5.3. In this example SAGA converges slowly in comparison even to SGLD. In fact the algorithm converges slowly in all our more complex experiments. The problem is particularly bad for large N. This is likely a result of the starting point for SAGA being far from the posterior mode. Empirically, we found that the gradient direction and magnitude can update very slowly in these cases. This is not an issue for simpler examples such as logistic regression, but for more complex examples we believe it could be a sign that the algorithm is getting stuck in, or moving slowly through, local modes where the gradient is comparatively flatter. The problem appears to be made worse for large N when it takes longer to update g_α. This is an example where the optimisation step of SGLD-CV is an advantage, as the algorithm is immediately started close to the posterior mode and so the efficiency gains are quickly noted. This issue with SAGA could be related to the starting point condition for SGLD-CV as detailed in Corollary 3.3.8. Due to the form of the Wasserstein bound, it is likely that SAGA would have a similar starting point condition. It appears that the speed of the burn-in of SGLD becomes more competetive with the initial SGD step for SGLD-CV as the dataset size increases. Despite this, a close look at the plots still shows good variance reduction for SGLD-CV and better log predictive density scores.

Once again we compare the log predictive density over a test set for SGLD-CV with and without ZV post-processing when applied to the Bayesian PMF problem, averaged over 5 runs at different seeds. Results are given in Figure 3.5.2. The plot shows box-plots of the log predictive density of the SGLD sample before and after

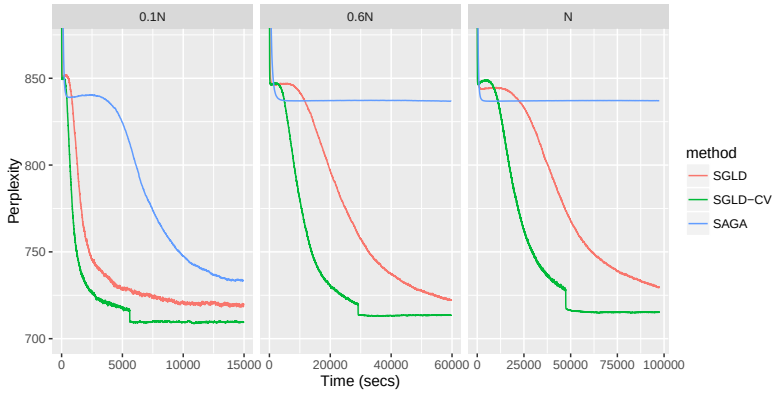


Figure 3.5.5: Perplexity of SGLD and SGLD-CV fit to an LDA model as the data size N is varied, averaged over 5 runs. The dataset consists of scraped Wikipedia articles.

post-processing using ZV control variates. The plots show excellent variance reduction of the chain.

3.5.3 Latent Dirichlet Allocation

Latent Dirichlet allocation (LDA) is an example of a topic model used to describe collections of documents by sets of discovered topics (Blei et al., 2003). The input consists of a matrix of word frequencies in each document, which is very sparse, motivating the use of a Bayesian approach to avoid over-fitting.

Due to storage constraints, it was not feasible to apply SGLD-ZV to this problem, so we focus on SGLD-CV. We scraped approximately 80,000 documents from Wikipedia, and used the 1,000 most common words to form our document-word matrix input. We used a similar formulation to Patterson and Teh (2013), though we did not use a Riemannian sampler, which should test the methods due to the gradient instability. We also instead use the expanded-natural parameterisation, since accord-

ing to empirical results by Patterson and Teh (2013) that was the best performing parameterisation for the non-Riemannian case. We set the number of topics K to be 20. Full details of hyperparameters are given in the Appendix.

Once again in our comparison of SGLD, SGLD-CV and SAGA, we vary the dataset size, this time by changing the number of documents used in fitting the model, from 10,000 to the full 81,538. We use batch sizes of 50 documents. We measure the performance of LDA using the *perplexity* on held out words from each document, a standard performance measure for this model, for more detail see Patterson and Teh (2013). The results are given in Figure 3.5.5. Here the scalability improvements of using SGLD-CV over SGLD are clear as the dataset size increases. This time the batch size is small compared to the dataset size, which probably makes the scalability improvements more obvious. The sudden drop in perplexity for the SGLD-CV plot occurs at the switch from the stochastic optimisation step to SGLD-CV. This is likely a result of the algorithm making efficient use of the Gibbs step to simulate the latent topics.

An interesting aspect of this problem is that it appears to have a pronounced local mode where each of the methods become trapped (this can be seen by the blip in the plot at a perplexity of around 850). SGLD-CV is the first to escape followed by SGLD, but SAGA takes a long time to escape. This is probably due to a similar aspect as the one discussed in the previous experiment (Section 3.5.2). Similar to the previous experiment, we find that while SAGA seems trapped, its gradient estimate changes very little, which could be a sign that the algorithm is moving very slowly through an area with a relatively flat gradient, such as a local mode. A simple solution would

be to start SAGA closer to the mode using a stochastic optimisation scheme.

3.6 Discussion

We have used control variates for stochastic gradient MCMC to reduce the variance in the gradient estimate. We have shown that in the strongly log-concave setting, and under standard asymptotics, this proposed SGLD-CV algorithm reduces the computational cost of stochastic gradient Langevin dynamics to $O(1)$. Our theoretical results give results on the computational cost under non-standard asymptotics also, and show there should be some benefit provided distance between the centring value $\hat{\theta}$ and the posterior mean inversely depends on N. The algorithm relies on a setup cost that estimates the posterior mode which replaces the burn-in of SGLD. We have explored the cost of this step both theoretically and empirically. We have empirically supported these scalability results on a variety of interesting and challenging problems from the statistics and machine learning literature using real world datasets. The simulation study also revealed that SGLD-CV was less susceptible to getting stuck in local stationary points than an alternative method that performs variance reduction using control variates, SAGA (Dubey et al., 2016). An interesting future extension would be to reduce the start-up cost of SGLD-CV, along with introducing automatic step-size tuning.

We showed that stochastic gradient MCMC methods calculate all the information needed to apply zero-variance post-processing control variates. This improves the inference of the output by reducing its variance. We explored how the variance

reduction is affected by the minibatch size and the gradient estimate, and show using SGLD-CV or SAGA rather than SGLD can achieve a better variance reduction. We demonstrated this variance reduction empirically. A limitation of these post-processing control variates is they require the whole chain, which can lead to high storage costs if the dimensionality of the sample space is high. Future work could explore ways to reduce the storage costs of stochastic gradient MCMC.

3.7 Acknowledgements

The first author gratefully acknowledges the support of the EPSRC funded EP/L015692/1 STOR-i Centre for Doctoral Training. This work was supported by EPSRC grants EP/K014463/1, EP/R018561/1, EP/S00159X/1 and EP/R01860X/1 and ONR Grant N00014-15-1-2380 and NSF CAREER Award IIS-1350133.

Chapter 4

sgmcmc: An R Package for Stochastic Gradient Markov Chain Monte Carlo

4.1 Introduction

This article introduces sgmcmc, an R package (R Development Core Team, 2008) for scalable Bayesian inference on a wide variety of models using stochastic gradient Markov chain Monte Carlo (SGMCMC). A disadvantage of most traditional MCMC methods are that they require calculations over the full dataset at each iteration; meaning the methods are prohibitively slow for large datasets. SGMCMC methods provide a solution to this issue. The methods use only a subset of the full dataset, known as a *minibatch*, at each MCMC iteration. While the methods no longer target the true posterior, they instead produce accurate approximations to the posterior at

a reduced computational cost.

The sgmcmc package implements a number of popular SGMCMC samplers including stochastic gradient Langevin dynamics (SGLD) (Welling and Teh, 2011), stochastic gradient Hamiltonian Monte Carlo (SGHMC) (Chen et al., 2014) and stochastic gradient Nosé-Hoover thermostats (SGNHT) (Ding et al., 2014). Recent work has shown how control variates can be used to reduce the computational cost of SGMCMC algorithms (Baker et al., 2018; Nagapetyan et al., 2017). For each of the samplers implemented in the package, there is also a corresponding control variate sampler providing improved sampling efficiency.

Performing statistical inference on a model using SGMCMC requires calculating the gradient of the log likelihood and log priors. Calculating gradients by hand is often time consuming and error prone. One of the major advantages of sgmcmc is that gradients are calculated within the package using automatic differentiation (Griewank and Walther, 2008). This means that users need only specify the log likelihood function and log prior for their model. The package calculates the gradients using TensorFlow (TensorFlow Development Team, 2015), which has recently been made available for R (Allaire et al., 2016). TensorFlow is an efficient library for numerical computation which can take advantage of a wide variety of architectures, as such, sgmcmc keeps much of this efficiency. Both sgmcmc and TensorFlow are available on CRAN, so sgmcmc can be installed by using the standard `install.packages` function. Though after the TensorFlow package has been installed, the extra `install_tensorflow()` function needs to be run, which installs the required Python implementation of Ten-

sorFlow.[1] The sgmcmc package also has a website with vignettes, tutorials and an API reference.[2]

SGMCMC methods have become popular in the machine learning literature but less so in the statistics community. We partly attribute this to the lack of available software. To the best of our knowledge, there are currently no R packages available for SGMCMC, probably the most popular programming language within the statistics community. The only package we are aware of which implements scalable MCMC is the Python package edward (Tran et al., 2016). This package implements both SGLD and SGHMC, but does not implement SGNHT or any of the control variate methods.

Section 4.2 introduces MCMC and discusses the software currently available for implementing MCMC algorithms, we discuss the scenarios where sgmcmc is designed to be used. In Section 4.3 we review the methodology behind the SGMCMC methods implemented in sgmcmc. Section 4.4 provides a brief introduction to TensorFlow. Section 4.5 overviews the structure of the package, as well as details of how the algorithms are implemented. Section 4.6 presents a variety of example simulations. Finally, Section 4.7 provides a discussion on benefits and drawbacks of the implementation, as well as how we plan to extend the package in the future.

[1] More information on installing TensorFlow for R can be found at `https://tensorflow.rstudio.com/`.

[2] sgmcmc website at `https://stor-i.github.io/sgmcmc`

4.2 Introduction to MCMC and Available Software

Suppose we have a dataset of size N, with data $\mathbf{x} = \{x_i\}_{i=1}^N$, where $x_i \in \mathcal{X}$ for some space $\mathcal{X}$. We denote the probability density of x_i as $p(x_i|\theta)$, where $\theta \in \Theta \subseteq \mathbb{R}^p$ are model parameters to be inferred. We assign a prior density $p(\theta)$ to the parameters and the resulting posterior is then $p(\theta|\mathbf{x}) \propto p(\theta)\prod_{i=1}^N p(x_i|\theta)$.

Often the posterior can only be calculated up to a constant of proportionality Z. In practice Z is rarely analytically tractable; so MCMC provides a way to construct a Markov chain using only the unnormalized posterior density $h(\theta) := p(\theta)\prod_{i=1}^N p(x_i|\theta)$. The Markov chain is designed so that its stationary distribution is the posterior $p(\theta|\mathbf{x})$. The result (once the chain has converged) is a sample $\{\theta_t\}_{t=1}^\top$ from the posterior, though this sample is not independent. A downside of these traditional MCMC algorithms is that they require the evaluation of the unnormalized density $h(\theta)$ at every iteration. This results in an $O(N)$ cost per iteration. Thus MCMC becomes prohibitively slow on large datasets.

The Metropolis-Hastings algorithm is a type of MCMC algorithm. New proposed samples θ' are drawn from a proposal distribution $q(\theta'|\theta)$ and then accepted with probability,

$$\min\left\{1, \frac{p(\theta'|\mathbf{x})q(\theta|\theta')}{p(\theta|\mathbf{x})q(\theta'|\theta)}\right\}. \tag{4.2.1}$$

Notice that the normalising constant Z cancels in (4.2.1), so we can interchange the posterior $p(\theta|\mathbf{x})$ with $h(\theta)$. The efficiency of the Metropolis-Hastings algorithm is dependent on the choice of proposal distribution, q.

There are a number of proposals for the Metropolis-Hastings algorithm which can have a very high acceptance rate. Some examples are the Metropolis-adjusted Langevin algorithm (MALA; see e.g., Roberts and Rosenthal (1998)) and Hamiltonian Monte Carlo (HMC; see Neal (2010)). The reason these proposals achieve such high acceptance rates is that they approximate a continuous diffusion process whose stationary distribution is $p(\theta|\mathbf{x})$. As an example, consider the MALA algorithm. The MALA algorithm uses a Euler discretisation of the Langevin diffusion as the proposal,

$$q(\theta'|\theta) = \mathcal{N}\left(\theta' \mid \theta + \frac{\epsilon}{2}\nabla_\theta \log p(\theta|\mathbf{x}), \epsilon \mathrm{I}\right), \tag{4.2.2}$$

where $\mathcal{N}(\theta|\mu, \Sigma)$ denotes a multivariate Normal density evaluated at θ with mean μ and variance Σ; I is simply the identity matrix; ϵ is a tuning parameter referred to as the *stepsize*. Discretising the diffusion introduces an approximation error, which is corrected by the Metropolis-Hastings acceptance step (4.2.1). This means that as $\epsilon \to 0$, we tend back towards the exact, continuous diffusion and the acceptance rate is 1. However this would result in a Markov chain that never moves. Thus picking ϵ is a balance between a high acceptance rate and good mixing.

There are a number of general purpose samplers for MCMC that fulfil different purposes to sgmcmc. The most popular samplers are Stan, BUGS and JAGS (Carpenter et al., 2017; Plummer, 2003; Lunn et al., 2000). The samplers BUGS and JAGS implement automated Gibbs sampling. These samplers work with both continuous and discrete parameter spaces and can be highly efficient. However because the samplers rely on Gibbs sampling, conjugate priors need to be used; also the samplers are not efficient when there is high correlation between the parameters (Carpenter

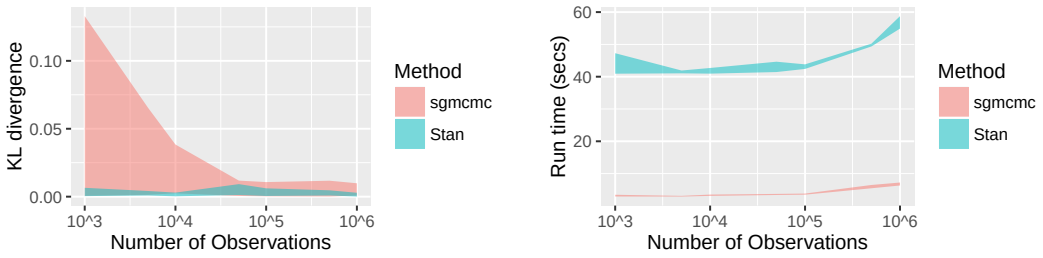


Figure 4.2.1: KL divergence (left) and run time (right) of the standard Stan algorithm and the `sgldcv` algorithm of the sgmcmc package when each are used to sample from data following a standard Normal distribution as the number of observations are increased.

et al., 2017). The package Stan implements state of the art Hamiltonian Monte Carlo techniques, which means non-conjugate priors can be used, and that the sampler is more robust when there is correlation between parameters. However the package cannot perform inference on discrete parameters, and requires that these are integrated out of the model.

The sgmcmc package aims to fill a gap when the dataset is large enough that other general purpose MCMC samplers such as Stan, BUGS and JAGS cannot be run or run prohibitively slowly. In the packages Stan, BUGS and JAGS, properly specified models will define a Markov chain whose stationary distribution is the posterior distribution. However a major problem with these methods are that as the number of observations gets large the algorithms run slowly. This has become a problem as dataset sizes have been increasing. The reason these methods are slow when running on large datasets are because they require a calculation over the full dataset at each iteration. The methods implemented in sgmcmc aim to account for this issue by only

using a subset of the dataset at each iteration. The main downside being that the stationary distribution is no longer the true posterior, just a close approximation. However, as the dataset size increases, the main tuning constant in sgmcmc, known as the stepsize, can be set smaller and the approximation to the posterior improves. Compared to Stan, BUGS and JAGS; sgmcmc offers significant computational advantages for Bayesian modelling with large datasets, but like Stan, a downside of sgmcmc is it requires that discrete parameters are integrated out. The package also requires more tuning than other general purpose samplers since satisfactory results for tuning these methods are still under development.

Figure 4.2.1 demonstrates in which scenarios practitioners may find sgmcmc useful. The standard Stan sampler and the `sgldcv` algorithm of the sgmcmc package are used to sample from the posterior of data drawn from a standard Normal with a $N(0, 10)$ prior. The KL divergence between the MCMC sample and the true posterior is calculated, and the plots show how this and the run time changes as the number of observations are increased. Since both Stan and TensorFlow models need to be compiled, we recompile the models each time they are run to keep the comparison fair; but it is worth mentioning that the Stan run time is much quicker for the small observation models if precompiled. We can see the run time of Stan increasing rapidly as the number of observations is increased, while the run time of the sgmcmc algorithm increases more slowly. This is a very simple model, used so that the KL divergence can be calculated exactly. As the model complexity increases the run time of Stan can quickly become unmanageable for large datasets. On the other hand, we can see that the KL divergence of the sgmcmc algorithm for this example is poor compared with

Stan for a small number of observations. However as the dataset size increases the KL divergence for the sgmcmc algorithm becomes much more reasonable compared with Stan. Thus sgmcmc is best used when the dataset size is slowing down the run time of the other general purpose algorithms, and practitioners can safely trade-off a small amount of accuracy in order to gain significant speed-ups.

4.3 Stochastic Gradient MCMC

Many popular MCMC proposal distributions, including HMC and MALA, described in (4.2.2), require the calculation of the gradient of the log posterior at each iteration, which is an $O(N)$ calculation. Stochastic gradient MCMC methods get around this by replacing the true gradient with the following unbiased estimate

$$\nabla_\theta \widehat{\log p(\theta_t|\mathbf{x})} := \nabla_\theta \log p(\theta_t) + \frac{N}{n} \sum_{i \in S_t} \nabla_\theta \log p(x_i|\theta_t), \tag{4.3.1}$$

calculated on some subset of the all observations $S_t \subset \{1, \ldots, N\}$, known as a *mini-batch*, with $|S_t| = n$.

Calculating the Metropolis-Hastings acceptance step (4.2.1) is another $O(N)$ calculation. To get around this, SGMCMC methods set the tuning constants such that the acceptance rate will be high, and remove the acceptance step from the algorithm altogether. By ignoring the acceptance step, and estimating the log posterior gradient, the per iteration cost of SGMCMC algorithms is $O(n)$, where n is the minibatch size. However, the algorithm no longer targets the true posterior but an approximation. There has been a body of theory exploring how these methods perform in different settings. Similar to MALA, the algorithms rely on a stepsize parameter ϵ. Some of

the algorithms have been shown to weakly converge as $\epsilon \to 0$.

4.3.1 Stochastic Gradient Langevin Dynamics

SGLD, first introduced by Welling and Teh (2011), is an SGMCMC approximation to the MALA algorithm. By substituting (4.3.1) into the MALA proposal equation (4.2.2), we arrive at the following update for θ

$$\theta_{t+1} = \theta_t + \frac{\epsilon_t}{2}\nabla_\theta \widehat{\log p(\theta_t|\mathbf{x})} + \zeta_t, \tag{4.3.2}$$

where $\zeta_t \sim \mathcal{N}(0, \epsilon_t)$.

Welling and Teh (2011) noticed that as $\epsilon_t \to 0$ this update will sample from the true posterior. Although the algorithm mixes slower as ϵ gets closer to 0, so setting the stepsize is a trade-off between convergence speed and accuracy. This motivated Welling and Teh (2011) to suggest setting ϵ_t to decrease to 0 as t increases. There is a body of theory that investigates the SGLD approximation to the true posterior (see e.g., Teh et al., 2016; Sato and Nakagawa, 2014; Vollmer et al., 2016). In particular, SGLD is found to converge weakly to the true posterior distribution asymptotically as $\epsilon_t \to 0$. The mean squared error of the algorithm is found to decrease at best with rate $O(T^{-1/3})$. In practice, the algorithm tends to mix poorly when ϵ is set to decrease to 0 (Vollmer et al., 2016), so in our implementation we use a fixed stepsize which has been shown to mix better empirically. Theoretical analysis for this case is provided in Vollmer et al. (2016). The tuning constant ϵ, referred to as the *stepsize* is a required argument in the package. It affects the performance of the algorithm considerably.

4.3.2 Stochastic Gradient Hamiltonian Monte Carlo

The stochastic gradient Hamiltonian Monte Carlo algorithm (SGHMC) (Chen et al., 2014) is similar to SGLD, but instead approximates the HMC algorithm (Neal, 2010). To ensure SGHMC is $O(n)$, the same unbiased estimate to the log posterior gradient is used (4.3.1). SGHMC augments the parameter space with momentum variables ν and samples approximately from a joint distribution $p(\theta, \nu|\mathbf{x})$, whose marginal distribution for θ is the posterior of interest. The algorithm performs the following updates at each iteration

$$\theta_{t+1} = \theta_t + \nu_t,$$

$$\nu_{t+1} = (1-\alpha)\nu_t + \epsilon\nabla_\theta \widehat{\log p(\theta_{t+1}|\mathbf{x})} + \zeta_t,$$

where $\zeta_t \sim \mathcal{N}(0, 2[\alpha - \hat{\beta}_t]\epsilon)$; ϵ and α are tuning constants and $\hat{\beta}_t$ is proportional to an estimate of the Fisher information matrix. In our current implementation, we simply set $\hat{\beta}_t := 0$, as in the experiments of the original paper by Chen et al. (2014). In future implementations, we aim to estimate $\hat{\beta}_t$ using a Fisher scoring estimate similar to Ahn et al. (2012). Often the dynamics are simulated repeatedly L times before the state is stored, at which point ν is resampled. The parameter L can be included in our implementation. The tuning constant ϵ is the stepsize and is a required argument in our implementation, as for SGLD its value affects performance considerably. The constant α tends to be fixed at a small value in practice. As a result, in our implementation it is an optional argument with default value 0.01.

4.3.3 Stochastic Gradient Nosé–Hoover Thermostat

Ding et al. (2014) suggest that the quantity $\hat{\beta}_t$ in SGHMC is difficult to estimate in practice. They appeal to analogues between these proposals and statistical physics in order to suggest a set of updates which do not need this estimation to be made. Once again Ding et al. (2014) augment the space with momentum parameters ν. They replace the tuning constant α with a dynamic parameter α_t known as the *thermostat* parameter. The algorithm performs the following updates at each iteration

$$\theta_{t+1} = \theta_t + \nu_t, \tag{4.3.3}$$

$$\nu_{t+1} = (1 - \alpha_t)\nu_t + \epsilon \nabla_\theta \widehat{\log p(\theta_{t+1}|\mathbf{x})} + \zeta_t, \tag{4.3.4}$$

$$\alpha_{t+1} = \alpha_t + \left[\frac{1}{p}(\nu_{t+1})^\top(\nu_{t+1}) - \epsilon\right]. \tag{4.3.5}$$

Here $\zeta_t \sim \mathcal{N}(0, 2a\epsilon)$; ϵ and a are tuning parameters to be chosen and p is the dimension of θ. The update for α in (4.3.5) requires a vector dot product, it is not obvious how to extend this when θ is higher order than a vector, such as a matrix or tensor. In our implementation, when θ is a matrix or tensor we use the standard inner product in those spaces (Abraham et al., 1988). The tuning constant ϵ is the stepsize and is a required argument in our implementation, as again its value affects performance considerably. The constant a, similarly to α in SGHMC, tends to be fixed at a small value in practice (Ding et al., 2014). As a result, in our implementation it is an optional argument with default value 0.01.

4.3.4 Stochastic Gradient MCMC with Control Variates

A key feature of SGMCMC methods is replacing the log posterior gradient calculation with an unbiased estimate. The unbiased gradient estimate, which can be viewed as a noisy version of the true gradient, can have high variance when estimated using a small minibatch of the data. Increasing the minibatch size will reduce the variance of the gradient estimate, but increase the per iteration computational cost of the SGMCMC algorithm. Recently control variates (Ripley, 2009) have been used to reduce the variance in the gradient estimate of SGMCMC (Dubey et al., 2016; Nagapetyan et al., 2017; Baker et al., 2018). Using these improved gradient estimates have been shown to lead to improvements in the MSE of the algorithm (Dubey et al., 2016), as well as its computational cost (Nagapetyan et al., 2017; Baker et al., 2018).

We implement the formulation of Baker et al. (2018), who replace the gradient estimate $\nabla_\theta \widehat{\log p(\theta|\mathbf{x})}$ with

$$\nabla_\theta \widetilde{\log p(\theta|\mathbf{x})} := \nabla_\theta \log p(\hat{\theta}|\mathbf{x}) + \nabla_\theta \widehat{\log p(\theta|\mathbf{x})} - \nabla_\theta \widehat{\log p(\hat{\theta}|\mathbf{x})}, \tag{4.3.6}$$

where $\hat{\theta}$ is an estimate of the posterior mode. This method requires the burn-in phase of MCMC to be replaced by an optimisation step which finds $\hat{\theta} := \text{argmax}_\theta \log p(\theta|\mathbf{x})$. There is then an $O(N)$ preprocessing step to calculate $\nabla_\theta \log p(\hat{\theta}|\mathbf{x})$, after which the chain can be started from $\hat{\theta}$ resulting in a negligible mixing time. Baker et al. (2018) and Nagapetyan et al. (2017) have shown that there are considerable improvements to the computational cost of SGLD when (4.3.6) is used in place of (4.3.1). In particular they showed that standard SGLD requires setting the minibatch size n to be $O(N)$ for arbitrarily good performance; while using control variates requires an $O(N)$ pre-

processing step, but after that a batch size of $O(1)$ can be used to reach the desired performance. Baker et al. (2018) also showed empirically that this particular formulation can lead to a reduction in the burn-in time compared with standard SGLD and the formulation of (Dubey et al., 2016), which tended to get stuck in local stationary points. This is because in complex scenarios the optimisation step is often faster than the burn-in time of SGMCMC. The package sgmcmc includes control variate versions of all the SGMCMC methods implemented: SGLD, SGHMC and SGNHT.

4.4 Brief TensorFlow Introduction

TensorFlow (TensorFlow Development Team, 2015) is a software library released by Google. The tool was initially designed to easily build deep learning models; but the efficient design and excellent implementation of automatic differentiation (Griewank and Walther, 2008) has made it useful more generally. This package is built on TensorFlow, and while we have tried to make the package as easy as possible to use, the requirement for some knowledge of TensorFlow is unavoidable; especially when declaring the log likelihood and log prior functions, or for high dimensional chains which will not fit into memory. With this in mind, we provide a brief introduction to TensorFlow in this section. This should provide enough knowledge for the rest of the article. A more detailed introduction to TensorFlow for R can be found at Allaire et al. (2016).

Any procedure written in TensorFlow follows three main steps. The first step is to declare all the variables, constants and equations required to run the algorithm. In

the background, these declarations enable TensorFlow to build a graph of the possible operations, and how they are related to other variables, constants and operations. Once everything has been declared, the TensorFlow session is begun and all the variables and operations are initialised. Then the previously declared operations can be run as required; typically these will be sequential and will be run in a for loop.

4.4.1 Declaring TensorFlow Tensors

Everything in TensorFlow, including operations, are represented as a tensor; which is basically a multi-dimensional array. There are a number of ways of creating tensors:

- `tf$constant(value)` – create a constant tensor with the same shape and values as `value`. The input `value` is generally an R array, vector or scalar. The most common use for this in the context of the package is for declaring constant parameters when declaring log likelihood and log prior functions.

- `tf$Variable(value)` – create a tensor with the same shape and values as `value`. Unlike `tf$constant`, this type of tensor can be changed by a declared *operation*. The input `value` is generally an R array, vector or scalar.

- `tf$placeholder(datatype, shape)` – create an empty tensor of type `datatype` and dimensions `shape` which can be fed all or part of a dataset, this is useful when declaring operations which rely on data which can change. When you have storage constraints (see Section 4.5.2) you can use a placeholder to declare test functions that rely on a test dataset. The `datatype` should be a TensorFlow data type, such as `tf$float32`; the `shape` should be an R vector or scalar, such

as `c(100,2)`.

- *operation* – an operation declares an operation on previously declared tensors. These use TensorFlow defined functions, such as those in its math library. This is essentially what you are declaring when coding the `logLik` and `logPrior` functions. The `params` input consists of a list of `tf$Variables`, representing the model parameters to be inferred. The `dataset` input consists of a list of `tf$placeholder` tensors, representing the minibatch of data fed at each iteration. Your job is to declare functions that return an operation on these tensors that define the log likelihood and log prior.

4.4.2 TensorFlow Operations

TensorFlow operations take other tensors as input and manipulate them to reach the desired output. Once the TensorFlow session has begun, these operations can be run as needed, and will use the current values for its input tensors. For example, we could declare a Normal density tensor which manipulates a `tf$Variable` tensor representing the parameters and a `tf$placeholder` tensor representing the current data point. The tensor could then use the `TensorFlow` `tf$contrib$distributions$Normal` object to return a tensor object of the current value for a Normal density given the current parameter value and the data point that's fed to the placeholder. We can break this example down into three steps. First we declare the tensors that we require:

```
library("tensorflow")



# Declare required tensors

loc = tf$Variable(rep(0, 2))

dataPoint = tf$placeholder(tf$float32, c(2))

scaleDiag = tf$constant(c(1, 1))

# Declare density operation tensor

distn = tf$contrib$distributions$MultivariateNormalDiag(loc, scaleDiag)

dens = distn$prob(dataPoint)
```

Here we have declared a `tf$Variable` tensor to hold the location parameter; a `tf$placeholder` tensor which will be fed the current data point; the scale parameter is fixed so we declare this as a `tf$constant` tensor. Next we declare the operation which takes the inputs we just declared and returns the Normal density value. The first line which is assigned to `distn` creates a `MultivariateNormalDiag` object, which is linked to the `loc` and `scaleDiag` tensors. Then `dens` evaluates the density of this distribution. The `dens` variable is now linked to the tensors `dataPoint` and `scaleDiag`, so if it is evaluated it will use the current values of those tensors to calculate the density estimate. Next we begin the TensorFlow session:

```
# Begin TensorFlow session, initialize variables

sess = tf$Session()

sess$run(tf$global_variables_initializer())
```

The two lines we just ran starts the TensorFlow session and initialises all the tensors we just declared. The TensorFlow graph has now been built and no new tensors can now be added. This means that all operations need to be declared before they can be evaluated. Now the session is started we can run the operation `dens` we declared given current values for `dataPoint` and `loc` as follows:

```
# Evaluate density, feeding random data point to placeholder
x = rnorm(2)
out = sess$run(dens, feed_dict = dict( dataPoint = x ) )
print(paste0("Density value for x is ", out))


# Get another random point and repeat!
x = rnorm(2)
out = sess$run(dens, feed_dict = dict( dataPoint = x ) )
print(paste0("Density value for x is ", out))
```

Since `dataPoint` is a `placeholder`, we need to feed it values each time. In the block of code above we feed `dataPoint` a random value simulated from a standard Normal. The `sess$run` expression then evaluates the current Normal density value given `loc` and `dataPoint`.

As mentioned earlier, this is essentially what is happening when you are writing the `logLik` and `logPrior` functions. These functions will be fed a list of `tf$Variable` objects to the `params` input, and a list of `tf$placeholder` objects to the `dataset` input. The output of the function will then be declared as a `TensorFlow` opera-

tion. This allows the gradient to be calculated automatically, while maintaining the efficiencies of TensorFlow.

TensorFlow implements a lot of useful functions to make building these operations easier. For example a number of distributions are implemented at `tf$contrib$distributions`,[3] (e.g., `tf$contrib$distributions$Normal` and `tf$contrib$distributions$Gamma`). TensorFlow also has a comprehensive math library which provides a variety of useful tensor operations.[4] For examples of writing TensorFlow operations see the worked examples in Section 4.5 or the sgmcmc vignettes.

4.5 Package Structure and Implementation

The package has 6 main functions. The first three: `sgld`, `sghmc` and `sgnht` will implement SGLD, SGHMC and SGNHT, respectively. The other three: `sgldcv`, `sghmccv` and `sgnhtcv` implement the control variate versions of SGLD, SGHMC and SGNHT, respectively. All of these are built on the TensorFlow library for R, which enables gradients to be automatically calculated and efficient computations to be performed in high dimensions. The usage for these functions is very similar, with the only differences in input being tuning parameters. These main functions are outlined in Table 4.5.1

[3]For full API details see `https://www.tensorflow.org/api_docs/python/tf/contrib/distributions` though note this is for Python, so the `.` object notation needs to be replaced by `$`, for example `tf.contrib.distributions.Normal` would be replaced by `tf$contrib$distributions$Normal`.

[4]See `https://www.tensorflow.org/api_guides/python/math_ops`.

Function	Algorithm
`sgld`	Stochastic gradient Langevin dynamics
`sghmc`	Stochastic gradient Hamiltonian Monte Carlo
`sgnht`	Stochastic gradient Nosé-Hoover thermostat
`sgldcv`	Stochastic gradient Langevin dynamics with control variates
`sghmccv`	Stochastic gradient Hamiltonian Monte Carlo with control variates
`sgnhtcv`	Stochastic gradient Nosé-Hoover thermostat with control variates

Table 4.5.1: Outline of 6 main functions implemented in sgmcmc.

The functions `sgld`, `sghmc` and `sgnht` have the same required inputs: `logLik`, `dataset`, `params` and `stepsize`. These determine respectively: the log likelihood function for the model; the data for the model; the parameters of the model; and the stepsize tuning constants for the SGMCMC algorithm. The input `logLik` is a function taking `dataset` and `params` as input, while the rest are defined as lists. Using lists in this way provides a lot of flexibility: allowing multiple parameters to be defined; use multiple datasets; and use different stepsizes for each parameter, which is vital if the scalings are different. It also allows users to easily reference parameters and datasets in the `logLik` function by simply referring to the relevant names in the list.

The functions also have a couple of optional parameters that are particularly important, `logPrior` and `minibatchSize`. These respectively define the log prior for the model; and the minibatch size, as it was defined in Section 4.3. By default, the `logPrior` is set to an uninformative uniform prior, which is fine to use for quick checks but will need to be set properly for more complex models. The `logPrior` is a

Function inputs	Definition
`logLik`	Log-likelihood function taking `dataset` and `params` as inputs
`dataset`	R list containing data
`params`	R list containing model parameters
`stepsize`	R list of stepsizes for each parameter
`optStepsize`	R numeric stepsize for control variate optimisation step
`logPrior`	Function of the parameters (on the log-scale); default $p(\theta) \propto 1$
`minibatchSize`	Size of minibatch per iteration as integer or proportion; default 0.01.
`nIters`	Number of MCMC iterations; default is 10,000.

Table 4.5.2: Outline of the key arguments required by the functions in Table 4.5.1.

function similar to `logLik`, but only takes `params` as input. The `minibatchSize` is a numeric, and can either be a proportion of the dataset size, if it is set between 0 and 1, or an integer. The default value is 0.01, meaning that 1% of the full dataset is used at each iteration.

The control variate functions have the same inputs as the non-control variate functions, with one more required input. The `optStepsize` input is a numeric that specifies the stepsize for the initial optimisation step to find the $\hat{\theta}$ maximising the posterior as defined in Section 4.3.4. For a full outline of the key inputs, see Table 4.5.2.

Often large datasets and high dimensional problems go hand in hand. In these high dimensional settings storing the full MCMC chain in memory can become an

issue. For this situation we provide functionality to run the chain one iteration at a time in a user defined loop. This enables the user to deal with the output at each iteration how they see fit. For example, they may wish to calculate a test function on the output to reduce the dimensionality of the chain; or they might calculate the required Monte Carlo estimates on the fly. We aim to extend this functionality to enable the user to define their own Gibbs updates alongside the SGMCMC procedure. This functionality is more involved, and requires more knowledge of TensorFlow, so we leave the details to the example in Section 4.5.2.

For the rest of this section we go into more detail about the usage of the functions using a worked example. The package is used to infer the bias and coefficient parameters in a logistic regression model. Section 4.5.1 demonstrates standard usage by performing inference on the model using the `sgld` and `sgldcv` functions. Section 4.5.2 demonstrates usage in problems where the full MCMC chain cannot be fit into memory. The same logistic regression model is used throughout.

4.5.1 Example Usage

In this example we use the functions `sgld` and `sgldcv` to infer the bias (or intercept) and coefficients of a logistic regression model. The `sgldcv` case is also available as a vignette. Suppose we have data $\mathbf{x}_1, \dots, \mathbf{x}_N$ of dimension d taking values in $\mathbb{R}^d$; and response variables $y_1, \dots, y_N$ taking values in $\{0, 1\}$. Then a logistic regression model with coefficients β and bias β_0 will have likelihood

$$p(\mathbf{X}, \mathbf{y}|\beta, \beta_0) = \prod_{i=1}^{N} \left[\frac{1}{1 + e^{-\beta_0 - \mathbf{x}_i\beta}}\right]^{y_i} \left[1 - \frac{1}{1 + e^{-\beta_0 - \mathbf{x}_i\beta}}\right]^{1-y_i} \tag{4.5.1}$$

We will use the `covertype` dataset (Blackard and Dean, 1999) which can be downloaded and loaded using the sgmcmc function `getDataset`, which downloads example datasets for the package. The `covertype` dataset uses mapping data to predict the type of forest cover. Our particular dataset is taken from the LIBSVM library (Chang and Lin, 2011), which converts the data to a binary problem, rather than multiclass. The dataset consists of a matrix of dimension 581012×55. The first column contains the labels $\mathbf{y}$, taking values in $\{0, 1\}$. The remaining columns are the explanatory variables $\mathbf{X}$, which have been scaled to take values in $[0, 1]$.

```
library("sgmcmc")

covertype = getDataset("covertype")

# Split the data into predictors and response

X = covertype[,2:ncol(covertype)]

y = covertype[,1]

# Create dataset list for input

dataset = list( "X" = X, "y" = y )
```

In the last line we defined the dataset as a list object which will be input to the relevant sgmcmc function. The list names can be arbitrary, but must be consistent with the variables declared in the `logLik` function (see below).

When accessing the data, it is assumed that observations are split along the first axis. In other words, `dataset$X` is a 2-dimensional matrix, and we assume that observation $\mathbf{x}_i$ is accessed at `dataset$X[i,]`. Similarly, suppose `Z` was a 3-dimensional array, we would assume that observation i would be accessed at `Z[i,,]`. Parameters

are declared in a similar way, except the list entries are the desired parameter starting points. There are two parameters for this model, the bias β_0 and the coefficients β, which can be arbitrarily initialised to 0.

```
# Get the dimension of X, needed to set shape of params
d = ncol(dataset$X)
params = list( "bias" = 0, "beta" = matrix( rep( 0, d ), nrow = d ) )
```

The log likelihood is specified as a function of the `dataset` and `params`, which are lists with the same names we declared earlier. The only difference is that the objects inside the lists will have automatically been converted to TensorFlow objects. The `dataset` list will contain TensorFlow placeholders. The `params` list will contain TensorFlow variables. The `logLik` function should be a function that takes these lists as input and returns the log likelihood value given the current parameters and data. This is done using TensorFlow operations, as this allows the gradient to be automatically calculated.

For the logistic regression model (4.5.1), the log likelihood is

$$\log p(\mathbf{X}, \mathbf{y}|\beta, \beta_0) = \sum_{i=1}^{N} y_i \log y_{\text{est}} + (1 - y_i) \log(1 - y_{\text{est}}),$$

where $y_{\text{est}} = [1 + e^{-\beta_0 - \mathbf{x}_i\beta}]^{-1}$, which coded as a `logLik` function is defined as follows

```
logLik = function(params, dataset) {
    yEst = 1 / (1 + tf$exp( - tf$squeeze(params$bias +
             tf$matmul(dataset$X, params$beta))))
    logLik = tf$reduce_sum(dataset$y * tf$log(yEst) +
```

```
            (1 - dataset$y) * tf$log(1 - yEst))

    return(logLik)

}
```

For more information about the usage of these TensorFlow functions, please see the TensorFlow documentation.[5]

Next, the log prior density, where we assume each β_j, for $j = 0, \ldots, d$, has an independent Laplace prior with location 0 and scale 1, so $\log p(\beta) \propto -\sum_{j=0}^{d} |\beta_j|$. Similar to `logLik`, this is defined as a function, but with only `params` as input

```
logPrior = function(params) {

    logPrior = - (tf$reduce_sum(tf$abs(params$beta)) +

            tf$reduce_sum(tf$abs(params$bias)))

    return(logPrior)

}
```

The final input that needs to be set is the `stepsize` for tuning the methods, this can be set as a list

```
stepsize = list("beta" = 2e-5, "bias" = 2e-5)
```

Setting the same stepsize for all parameters is done as `stepsize = 2e-5`. This shorthand can also be used for any of the optional tuning parameters which need

[5] Documentation for TensorFlow for R available at `https://tensorflow.rstudio.com/tensorflow/`

to specified as lists. The stepsize parameter will generally require a bit of tuning in order to get good performance, for this we recommend using cross validation (see e.g., Friedman et al., 2001, Chapter 7).

Everything is now ready to run a standard SGLD algorithm, with `minibatchSize` set to 500. To keep things reproducible we'll set the seed to 13.

```
output = sgld( logLik, dataset, params, stepsize,
        logPrior = logPrior, minibatchSize = 500, nIters = 10000, seed = 13 )
```

The output of the function is also a list with the same names as the `params` list. Suppose a given parameter has shape $(d_1, \ldots, d_l)$, then the output will be an array of shape $(\text{nIters}, d_1, \ldots, d_l)$. So in our case, `output$beta[i,,]` is the i^{th} MCMC sample from the parameter β; and `dim(output$beta)` is `c(10000, 54, 1)`.

In order to run a control variate algorithm such as `sgldcv` we need one additional argument, which is the stepsize for the initial optimisation step. The optimisation uses the TensorFlow `GradientDescentOptimizer`. The stepsize should be quite similar to the stepsize for SGLD, though is often slightly larger. First, so that we can measure the performance of the chain, we shall set a seed and randomly remove some observations from the full `dataset` to form a `testset`. We also set a short burn-in of 1000.

```
set.seed(13)
testSample = sample(nrow(dataset$X), 10^4)
testset = list( "X" = dataset$X[testSample,], "y" = dataset$y[testSample] )
```

```
dataset = list( "X" = dataset$X[-testSample,], "y" = dataset$y[-testSample] )

output = sgldcv( logLik, dataset, params, 5e-6, 5e-6,

        logPrior = logPrior, minibatchSize = 500, nIters = 11000, seed = 13 )

# Remove burn-in

output$beta = output$beta[-c(1:1000),,]

output$bias = output$bias[-c(1:1000)]
```

A common performance measure for a classifier is the *log loss*. Given an observation with data $\mathbf{x}_i$ and response y_i, logistic regression predicts that $y_i = 1$ with probability

$$\pi(\mathbf{x}_i, \beta, \beta_0) = \frac{1}{1 + e^{-\beta_0 - \mathbf{x}_i\beta}}.$$

Given a test set T of data response pairs $(\mathbf{x}, y)$, the log loss $s(\cdot)$, of a binary chain, is defined as

$$s(\beta, \beta_0, T) = -\frac{1}{|T|} \sum_{(\mathbf{x},y)\in T} \left[y \log \pi(\mathbf{x}, \beta, \beta_0) + (1 - y) \log(1 - \pi(\mathbf{x}, \beta, \beta_0)) \right]. \quad (4.5.2)$$

To check convergence of `sgldcv` we'll plot the log loss every 10 iterations, using the `testset` we removed earlier. Results are given in Figure 4.5.1. The plot shows the `sgldcv` algorithm converging to a stationary after a short burn-in period. The burn-in period is so short due to the initial optimisation step.

```
iterations = seq(from = 1, to = 10^4, by = 10)

logLoss = rep(0, length(iterations))

# Calculate log loss every 10 iterations

for ( iter in 1:length(iterations) ) {
```

```
    j = iterations[iter]
    beta0_j = output$bias[j]
    beta_j = output$beta[j,]
    for ( i in 1:length(testset$y) ) {
        piCurr = 1 / (1 + exp(- beta0_j - sum(testset$X[i,] * beta_j)))
        y_i = testset$y[i]
        logLossCurr = - ( (y_i * log(piCurr) + (1 - y_i) * log(1 - piCurr)) )
        logLoss[iter] = logLoss[iter] + 1 / length(testset$y) * logLossCurr
    }
}
# Plot output
plotFrame = data.frame("Iteration" = iterations, "logLoss" = logLoss)
ggplot(plotFrame, aes(x = Iteration, y = logLoss)) +
    geom_line(color = "maroon") +
    ylab("Log loss of test set")
```

4.5.2 Example Usage: Storage Constraints

Often large datasets and high dimensionality go hand in hand. Sometimes the dimensionality is so high that storage of the full MCMC chain in memory becomes an issue. There are a number of ways around this, including: calculating estimates of the desired posterior quantity on the fly; reducing the dimensionality of the chain using a test function; or just periodically saving a the chain to the hard disk and starting

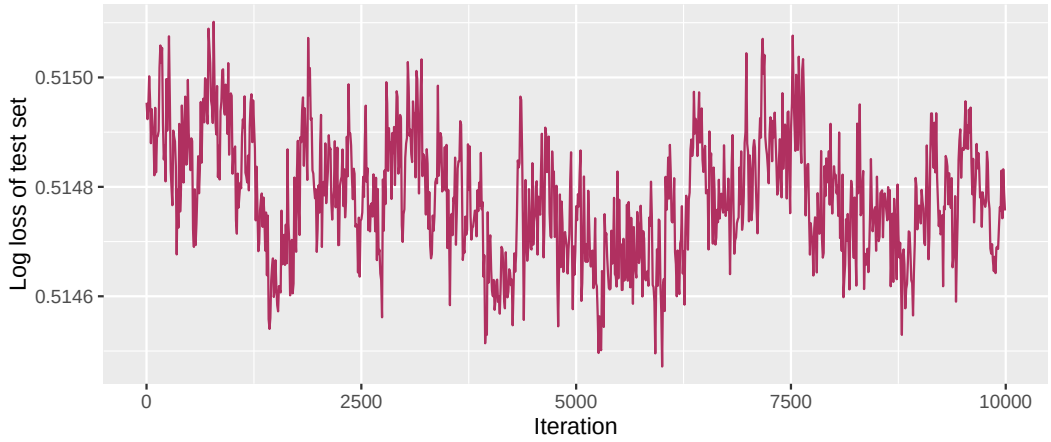


Figure 4.5.1: Log loss on a test set for parameters simulated using the `sgldcv` algorithm after 1000 iterations of burn-in. Logistic regression problem with the covertype dataset.

from scratch. To deal with high storage costs sgmcmc provides functionality to run SGMCMC algorithms step by step. This allows users to deal with the output as they see fit at each iteration.

In this section, we detail how to run SGMCMC chains step by step. To do this requires more knowledge of TensorFlow, including using TensorFlow sessions and building custom placeholders and tensors. For more details see the TensorFlow for R documentation (Allaire et al., 2016). The step by step procedure works similarly to a standard TensorFlow procedure: TensorFlow variables, tensors and placeholders are declared; then the TensorFlow session is started and all the required tensors are initialised; finally the SGMCMC algorithm is run step by step in a user defined loop, and the user evaluates tensors as required.

To demonstrate this concept we keep things simple and use the logistic regression example introduced in the previous section. While this example can fit into memory, it

allows us to demonstrate the concepts without getting bogged down in a complicated model. For a more realistic example, where the output does not fit into memory, see the Bayesian neural network model in Section 4.6.3.

We start by assuming the `dataset`, `params`, `logLik`, `logPrior` and `stepsize` objects have been created in exactly the same way as in the previous example (Section 4.5.1). Now suppose we want to make inference using stochastic gradient Langevin dynamics (SGLD), but we want to run it step by step. The first step is to initialise an `sgld` object using the function `sgldSetup`. For every function in Table 4.5.1 there is a corresponding `*Setup` function, such as `sghmccvSetup` or `sgnhtSetup`. This function will create all the TensorFlow objects required, as well as declare the dynamics of the SGMCMC algorithm. For our example we can run the following

```
sgld = sgldSetup(logLik, dataset, params, stepsize,
        logPrior = logPrior, minibatchSize = 500, seed = 13)
```

This `sgld` object is a type of `sgmcmc` object, it is an R `S3` object, which is essentially a list with a number of entries. The most important of these entries for building SGMCMC algorithms is called `params`, which holds a list, with the same names as in the `params` that were fed to `sgldSetup`, but this list contains `tf$Variable` objects. This is how the tensors are accessed which hold the current parameter values in the chain. For more details on the attributes of these objects, see the documentation for `sgldSetup`, `sgldcvSetup`, etc.

Now that we have created the `sgld` object, we want to initialise the TensorFlow variables and the `sgmcmc` algorithm chosen. For a standard algorithm, this will ini-

tialise the TensorFlow graph and all the tensors that were created. For an algorithm with control variates (e.g., `sgldcv`), this will also find the $\hat{\theta}$ estimates of the parameters and calculate the full log posterior gradient at that point; as detailed in Section 4.3.4. The function used to do this is `initSess`,

```
sess = initSess(sgld)
```

The `sess` returned by `initSess` is the current TensorFlow session, which is needed to run the SGMCMC algorithm of choice, and to access any of the tensors needed, such as `sgld$params`.

Now we have everything to run an SGLD algorithm step by step as follows

```
for (i in 1:10^4) {

    sgmcmcStep(sgld, sess)

    currentState = getParams(sgld, sess)

}
```

Here the function `sgmcmcStep` will update `sgld$params` using a single update of SGLD, or whichever SGMCMC algorithm is given. The function `getParams` will return a list of the current parameters as R objects rather than as tensors.

This simple example of running SGLD step by step only stores the most recent value in the chain, which is useless for a Monte Carlo method. Also, for large scale examples, it is often useful to reduce the dimension of the chain by calculating some test function $g(\cdot)$ of θ at each iteration rather than the parameters themselves. This example will demonstrate how to store a test function at each iteration, as well as

calculating the estimated posterior mean on the fly. We assume that a new R session has been started and the `sgld` object has just been created using `sgldSetup` with the same properties as in the example above. We assume that no TensorFlow session has been created (i.e., `initSess` has not been run yet).

Before the TensorFlow session has been declared, the user is able to create their own custom tensors. This is useful, as test functions can be declared beforehand using the `sgld$params` variables, which allows the test functions to be quickly calculated by just evaluating the tensors in the current session. The test function used here is once again the log loss of a test set, as defined in (4.5.2).

Suppose we input `sgld$params` and the `testset` T to the `logLik` function. Then the log loss is actually $-\frac{1}{|T|}$ times this value. This means we can easily create a tensor that calculates the log loss by creating a list of placeholders that hold the test set, then using the `logLik` function with the `testset` list and `sgld$params` as input. We can do this as follows

```
testPlaceholder = list()
testPlaceholder[["X"]] = tf$placeholder(tf$float32, dim(testset[["X"]]))
testPlaceholder[["y"]] = tf$placeholder(tf$float32, dim(testset[["y"]]))
testSize = as.double(nrow(testset[["X"]]))
logLoss = - logLik(sgld$params, testPlaceholder) / testSize
```

This placeholder is then fed the full `testset` each time the log loss is calculated. Now we will declare the TensorFlow session, and run the chain step by step. At each iteration we will calculate the current Monte Carlo estimate of the parameters. The

log loss will be stored every 100 iterations. We omit a plot of the log loss as it is similar to Figure 4.5.1.

```
sess = initSess(sgld)

# Fill a feed dict with full test set (used to calculate log loss)

feedDict = dict()

feedDict[[testPlaceholder[["X"]]]] = testset[["X"]]

feedDict[[testPlaceholder[["y"]]]] = testset[["y"]]

# Burn-in chain

message("Burning-in chain...")

message("iteration\tlog loss")

for (i in 1:10^4) {

# Print progress

    if (i %% 100 == 0) {

        progress = sess$run(logLoss, feed_dict = feedDict)

        message(paste0(i, "\t", progress))

    }

    sgmcmcStep(sgld, sess)

}

# Initialise posterior mean estimate using value after burn-in

postMean = getParams(sgld, sess)

logLossOut = rep(0, 10^4 / 100)

# Run chain
```

```
message("Running SGMCMC...")

for (i in 1:10^4) {

    sgmcmcStep(sgld, sess)

    # Update posterior mean estimate

    currentState = getParams(sgld, sess)

    for (paramName in names(postMean)) {

        postMean[[paramName]] = (postMean[[paramName]] * i +

                currentState[[paramName]]) / (i + 1)

    }

    # Print and store log loss

    if (i %% 100 == 0) {

        logLossOut[i/100] = sess$run(logLoss, feed_dict = feedDict)

        message(paste0(i, "\t", logLossOut[i/100]))

    }

}
```

4.6 Simulations

In this section we demonstrate the algorithms and performance by simulating from a variety of models using all the implemented methods and commenting on the performance of each. These simulations are reproducible and available in the supplementary material and on Github.[6] For more usage tutorials similar to Sections 4.5.1 and 4.5.2,

[6] https://github.com/jbaker92/sgmcmc-simulations

please see the vignettes on the package website.[7]

4.6.1 Gaussian Mixture

In this model we assume our dataset $x_1, \ldots, x_N$ is drawn i.i.d from

$$X_i \,|\, \theta_1, \theta_2 \sim \frac{1}{2}\mathcal{N}(\theta_1, \mathrm{I}_2) + \frac{1}{2}\mathcal{N}(\theta_2, \mathrm{I}_2), \quad i = 1, \ldots, N; \tag{4.6.1}$$

where θ_1, θ_2 are parameters to be inferred and I_2 is the 2×2 identity matrix. We assume the prior $\theta_1, \theta_2 \sim \mathcal{N}(0, 10\mathrm{I}_2)$. To generate the synthetic dataset, we simulate 10^3 observations from $\frac{1}{2}\mathcal{N}\left([0,0]^\top, \mathrm{I}_2\right) + \frac{1}{2}\mathcal{N}\left([0.1, 0.1]^\top, \mathrm{I}_2\right)$. While this is a small number of observations, it allows us to compare the results to a full Hamiltonian Monte Carlo (HMC) scheme using the R implementation of Stan (Carpenter et al., 2017). The full HMC scheme should sample from close to the true posterior distribution, so acts as a good surrogate for the truth. We compare each sgmcmc algorithm implemented to the HMC sample to compare performance. Larger scale examples are given in Sections 4.6.2 and 4.6.3. We ran all methods for 10^4 iterations, except SGHMC, since the computational cost is greater for this method due to the trajectory parameter L. We ran SGHMC for 2,000 iterations, using default trajectory $L = 5$, as this ensures the overall computational cost of the method is similar to the other methods. We used a burn-in step of 10^4 iterations, except for the control variate methods, where we used 10^4 iterations in the initial optimisation step, with no burn-in. Again this ensures comparable computational cost across different methods.

The `logLik` and `logPrior` functions used for this model are as follows

[7] `https://stor-i.github.io/sgmcmc`

```
logLik = function( params, dataset ) {
    # Declare Sigma (assumed known)
    SigmaDiag = c(1, 1)
    # Declare distribution of each component
    component1 = tf$contrib$distributions$MultivariateNormalDiag(
            params$theta1, SigmaDiag )
    component2 = tf$contrib$distributions$MultivariateNormalDiag(
            params$theta2, SigmaDiag )
    # Declare allocation probabilities of each component
    probs = tf$contrib$distributions$Categorical(c(0.5,0.5))
    # Declare full mixture distribution given components and probs
    distn = tf$contrib$distributions$Mixture(
            probs, list(component1, component2))
    # Declare log likelihood
    logLik = tf$reduce_sum( distn$log_prob(dataset$X) )
    return( logLik )
}


logPrior = function( params ) {
    # Declare hyperparameters mu0 and Sigma0
    mu0 = c( 0, 0 )
    Sigma0Diag = c(10, 10)
```

```
    # Declare prior distribution
    priorDistn = tf$contrib$distributions$MultivariateNormalDiag(
            mu0, Sigma0Diag )
    # Declare log prior density and return
    logPrior = priorDistn$log_prob( params$theta1 ) +
            priorDistn$log_prob( params$theta2 )
    return( logPrior )
}
```

The following list determines the stepsizes used for each method, the `optStepsize` used for control variate methods was `5e-5`.

```
stepsizeList = list("sgld" = 5e-3, "sghmc" = 5e-4, "sgnht" = 3e-4,
        "sgldcv" = 1e-2, "sghmccv" = 1.5e-3, "sgnhtcv" = 3e-3)
```

We set the seed to be 2 using the optional `seed` argument and use a minibatch size of 100. We also used a seed of 2 when generating the data (see the supplementary material for full details). Starting points were sampled from a standard Normal.

The results are plotted in Figure 4.6.1. The black contours represent the best guess at the true posterior, which was found using the standard HMC procedure in Stan. The coloured contours that overlay the black contours are the approximations of each of the SGMCMC methods implemented by sgmcmc. This allows us to compare the SGMCMC estimates with the 'truth' by eye.

In the simulation, we obtain two chains, one approximating θ_1 and the other

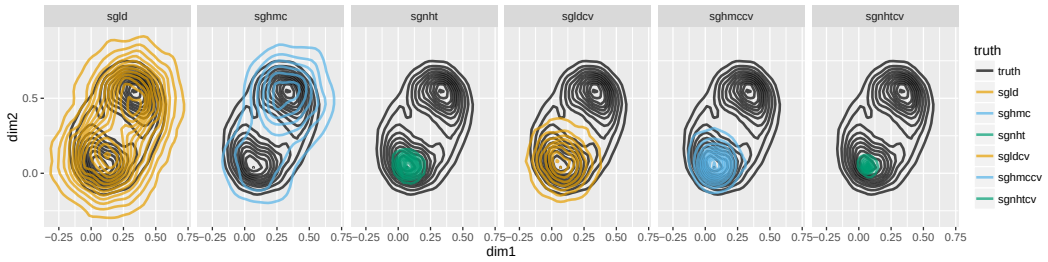


Figure 4.6.1: Plots of the approximate posterior for θ_1 simulated using each of the methods implemented by sgmcmc, compared with a full HMC run, treated as the truth, for the Gaussian mixture model (4.6.1).

approximating θ_2. In order to examine how well the methods explore both modes, we take just θ_1 and compare this to the HMC run for θ_1. The results are quite variable, and it demonstrates a point nicely: there seems to be a trade-off between predictive accuracy and exploration. Many methods have demonstrated good performance using predictive accuracy; where a test set is removed from the full dataset to assess how well the fitted model performs on the test set. This is a useful technique for complex models, which are high dimensional and have a large number of data points, as they cannot be plotted, and an MCMC run to act as the 'truth' cannot be fitted.

However, this example shows that it does not give the full picture. A lot of the methods which show improved predictive performance (e.g., control variate methods and especially `sgnht`) over `sgld` appear here to perform worse at exploring the full space. In this example, `sgld` performs the best at exploring both components, though it over-estimates posterior uncertainty. The algorithm `sghmc` also explores both components but somewhat unevenly. We find that `sgnht`, while being shown to have better predictive performance in the original paper (Ding et al., 2014), does not do

nearly as well as the other algorithms at exploring the space and appears to collapse to the posterior mode. The control variate methods, shown in the following sections, and in Baker et al. (2018), appear to have better predictive performance than `sgld`, but do not explore both components either. For example, `sgldcv` explores the space the best but over-estimates uncertainty of the first component, since it relies on SGLD updates which also overestimates uncertainty. In contrast, `sgnhtcv` collapses to a posterior mode since it relies on the SGNHT updates which also collapse.

4.6.2 Bayesian Logistic Regression

In this section, we apply all the methods to the logistic regression example in Section 4.5.1. We compare the performance of the methods by calculating the log loss of a test set every 10 iterations, again as detailed in Section 4.5.1. The standard methods (`sgld`, `sghmc`, `sgnht`) were run for 10^4 iterations with an additional 10^4 iterations of burn-in; except for `sghmc` which has $5\times$ the computational cost, so is ran for 2,000 iterations with 2,000 iterations of burn-in. The control variate methods were run for 10^4 iterations with an additional 10^4 iterations for the initial optimisation step, and no burn-in; again except for `sghmccv` which was run for 2,000 iterations. This means that all the methods should be somewhat comparable in terms of computation time.

The following list determines the stepsizes used for each method, the `optStepsize` used was `1e-6`.

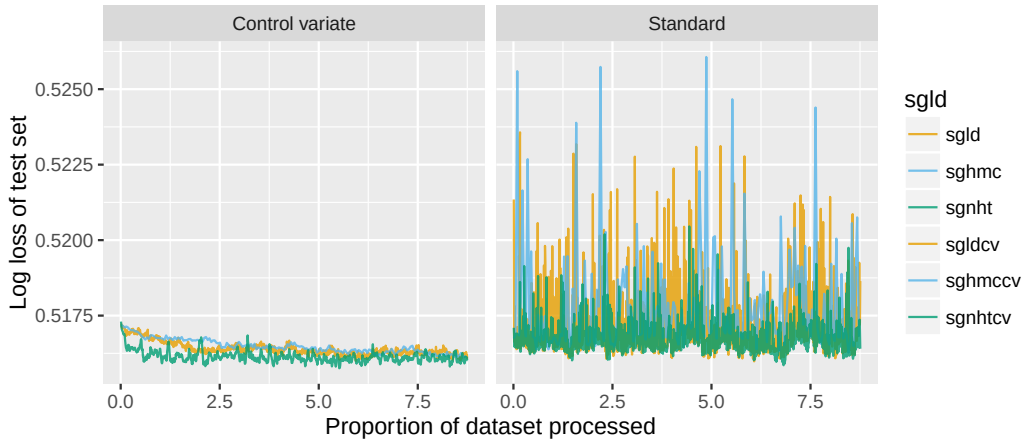


Figure 4.6.2: Plots of the log loss of a test set for β_0 and β simulated using each of the methods implemented by sgmcmc. Logistic regression problem with the covertype dataset.

```
stepsizes = list("sgld" = 5e-6, "sghmc" = 1e-7, "sgnht" = 1e-7, "sgldcv" = 1e-5,
        "sghmccv" = 1e-6, "sgnhtcv" = 5e-7)
```

We set the seed to be 1 for each of the simulations, and when generating the test data (see the supplementary material for reproducible code) and use a minibatch size of 500. Starting points are sampled from a standard Normal.

Results are plotted in Figure 4.6.2. All of the algorithms show decent performance. Methods which use control variates have significantly better predictive performance; and result in chains with lower variance. `sghmc` has lower variance than `sgld` and `sgnht`, though this could be related to the high computational cost. One might envisage setting a lower trajectory L would result in a chain with higher variance. `sgldcv` takes longer to burn-in than the other control variate methods. The algorithm `sgld` has the highest variance by far; this could be related to our discussion in Section

4.6.1 on exploration versus accuracy.

4.6.3 Bayesian Neural Network

In this simulation we demonstrate a very high dimensional chain. This gives a more realistic example of when we would want to run the chain step by step. The model is a two layer Bayesian neural network which is fit to the MNIST dataset (LeCun and Cortes, 2010). The MNIST dataset consists of 28×28 pixel images of handwritten digits from zero to nine. The images are flattened to be a vector of length 784. The dataset is available as a standard dataset from the TensorFlow library, with a matrix of 55,000 training vectors and 10,000 test vectors, each with their corresponding labels. The dataset can be constructed in a similar way to the logistic regression example of Section 4.5.1, using the standard dataset in the package `mnist`.

```
library("sgmcmc")

mnist = getDataset("mnist")

dataset = list("X" = mnist$train$images, "y" = mnist$train$labels)

testset = list("X" = mnist$test$images, "y" = mnist$test$labels)
```

We build the same neural network model as in the original SGHMC paper by Chen et al. (2014). Suppose Y_i takes values in $\{0, \dots, 9\}$, so is the output label of a digit, and $\mathbf{x}_i$ is the input vector, with $\mathbf{X}$ the full $N \times 784$ dataset, where N is the number

of observations. The model is then as follows

$$Y_i \,|\, \theta, \mathbf{x}_i \sim \text{Categorical}(\beta(\theta, \mathbf{x}_i)), \tag{4.6.2}$$

$$\beta(\theta, \mathbf{x}_i) = \sigma\left(\sigma\left(\mathbf{x}_i^\top B + b\right) A + a\right). \tag{4.6.3}$$

Here A, B, a, b are parameters to be inferred with $\theta = (A, B, a, b)$; $\sigma(\cdot)$ is the softmax function (a generalisation of the logistic link function). A, B, a and b are matrices with dimensions: 100×10, 784×100, 1×10 and 1×100 respectively. Each element of these parameters is assigned a Normal prior

$$A_{kl}|\lambda_A \sim \mathcal{N}(0, \lambda_A^{-1}), \quad B_{jk}|\lambda_B \sim \mathcal{N}(0, \lambda_B^{-1}),$$

$$a_l|\lambda_a \sim \mathcal{N}(0, \lambda_a^{-1}), \quad b_k|\lambda_b \sim \mathcal{N}(0, \lambda_b^{-1}),$$

$$j = 1, \ldots, 784; \quad k = 1, \ldots, 100; \quad l = 1, \ldots, 10;$$

where λ_A, λ_B, λ_a and λ_b are hyperparameters. Finally, we assume

$$\lambda_A, \lambda_B, \lambda_a, \lambda_b \sim \text{Gamma}(1, 1).$$

The model contains a large number of high dimensional parameters, and unless there is sufficient RAM available, a standard chain of length 10^4 will not fit into memory. First, we shall create the `params` dictionary, and then code the `logLik` and `logPrior` functions. We can sample the initial λ parameters from a standard Gamma distribution, and the remaining parameters from a standard Normal as follows

```
# Sample initial weights from standard Normal

d = ncol(dataset$X)

params = list()
```

```
params$A = matrix(rnorm(10*100), ncol = 10)

params$B = matrix(rnorm(d*100), ncol = 100)

# Sample initial bias parameters from standard Normal

params$a = rnorm(10)

params$b = rnorm(100)

# Sample initial precision parameters from standard Gamma

params$lambdaA = rgamma(1, 1)

params$lambdaB = rgamma(1, 1)

params$lambdaa = rgamma(1, 1)

params$lambdab = rgamma(1, 1)
```

```
logLik = function(params, dataset) {

    # Calculate estimated probabilities

    beta = tf$nn$softmax(tf$matmul(dataset$X, params$B) + params$b)

    beta = tf$nn$softmax(tf$matmul(beta, params$A) + params$a)

    # Calculate log likelihood of categorical distribution with prob. beta

    logLik = tf$reduce_sum(dataset$y * tf$log(beta))

    return(logLik)

}
```

```
logPrior = function(params) {

    distLambda = tf$contrib$distributions$Gamma(1, 1)

    distA = tf$contrib$distributions$Normal(0, tf$rsqrt(params$lambdaA))
```

```
    logPriorA = tf$reduce_sum(distA$log_prob(params$A)) +
           distLambda$log_prob(params$lambdaA)
    distB = tf$contrib$distributions$Normal(0, tf$rsqrt(params$lambdaB))
    logPriorB = tf$reduce_sum(distB$log_prob(params$B)) +
           distLambda$log_prob(params$lambdaB)
    dista = tf$contrib$distributions$Normal(0, tf$rsqrt(params$lambdaa))
    logPriora = tf$reduce_sum(dista$log_prob(params$a)) +
           distLambda$log_prob(params$lambdaa)
    distb = tf$contrib$distributions$Normal(0, tf$rsqrt(params$lambdab))
    logPriorb = tf$reduce_sum(distb$log_prob(params$b)) +
           distLambda$log_prob(params$lambdab)
    logPrior = logPriorA + logPriorB + logPriora + logPriorb
    return(logPrior)
}
```

Similar to Section 4.5.2, we use the log loss as a test function. This time though it is necessary to update the definition, as the logistic regression example was a binary problem whereas now we have a multiclass problem. Given a test set T of pairs $(\mathbf{x}, y)$, now y can take values in $\{0, \dots, K\}$, rather than just binary values. To account for this we redefine the definition of log loss to be

$$s(\theta, T) = -\frac{1}{|T|} \sum_{\mathbf{x},y \in T} \sum_{k=1}^{K} \mathbf{1}_{y=k} \log \beta_k(\theta, \mathbf{x}),$$

where $\mathbf{1}_A$ is the indicator function, and $\beta_k(\theta, \mathbf{x})$ is the k^{th} element of $\beta(\theta, \mathbf{x})$ as defined in (4.6.3).

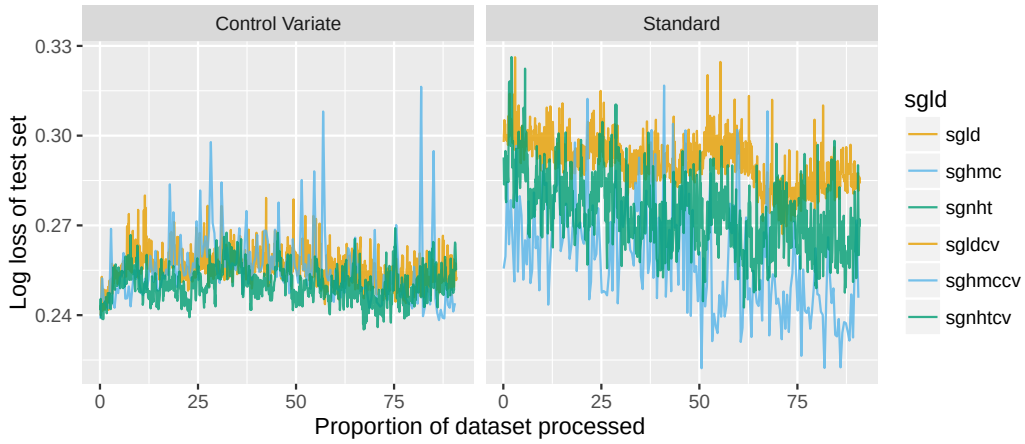


Figure 4.6.3: Plots of the log loss of a test set for θ simulated using each of the methods implemented by sgmcmc. Bayesian neural network model with the MNIST dataset.

As in Section 4.5.2, the log loss is simply $-\frac{1}{|T|}$ times the `logLik` function, if we feed it the `testset` rather than the `dataset`. This means the `logLoss` tensor can be declared in a similar way to Section 4.5.2

```
testPlaceholder = list()

testPlaceholder[["X"]] = tf$placeholder(tf$float32, dim(testset[["X"]]))

testPlaceholder[["y"]] = tf$placeholder(tf$float32, dim(testset[["y"]]))

testSize = as.double(nrow(testset[["X"]]))

logLoss = - logLik(sgld$params, testPlaceholder) / testSize
```

We can run the chain in exactly the same way as Section 4.5.2, and so omit the code for this. We ran 10^4 iterations of each of the algorithms in Table 4.5.1, calculating the log loss for each every 10 iterations. The standard algorithms have 10^4 iterations of burn-in while the control variate algorithms have no burn-in, but 10^4 iterations in

the initial optimisation step. Note that due to the trajectory parameter L, `sghmc` and `sghmccv` will have 5 times the computational cost of the other algorithms. Therefore, we ran these algorithms for 2,000 iterations instead, to make the computational cost comparable. We used the following list of stepsizes

```
list("sgld" = 1e-4, "sghmc" = 1e-5, "sgnht" = 5e-6, "sgldcv" = 5e-5,
        "sghmccv" = 1e-5, "sgnhtcv" = 5e-7)
```

Generally these are the stepsizes which produce the smallest log loss; except when these chains did not seem to explore the space fully, in which case we increased the stepsize slightly. We set the seed to be 1 for each of the simulations, and when generating the test data (see the supplementary material for reproducible code).

The results are plotted in Figure 4.6.3. Again we see improvements in the predictive performance of the control variate methods. Among the standard methods, `sghmc` and `sgnht` have the best predictive performance; which is to be expected given the apparent trade-off between accuracy and exploration.

4.7 Discussion

We presented the R package sgmcmc, which enables Bayesian inference with large datasets using stochastic gradient Markov chain Monte Carlo. The package only requires the user to specify the log likelihood and log prior functions; and any differentiation required can be performed automatically. The package is based on TensorFlow, an efficient library for numerical computation that can take advantage of many differ-

ent architectures, including GPUs. The sgmcmc package keeps much of this efficiency. The package provides functionality to deal with cases where the full MCMC chain is too large to fit into memory. As the chain can be run step by step at each iteration, there is flexibility for these cases.

We implemented the methods on a variety of statistical models, many on realistic datasets. One of these statistical models was a neural network, for which the full MCMC chain would not fit into memory. In this case we demonstrated building test functions and calculating the Monte Carlo estimates on the fly. We empirically demonstrated the predictive performance of the algorithms and the trade-off that appears to occur between predictive performance and exploration.

Many complex models for which SGMCMC methods have been found to perform well require Gibbs updates to be performed periodically (Patterson and Teh, 2013; Li et al., 2016). In the future we would like to build functionality for user defined Gibbs steps that can be updated step by step alongside the sgmcmc algorithms. SGHMC has been implemented by setting the value $\hat{\beta}_t = 0$, as in the experiments of the original paper Chen et al. (2014). In the future, we would like to implement a more sophisticated approach to set this value, such as using a similar estimate to Ahn et al. (2012).

Chapter 5

Large-Scale Stochastic Sampling from the Probability Simplex

5.1 Introduction

Stochastic gradient Markov chain Monte Carlo (SGMCMC) has become a popular method for scalable Bayesian inference (Welling and Teh, 2011; Chen et al., 2014; Ding et al., 2014; Ma et al., 2015). The foundation of SGMCMC methods is a class of continuous-time processes that explore a target distribution—e.g., the posterior—using gradient information; these processes converge to a Markov chain which samples from the posterior distribution exactly. SGMCMC methods replace the costly full-data gradients with minibatch-based stochastic gradients, which provides one source of error. Another source of error arises from the fact that the continuous-time processes are almost never tractable to simulate; instead, discretizations are relied upon. In the non-SG scenario, the discretization errors are corrected for using Metropolis-Hastings

corrections. However, this is not (generically) feasible in the SG setting. The result of these two sources of error is that SGMCMC targets an approximate posterior (Welling and Teh, 2011; Teh et al., 2016; Vollmer et al., 2016).

Another significant limitation of SGMCMC methods is that they struggle to sample from constrained spaces. Naively applying SGMCMC can lead to invalid, or inaccurate values being proposed. The result is large errors near the boundary of the space (Patterson and Teh, 2013; Ma et al., 2015; Li et al., 2016). A particularly important constrained space is the simplex space, which is used to model discrete probability distributions. A parameter ω of dimension d lies in the simplex if it satisfies the following conditions: $\omega_j \geq 0$ for all $j = 1, \ldots, d$ and $\sum_{j=1}^{d} \omega_j = 1$. Many popular models contain simplex parameters. For example, latent Dirichlet allocation (LDA) is defined by a set of topic-specific distributions on words and document-specific distributions on topics. Probabilistic network models often define a link probability between nodes. More generally, mixture and mixed membership models have simplex-constrained mixture weights; even the hidden Markov model can be cast in this framework with simplex-constrained transition distributions. As models become large-scale, these vectors ω often become *sparse*–i.e., many ω_j are close to zero—pushing them to the boundaries of the simplex. All the models mentioned have this tendency. For example in network data, nodes often have relatively few links compared to the size of the network, e.g., the number of friends the average social network user has will be small compared with the size of the whole social network. In these cases the problem of sampling from the simplex space becomes even *harder*; since many values will be very close to the boundary of the space.

Patterson and Teh (2013) develop an improved SGMCMC method for sampling from the probability simplex: stochastic gradient Riemannian Langevin dynamics (SGRLD). The improvements achieved are through an astute transformation of the simplex parameters, as well as developing a Riemannian (see Girolami and Calderhead, 2011) variant of SGMCMC. This method achieved state-of-the-art results on an LDA model. However, we show despite the improvements over standard SGMCMC, the discretization error of this method still causes problems on the simplex. In particular, it leads to asymptotic biases which dominate at the boundary of the space and causes significant inaccuracy.

To counteract this, we design an SGMCMC method based on the Cox-Ingersoll-Ross (CIR) process. The resulting process, which we refer to as the stochastic Cox-Ingersoll-Ross process (SCIR), has *no discretization error*. This process can be used to simulate from gamma random variables directly, which can then be moved into the simplex space using a well known, standard transformation. The CIR process has a lot of nice properties. One is that the transition equation is known exactly, which is what allows us to simulate from the process without discretization error. We are also able to characterize important theoretical properties of the SCIR algorithm, such as the non-asymptotic moment generating function, and thus its mean and variance.

We demonstrate the impact of this SCIR method on a broad class of models. Included in these experiments is the development of a scalable sampler for Dirichlet processes, based on the slice sampler of Walker (2007); Papaspiliopoulos (2008); Kalli et al. (2011). To our knowledge the application of SGMCMC methods to Bayesian nonparametric models has not been explored, and we consider this a further con-

tribution of the article. All proofs in this article are relegated to the Supplementary Material. All code for the experiments will be made available online, and full details of hyperparameter and tuning constant choices has been detailed in the Supplementary Material.

5.2 Stochastic Gradient MCMC on the Probability Simplex

5.2.1 Stochastic Gradient MCMC

Consider Bayesian inference for continuous parameters $\theta \in \mathbb{R}^d$ based on data $\mathbf{x} = \{x_i\}_{i=1}^N$. Denote the density of x_i as $p(x_i|\theta)$ and assign a prior on θ with density $p(\theta)$. The posterior is then defined, up to a constant of proportionality, as $p(\theta|\mathbf{x}) \propto p(\theta)\prod_{i=1}^N p(x_i|\theta)$, and has distribution π. We define $f(\theta) := -\log p(\theta|\mathbf{x})$. Whilst MCMC can be used to sample from π, such algorithms require access to the full data set at each iteration. Stochastic gradient MCMC (SGMCMC) is an approximate MCMC algorithm that reduces this per-iteration computational and memory cost by using only a small subset of data points at each step.

The most common SGMCMC algorithm is stochastic gradient Langevin dynamics (SGLD), first introduced by Welling and Teh (2011). This sampler uses the Langevin diffusion, defined as the solution to the stochastic differential equation

$$d\theta_t = -\nabla f(\theta_t)dt + \sqrt{2}dW_t, \tag{5.2.1}$$

where W_t is a d-dimensional Wiener process. Similar to MCMC, the Langevin diffu-

sion defines a Markov chain whose stationary distribution is π.

Unfortunately, simulating from (5.2.1) is rarely possible, and the cost of calculating ∇f is $O(N)$ since it involves a sum over all data points. The idea of SGLD is to introduce two approximations to circumvent these issues. First, the continuous dynamics are approximated by discretizing them, in a similar way to Euler's method for ODEs. This approximation is known as the *Euler-Maruyama* method. Next, in order to reduce the cost of calculating ∇f, it is replaced with a cheap, unbiased estimate. This leads to the following update equation, with user chosen stepsize h

$$\theta_{m+1} = \theta_m - h\nabla \hat{f}(\theta) + \sqrt{2h}\eta_m, \qquad \eta_m \sim N(0,1). \tag{5.2.2}$$

Here, $\nabla \hat{f}$ is an unbiased estimate of ∇f whose computational cost is $O(n)$ where $n \ll N$. Typically, we set $\nabla \hat{f}(\theta) := -\nabla \log p(\theta) - N/n \sum_{i \in S_m} \nabla \log p(x_i|\theta)$, where $S_m \subset \{1, \ldots, N\}$ resampled at each iteration with $|S_m| = n$. Applying (5.2.2) repeatedly defines a Markov chain that approximately targets π (Welling and Teh, 2011). There are a number of alternative SGMCMC algorithms to SGLD (Chen et al., 2014; Ding et al., 2014; Ma et al., 2015), based on approximations to other diffusions that also target the posterior distribution.

Because the gradient error is typically larger than the discretisation error, recent work has investigated reducing the error introduced by approximating the gradient using minibatches (Dubey et al., 2016; Nagapetyan et al., 2017; Baker et al., 2018; Chatterji et al., 2018). While by comparison, the discretization error is generally smaller, in this work we investigate an important situation where it degrades performance considerably.

5.2.2 SGMCMC on the Probability Simplex

We aim to make inference on the simplex parameter ω of dimension d, where $\omega_j \geq 0$ for all $j = 1, \ldots, d$ and $\sum_{j=1}^{d} \omega_j = 1$. We assume we have categorical data $\mathbf{z}_i$ of dimension d for $i = 1, \ldots, N$, so z_{ij} will be 1 if data point i belongs to category j and z_{ik} will be zero for all $k \neq j$. We assume a Dirichlet prior $\text{Dir}(\alpha)$ on ω, with density $p(\omega) \propto \prod_{j=1}^{d} \omega_d^{\alpha_j}$, and that the data is drawn from $\mathbf{z}_i \,|\, \omega \sim \text{Categorical}(\omega)$ leading to a $\text{Dir}(\alpha + \sum_{i=1}^{N} \mathbf{z}_i)$ posterior. An important transformation we will use repeatedly throughout this article, is that if we have d random gamma variables $X_j \sim \text{Gamma}(\alpha_j, 1)$. Then $(X_1, \ldots, X_d)/\sum_j X_j$ will have $\text{Dir}(\alpha)$ distribution, where $\alpha = (\alpha_1, \ldots, \alpha_d)$.

In this simple case the posterior of ω can be exactly calculated. However, in the applications we consider the $\mathbf{z}_i$ are latent variables, and they are also simulated as part of a larger Gibbs sampler. Thus the $\mathbf{z}_i$ will change at each iteration of our algorithm. We are interested in the situation where this is the case, and N is large, so that standard MCMC runs prohibitively slowly. The idea of SGMCMC in this situation is to use sub-samples of the $\mathbf{z}_i$s to propose appropriate local-moves to ω.

Applying SGMCMC to models which contain simplex parameters is challenging due to their constraints. Naively applying SGMCMC can lead to invalid values being proposed. The first work to introduce an SGMCMC algorithm specifically for the probability simplex was Patterson and Teh (2013), the algorithm is a variant of SGLD known as stochastic gradient Riemannian Langevin dynamics (SGRLD). Patterson and Teh (2013) try a variety of transformations for ω which will move the problem

onto a space in $\mathbb{R}^d$, where standard SGMCMC can be applied. They also build upon standard SGLD by developing a Riemannian variant (see Girolami and Calderhead, 2011). Riemannian MCMC often improves performance when different dimensional components of the parameter have different scales, or are highly correlated. These are both often the case when the parameter lies in the simplex. The parameterisation Patterson and Teh (2013) find numerically performs the best is $\omega_j = |\theta_j| / \sum_{j=1}^{d} |\theta_j|$. They use a mirrored gamma prior for θ_j, which has density $p(\theta_j) \propto |\theta_j|^{\alpha_j - 1} e^{-|\theta_j|}$. This means the prior for ω remains the required Dirichlet distribution. They calculate the density of $\mathbf{z}_i$ given θ using a change of variables and use an SGLD update to calculate θ.

5.2.3 SGRLD on Sparse Simplex Spaces

Patterson and Teh (2013) suggested that the boundary of the space is where most problems occur using these kind of samplers. In many popular applications, such as LDA and modeling sparse networks, some of the components ω_j will be close to 0, referred to as a *sparse* space. In other words, there will be many j for which $\sum_{i=1}^{N} z_{ij} = 0$. In fact, this is their main motivation for introducing the Riemannian ideas to their SGLD algorithm. In order to demonstrate the problems with using SGRLD in this case, we provide a similar experiment to Patterson and Teh (2013). We use SGRLD to simulate from a sparse simplex parameter ω of dimension ten with $N = 1000$. We set $\sum_{i=1}^{N} z_{i1} = 800$, $\sum_{i=1}^{N} z_{i2} = \sum_{i=1}^{N} z_{i3} = 100$, and $\sum_{i=1}^{N} z_{ij} = 0$, for $3 < j \leq 10$. The prior parameter α was set to 0.1 for all components. Leading to a highly sparse Dirichlet posterior, i.e. given the data $\mathbf{z}$, many components of ω will

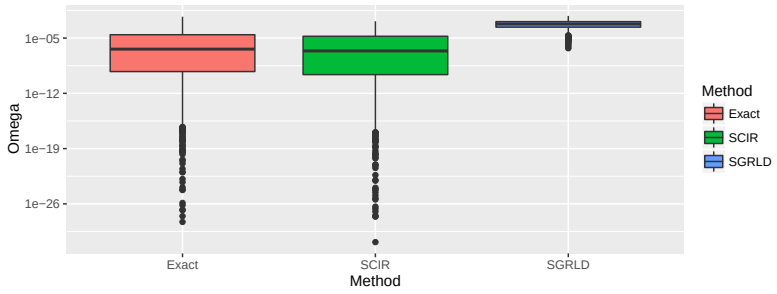


Figure 5.2.1: Boxplots of a 1000 iteration sample from SGRLD and SCIR fit to a sparse Dirichlet posterior, compared to 1000 exact independent samples. On the log scale.

be close to zero. We will refer back to this experiment as the *running experiment*. In Figure 5.2.1 we provide boxplots from a sample of the fifth component of ω using SGRLD after 1000 iterations with 1000 iterations of burn-in, compared with boxplots from an exact sample. The method SCIR will be introduced later. We can see from Figure 5.2.1 that SGRLD rarely proposes small values of ω. This becomes a significant issue for sparse Dirichlet distributions, since the lack of small values leads to a poor approximation to the posterior; as we can see from the boxplots.

We hypothesize that the reason SGRLD struggles when ω_j is near the boundary is due to the discretization by h, and we now try to diagnose this issue in detail. The problem relates to the bias of SGLD, caused by the discretization of the algorithm. We use the results of Vollmer et al. (2016) to characterize this bias for a fixed stepsize h. For similar results when the stepsize scheme is decreasing, we refer the reader to Teh et al. (2016). Proposition 5.2.1 is a simple application of Vollmer et al. (2016, Theorem 3.3), so we refer the reader to that article for full details of the assumptions. For simplicity of the statement, we assume that θ is 1-dimensional, but the results

are easily adapted to the d-dimensional case.

Proposition 5.2.1. *(Vollmer et al., 2016) Under Vollmer et al. (2016, Assumptions 3.1 and 3.2), assume θ is 1-dimensional. Let θ_m be iteration m of an SGLD algorithm for $m = 1, \ldots, M$, then the asymptotic bias defined by $\lim_{M\to\infty} \left| 1/M \sum_{m=1}^{M} \mathbb{E}[\theta_m] - \mathbb{E}_\pi[\theta] \right|$ has leading term $O(h)$.*

While ordinarily this asymptotic bias will be hard to disentangle from other sources of error, as $\mathbb{E}_\pi[\theta]$ gets close to zero, h will have to be set prohibitively small to give a good approximation to θ. The crux of the issue is that, while the *absolute error* remains the same, at the boundary of the space the *relative error* is large since θ is small, and biased upwards due to the positivity constraint. To counteract this, in the next section we introduce a method which has no discretization error. This allows us to prove that the asymptotic bias, as defined in Proposition 5.2.1, will be zero for any choice of stepsize h.

5.3 The Stochastic Cox-Ingersoll-Ross Algorithm

We now wish to counteract the problems with SGRLD on sparse simplex spaces. First, we make the following observation: rather than applying a reparameterization of the prior for ω; we can model the posterior for θ_j directly and independently as $\theta_j \,|\, \mathbf{z} \sim \text{Gamma}(\alpha_j + \sum_{i=1}^{N} z_{ij}, 1)$. Then using the gamma reparameterization $\omega = \theta / \sum_j \theta_j$ still leads to the desired Dirichlet posterior. This leaves the θ_j in a much simpler form, and this simpler form enables us to remove all discretization error. We do this by using an alternative underlying process to the Langevin diffusion. The

diffusion we use is known as the Cox-Ingersoll-Ross (CIR) process, commonly used in mathematical finance. A CIR process θ_t with parameter a and stationary distribution Gamma$(a, 1)$ has the following form

$$d\theta_t = (a - \theta_t)dt + \sqrt{2\theta_t}dW_t. \tag{5.3.1}$$

The standard CIR process has more parameters, but we found changing these made no difference to the properties of our proposed scalable sampler and so we omit them (for exact details see the Supplementary Material).

The CIR process has many nice properties. One that is particularly useful for us is that the *transition density* is known exactly. Define $\chi^2(\nu, \mu)$ to be the non-central chi-squared distribution with ν degrees of freedom and non-centrality parameter μ. If at time t we are at state ϑ_t, then the probability distribution of θ_{t+h} is given by

$$\theta_{t+h} \,|\, \theta_t = \vartheta_t \sim \frac{1 - e^{-h}}{2} W, \qquad W \sim \chi^2\left(2a, 2\vartheta_t \frac{e^{-h}}{1 - e^{-h}}\right). \tag{5.3.2}$$

This transition density allows us to simulate directly from the CIR process with no discretization error. Furthermore, it has been proved that the CIR process is negative with probability zero (Cox et al., 1985), meaning we will not need to take absolute values as is required for the SGRLD algorithm.

5.3.1 Adapting for Large Datasets

The next issue we need to address is how to sample from this process when the dataset is large. Suppose that z_i is data for $i = 1, \ldots, N$, for some large N, and that our target distribution is Gamma$(a, 1)$, where $a = \alpha + \sum_{i=1}^{N} z_i$. We want to

approximate the target by simulating from the CIR process using only a subset of $\mathbf{z}$ at each iteration. A natural thing to do would be at each iteration to replace a in the transition density equation (5.3.2) with an unbiased estimate $\hat{a} = \alpha + N/n \sum_{i \in S} z_i$, where $S \subset \{1, \ldots, N\}$, similar to SGLD. We will refer to a CIR process using unbiased estimates in this way as the stochastic CIR process (SCIR). Fix some stepsize h, which now determines how often $\hat{a}$ is resampled rather than the granularity of the discretization. Suppose $\hat{\theta}_m$ follows the SCIR process, then it will have the following update

$$\hat{\theta}_{m+1} \,|\, \hat{\theta}_m = \vartheta_m \sim \frac{1 - e^{-h}}{2} W, \qquad W \sim \chi^2 \left(2\hat{a}_m, 2\vartheta_m \frac{e^{-h}}{1 - e^{-h}} \right), \tag{5.3.3}$$

where $\hat{a}_m = \alpha + N/n \sum_{i \in S_m} z_i$.

We can show that this algorithm will approximately target the true posterior distribution in the same sense as SGLD. To do this, we draw a connection between the SCIR process and an SGLD algorithm, which allows us to use the arguments of SGLD to show that the SCIR process will target the desired distribution. More formally, we have the following relationship:

Theorem 5.3.1. *Let θ_t be a CIR process with transition 5.3.2. Then $U_t := g(\theta_t) = 2\sqrt{\theta_t}$ follows a Langevin diffusion whose stationary distribution is in the generalized gamma family, with density $p(u) \propto u^{2a-1} e^{-u^2/4}$.*

Theorem 5.3.1, allows us to show that applying the transformation $g(\cdot)$ to the approximate SCIR process, leads to a discretization free SGLD algorithm for a generalized gamma distribution. Similarly, applying $g^{-1}(\cdot)$ to the approximate target of this SGLD algorithm leads to the desired Gamma$(a, 1)$ distribution. Full details

are given after the proof of Theorem 5.3.1. The result means that similar to SGLD, we can replace the CIR parameter a with an unbiased estimate $\hat{a}$ created from a minibatch of data. Provided we re-estimate a from one iteration to the next using different minibatches, the approximate target distribution will still be Gamma$(a, 1)$. As in SGLD, there will be added error based on the noise in the estimate $\hat{a}$. However, from the desirable properties of the CIR process we are able to quantify this error more easily than for the SGLD algorithm, and we do this in Section 5.4.

Algorithm 3 below summarizes how SCIR can be used to sample from the simplex parameter $\omega \,|\, \mathbf{z} \sim \text{Dir}(\alpha + \sum_{i=1}^{N} \mathbf{z}_i)$. This can be done in a similar way to SGRLD, with the same per-iteration computational cost, so the improvements we demonstrate later are essentially for free.

Algorithm 3: Stochastic Cox-Ingersoll-Ross (SCIR) for sampling from the probability simplex.

Input: Starting points θ_0, stepsize h, minibatch size n.

Result: Approximate sample from $\omega \,|\, \mathbf{z}$.

for $m = 1$ **to** M **do**

  Sample minibatch S_m from $\{1, \ldots, N\}$

  for $j = 1$ **to** d **do**

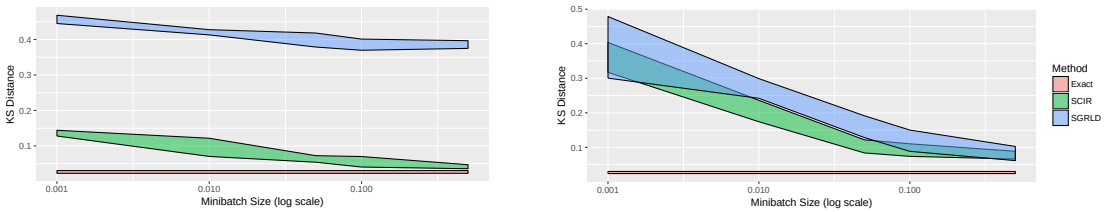
    Set $\hat{a}_j \leftarrow \alpha + N/n \sum_{i \in S_m} z_{ij}$.

    Sample $\hat{\theta}_{mj} \,|\, \hat{\theta}_{(m-1)j}$ using (5.3.3) with parameter $\hat{a}_j$ and stepsize h.

  end

  Set $\omega_m \leftarrow \theta_m / \sum_j \theta_{mj}$.

end



(a) (b)

Figure 5.3.1: Kolmogorov-Smirnov distance for SGRLD and SCIR at different minibatch sizes when used to sample from (a), a sparse Dirichlet posterior and (b) a dense Dirichlet posterior.

5.3.2 SCIR on Sparse Data

We test the SCIR process on two synthetic experiments. The first experiment is the running experiment on the sparse Dirichlet posterior of Section 5.2.3. The second experiment allocates 1000 datapoints equally to each component, leading to a highly dense Dirichlet posterior. For both experiments, we run 1000 iterations of optimally tuned SGRLD and SCIR algorithms and compare to an exact sample. For the sparse experiment, Figure 5.2.1 shows boxplots of samples from the fifth component of ω, which is sparse. For both experiments, Figure 5.3.1 plots the Kolmogorov-Smirnov distance (d_{KS}) between the approximate samples and the true posterior (full details of the distance measure are given in the Supplementary Material). For the boxplots, a minibatch of size 10 is used; for the d_{KS} plots the proportion of data in the minibatch is varied from 0.001 to 0.5. The d_{KS} plots show the runs of five different seeds, which gives some idea of variability.

The boxplots of Figure 5.2.1 demonstrate that the SCIR process is able to handle

smaller values of ω much more readily than SGRLD. The impact of this is demonstrated in Figure 5.3.1a, the sparse d_{KS} plot. Here the SCIR process is achieving much better results than SGRLD, and converging towards the exact sample at larger minibatch sizes. The dense d_{KS} plot of Figure 5.3.1b shows that as we move to the dense setting the samplers have similar properties. The conclusion is that the SCIR algorithm is a good choice of simplex sampler for either the dense or sparse case.

5.4 Theoretical Analysis

In the following theoretical analysis we wish to target a Gamma$(a, 1)$ distribution, where $a = \alpha + \sum_{i=1}^{N} z_i$ for some data $\mathbf{z}$. We run an SCIR algorithm with stepsize h for M iterations, yielding the sample $\hat{\theta}_m$ for $m = 1, \ldots, M$. We compare this to an exact CIR process with stationary distribution Gamma$(a, 1)$, defined by the transition equation in (5.3.2). We do this by deriving the moment generating function (MGF) of $\hat{\theta}_m$ in terms of the MGF of the exact CIR process. This allows us to quantify the moments of $\hat{\theta}_m$ in the analysis to follow. For simplicity, we fix a stepsize h and, abusing notation slightly, set θ_m to be a CIR process that has been run for time mh.

Theorem 5.4.1. *Let $\hat{\theta}_M$ be the SCIR process defined in* (5.3.3) *starting from θ_0 after M steps with stepsize h. Let θ_M be the corresponding exact CIR process, also starting from θ_0, run for time Mh, and with coupled noise. Then the MGF of $\hat{\theta}_M$ is given by*

$$M_{\hat{\theta}_M}(s) = M_{\theta_M}(s) \prod_{m=1}^{M} \left[\frac{1 - s(1 - e^{-mh})}{1 - s(1 - e^{-(m-1)h})} \right]^{-(\hat{a}_m - a)}, \tag{5.4.1}$$

where we have

$$M_{\theta_M}(s) = \left[1 - s(1 - e^{-Mh})\right]^{-a} \exp\left[\theta_0 \frac{se^{-Mh}}{1 - s(1 - e^{-Mh})}\right].$$

The proof of this result follows by induction from the properties of the non-central chi-squared distribution. The result shows that the MGF of the SCIR can be written as the MGF of the exact underlying CIR process, as well as an error term in the form of a product. Deriving the MGF enables us to find the non-asymptotic bias and variance of the SCIR process, which is more interpretable than the MGF itself. The results are stated formally in the following Corollary.

Corollary 5.4.2. *Given the setup of Theorem 5.4.1,*

$$\mathbb{E}[\hat{\theta}_M] = \mathbb{E}[\theta_M] = \theta_0 e^{-Mh} + a(1 - e^{Mh}),$$

so that, since $\mathbb{E}_\pi[\theta] = a$, *then* $\lim_{M\to\infty} |\frac{1}{M}\sum_{m=1}^{M} \mathbb{E}[\hat{\theta}_m] - \mathbb{E}_\pi[\theta]| = 0$ *and SCIR is asymptotically unbiased. Similarly,*

$$\mathbb{V}\text{ar}[\hat{\theta}_M] = \mathbb{V}\text{ar}[\theta_M] + (1 - e^{-2Mh})\frac{1 - e^{-h}}{1 + e^{-h}}\mathbb{V}\text{ar}[\hat{a}],$$

where $\mathbb{V}\text{ar}[\hat{a}] = \mathbb{V}\text{ar}[\hat{a}_m]$ *for* $m = 1, \ldots, M$ *and*

$$\mathbb{V}\text{ar}[\theta_M] = 2\theta_0(e^{-Mh} - e^{-2Mh}) + a(1 - e^{-Mh})^2.$$

The results show that the approximate process is asymptotically unbiased. We believe this explains the improvements the method has over SGRLD in the experiments of Sections 5.3.2 and 5.5. We also obtain the non-asymptotic variance as a simple sum of the variance of the exact underlying CIR process, and a quantity involving the variance of the estimate $\hat{a}$. This is of a similar form to the strong error of SGLD

(Sato and Nakagawa, 2014), though without the contribution from the discretization. The variance of the SCIR is somewhat inflated over the variance of the CIR process. Reducing this variance would improve the properties of the SCIR process and would be an interesting avenue for further work. Control variate ideas could be applied for this purpose (Nagapetyan et al., 2017; Baker et al., 2018; Chatterji et al., 2018) and they may prove especially effective since the mode of a gamma distribution is known exactly.

5.5 Experiments

5.5.1 Latent Dirichlet Allocation

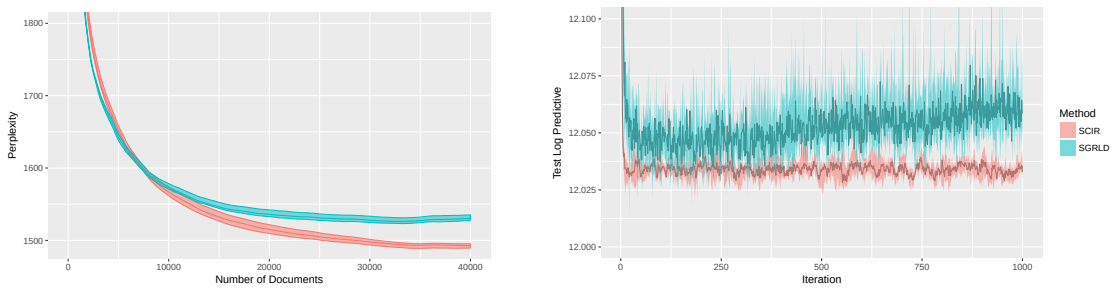
Latent Dirichlet allocation (LDA, see Blei et al., 2003) is a popular model used to summarize a collection of documents by clustering them based on underlying topics. The data for the model is a matrix of word frequencies, with a row for each document. LDA is based on a generative procedure. For each document l, a discrete distribution over the K potential topics, θ_l, is drawn as $\theta_l \sim \text{Dir}(\alpha)$ for some suitably chosen hyperparameter α. Each topic k is associated with a discrete distribution ϕ_k over all the words in a corpus, meant to represent the common words associated with particular topics. This is drawn as $\phi_k \sim \text{Dir}(\beta)$, for some suitable β. Finally, for each word in document l, a topic k is drawn from θ_l; then the word itself is drawn from ϕ_k.

LDA is a good example for this method because ϕ_k is likely to be very sparse, there are many words which will not be associated with a given topic at all. The code is an adaption of the code released by Patterson and Teh (2013), which we

apply to a dataset of scraped Wikipedia documents. At each iteration a minibatch of 50 documents is sampled in an online manner. We use the same vocabulary set as in Patterson and Teh (2013) which consists of approximately 8000 words. The exponential of the average log-predictive on a held out set of 1000 documents is calculated every 5 iterations to evaluate the model. This quantity is known as the perplexity, and use a document completion approach to calculate it (Wallach et al., 2009). The perplexity is plotted for five runs using different seeds, which gives an idea of variability. Similar to Patterson and Teh (2013), for both methods we use a decreasing stepsize scheme of the form $h_m = h[1 + m/\tau]^{-\kappa}$. The results are plotted in Figure 5.5.1a. While the initial convergence rate is similar, SCIR keeps descending past where SGRLD begins to converge. This experiment serves as a good example for the impact that removing the discretization error has for this problem. Further impact would probably be seen if a larger vocabulary size were used, leading to sparser topic vectors. In real-world applications of LDA, it is quite common to use vocabulary sizes above 8000.

5.5.2 Bayesian Nonparametric Mixture Model

We apply SCIR to sample from a Bayesian nonparametric mixture model of categorical data, based on Dunson and Xing (2009). To the best of our knowledge, the development of SGMCMC methods for Bayesian nonparametric models has not been considered before, so we deem this to be another contribution of the work. In particular, we develop a truncation free, scalable sampler based on SGMCMC for Dirichlet processes (DP, see Ferguson, 1973). For more thorough details of DPs and the



(a) (b)

Figure 5.5.1: (a) plots the perplexity of SGRLD and SCIR when used to sample from the LDA model of Section 5.5.1 applied to Wikipedia documents; (b) plots the log predictive on a test set of the anonymous Microsoft user dataset, sampling the mixture model defined in Section 5.5.2 using SCIR and SGRLD.

stochastic sampler developed, the reader is referred to the Supplementary Material. The model can be expressed as follows

$$\mathbf{x}_i \,|\, \theta, z_i \sim \text{Multi}(n_i, \theta_{z_i}), \qquad \theta, z_i \sim \text{DP}(\text{Dir}(a), \alpha). \tag{5.5.1}$$

Here $\text{Multi}(m, \phi)$ is a multinomial distribution with m trials and associated discrete probability distribution ϕ; $\text{DP}(G_0, \alpha)$ is a DP with base distribution G_0 and concentration parameter α. The DP component parameters and allocations are denoted by θ and z_i respectively. We define the number of observations N by $N := \sum_i n_i$, and let L be the number of instances of $\mathbf{x}_i$, $i = 1, \ldots, L$. This type of mixture model is commonly used to model the dependence structure of categorical data, such as for genetic or natural language data (Dunson and Xing, 2009). The use of DPs (Ferguson, 1973) means we can account for the fact that we do not know the true dependence structure. DPs allow us to learn the number of mixture components in a penalized

way during the inference procedure itself.

We apply this model to the anonymous Microsoft user dataset (Breese et al., 1998). This dataset consists of approximately $N = 10^5$ instances of $L = 30000$ anonymized users. Each instance details part of the website the user visits, which is one of $d = 294$ categories (here d denotes the dimension of $\mathbf{x}_i$). We use the model to try and characterize the typical usage patterns of the website. Since there are a lot of categories and only an average of three observations for any one user, these data are expected to be sparse. To infer the model, we use an novel algorithm, which is a minibatched version of the slice sampler (Walker, 2007; Papaspiliopoulos, 2008; Kalli et al., 2011). We assign an uninformative gamma prior on α, and this is inferred similarly to Escobar and West (1995). We minibatch the users at each iteration using $n = 1000$. For multimodal mixture models such as this, SGMCMC methods are known to get stuck in local modes (see Section 2.4.2), so we use a fixed stepsize for both SGRLD and SCIR. Once again, we plot runs over 5 seeds to give an idea of variability. The results are plotted in Figure 5.5.1b. They show that SCIR consistently converges to a lower log predictive test score, and appears to have lower variance than SGRLD. SGRLD also appears to be producing worse scores as the number of iterations increases. We found that SGRLD had a tendency to propose many more clusters than were required. This is probably due to the asymptotic bias of Proposition 5.2.1, since this would lead to an inferred model that has a higher α parameter than is set, meaning more clusters would be proposed than are needed. In fact, setting a higher α parameter appeared to alleviate this problem, but led to a worse fit, which is more evidence that this is the case.

5.6 Discussion

We presented an SGMCMC method, the SCIR algorithm, for simplex spaces. We show that the method has no discretization error and is asymptotically unbiased. Our experiments demonstrate that these properties give the sampler improved performance over other SGMCMC methods for sampling from sparse simplex spaces. Many important large-scale models are sparse, so this is an important contribution. A number of useful theoretical properties for the sampler were derived, including the non-asymptotic variance and moment generating function. Finally, we demonstrate the impact of the sampler on a variety of interesting problems including a novel scalable Dirichlet process sampler. An interesting line of further work would be reducing the non-asymptotic variance, which could be done by means of control variates.

Chapter 6

Conclusions

6.1 Discussion

This thesis has addressed the problem of making MCMC scalable to large datasets. Stochastic gradient MCMC, which aims to cheaply approximate Itô processes that target the posterior exactly, has become a popular scalable MCMC algorithm. The method has substantial theoretical foundation in the form of Itô processes and the Euler–Maruyama method. However, the methods no longer target the posterior exactly, and more work needs to be done to improve the error introduced using these methods.

While SGLD has improved per-iteration cost, there are many empirical results suggesting the method still has $O(N)$ overall computational cost (Welling and Teh, 2011; Nagapetyan et al., 2017). Here by computational cost we mean the number of observations the algorithm needs to process in order to reach a given arbitrary accuracy. This means the scalability of SGLD will grow linearly with the dataset size.

In Chapter 3 we demonstrate how this phenomenon can be avoided using control variates. We introduce the algorithm SGLD-CV, which subject to two one-off passes through the dataset, leads to an algorithm with overall computational cost $O(1)$. We also investigate the usage of post-processing control variates within SGMCMC algorithms, including the variance reduction achieved by SGMCMC.

SGMCMC has received a lot of attention from the machine learning community, but less so from the statistics community. We propose this may be due to lack of computational tools, especially for the programming language R, favoured by many statisticians. In light of this we develop the R software package sgmcmc. Chapter 4 details a background to the package and its usage.

Typical implementations of SGLD on simplex spaces (i.e. sampling from Dirichlet distributions) result in an unstable gradient. Patterson and Teh (2013) get around this by using judicious transformations of the parameters. However, in Chapter 5 we show that this method still has large biases on sparse simplex spaces, where many of the components are zero, due to discretisation of the underlying Langevin diffusion. This is an important problem, as many vital, large-scale models such as network and natural language models rely on sampling from sparse simplex spaces. To counteract this, we develop a scalable algorithm for sampling from the probability simplex that is discretisation free. We show this removes the large biases caused by the discretisation error.

6.2 Future Work

Scalable MCMC, and more specifically SGMCMC, is still an active and open area of research. An important problem that needs to be addressed is that SGMCMC methods still have quite high error, compared to, for example ULA, due to the noise in the gradient estimate. Therefore, more sophisticated variance reduction methods than those introduced in Chapter 3, especially those that do not require setup costs, could prove fruitful. On the other hand, work that allows the piecewise deterministic process samplers of Bierkens et al. (2018a); Bouchard-Côté et al. (2018) to be applied more simply could be more valuable to the community still; since these methods are known to be exact.

Chapter 5 potentially opens up quite a few extensions. There are a number of diffusions whose transition densities are known, which could be exploited in similar ways to the Cox-Ingersoll-Ross process. The normalisation of Gamma random variates in order to simulate from Dirichlet distributions has connections to Bayesian nonparametrics, where Gamma processes can be normalised to create Dirichlet processes. Recently, it has been shown that many traditional models based on exchangeability of the adjacency matrix, such as the stochastic block model, are not able to model sparsity (Caron and Fox, 2017). This leads Caron and Fox (2017) to suggest modelling networks using Bayesian nonparametric models; one Bayesian nonparametric model that has particularly good properties for this case is the generalised Gamma process. Let X be a Gamma distributed random variable, then X^a for $a \in \mathbb{R}_+$ is generalised Gamma distributed, with full flexibility over the distribution parameters. This means

the SCIR process could be used to develop scalable samplers from generalised Gamma processes, and hopefully this modern class of network model.

Appendix A

Appendix to Chapter 3

A.1 Computational Cost Proofs

Proof of Proposition 3.3.6

Proof. Let π be the invariant distribution of the underlying dynamics, so that it has density $e^{-f(\theta)} = p(\theta|\mathbf{x})$, and define $W_2(\nu_k, \pi)$ to be the Wasserstein distance between ν_k and π. Define ξ_k to be the SGLD-CV gradient noise term. Then we can write a single step of SGLD-CV as

$$\theta_{k+1} = \theta_k + h\nabla f(\theta_k) + h\xi_k + \sqrt{2h}\zeta_k,$$

We have that $\theta_k \sim \nu_k$, and follow similarly to the proof of Dalalyan and Karagulyan (2017, Proposition 2). First define Y_0 to be a draw from the invariant distribution π, such that the joint distribution of Y_0 and θ_k minimises $\mathbb{E}\left\|Y_0 - \theta_k\right\|^2$. Here $\|.\|$ denotes the Euclidean distance for $\mathbb{R}^d$. It follows that $\mathbb{E}\left\|Y_0 - \theta_k\right\|^2 = W_2^2(\nu_k, \pi)$.

Let B_t be a d-dimensional Wiener process, independent of θ_k, Y_0 and ξ_k but which

we couple to the injected noise ζ_k so that $B_h = \sqrt{h}\zeta_k$. Now let Y_t, $t > 0$, follow the diffusion

$$Y_t = Y_0 + \int_0^t \nabla f(Y_s)ds + \sqrt{2}B_t. \tag{A.1.1}$$

Let $\Delta_k = Y_0 - \theta_k$ and $\Delta_{k+1} = Y_h - \theta_{k+1}$. Since we started the process Y_t from $Y_0 \sim \pi$, then it follows that $Y_t \sim \pi$ for all $t > 0$. Also since $W_2^2(\nu_{k+1}, \pi)$ minimises the expected squared distance between two random variables with marginals ν_{k+1} and π then it follows that $W_2^2(\nu_{k+1}, \pi) \leq \mathbb{E}\left\|\Delta_{k+1}\right\|^2$.

Let us define

$$U = \nabla f(\theta_k + \Delta_k) - \nabla f(\theta_k), \tag{A.1.2}$$

$$V = \int_0^h \left[\nabla f(Y_t) - \nabla f(Y_0)\right] dt. \tag{A.1.3}$$

Then by the unbiasedness of the gradient estimation, ξ_k has mean 0 regardless of the value of θ_k. Thus

$$\begin{aligned}\mathbb{E}\left\|\Delta_{k+1}\right\|^2 &= \mathbb{E}\left\|\Delta_k + hU + V\right\|^2 + \mathbb{E}\left\|h\xi_k\right\|^2 \\ &\leq \left[\mathbb{E}\left\|\Delta_k - hU\right\| + \mathbb{E}\left\|V\right\|\right]^2 + h^2\mathbb{E}\left\|\xi_k\right\|^2.\end{aligned}$$

We can then apply Lemmas 2 and 4 in Dalalyan and Karagulyan (2017), stated below in Lemmas A.1.1 and A.1.2, as well as applying the gradient noise bound in Lemma 3.3.3, to obtain a bound on $W_2^2(\nu_{k+1}, \pi)$ given $W_2^2(\nu_k, \pi)$.

Lemma A.1.1. *With U as defined in* (A.1.2)*, if $h < 2m/(2M^2+m^2)$, then $\left\|\Delta_k - hU\right\| \leq (1 - mh)\left\|\Delta_k\right\|$.*

The original lemma by Dalalyan and Karagulyan (2017) assumed $h < 2/(m+M)$, but this holds when $h < 2m/(2M^2 + m^2)$ as $m \leq M$.

Lemma A.1.2. *Under Assumption 3.3.1. Let V be as defined in* (A.1.3)*, then*

$$\mathbb{E}\left\|V\right\| \leq \frac{1}{2}(h^4M^3d)^{\frac{1}{2}} + \frac{2}{3}(2h^3d)^{\frac{1}{2}}M.$$

Finally we can apply Lemma 3.3.3, as stated in the main body, to get

$$\begin{aligned}\mathbb{E}\left\|\xi_k\right\|^2 &\leq \frac{M^2}{n}\mathbb{E}\left\|\theta_k - \hat{\theta}\right\|^2 \\ &\leq \frac{2M^2}{n}\mathbb{E}\left\|\theta_k - Y_0\right\|^2 + \frac{2M^2}{n^2}\mathbb{E}\left\|Y_0 - \hat{\theta}\right\|^2 \\ &\leq \frac{2M^2}{n}W_2^2(\nu_k,\pi) + \frac{2M^2}{n}\mathbb{E}\left\|Y_0 - \hat{\theta}\right\|^2\end{aligned}$$

Using Theorem 1 of Durmus and Moulines (2017b)

$$\mathbb{E}\left\|Y_0 - \hat{\theta}\right\|^2 \leq \mathbb{E}\left\|\hat{\theta} - \bar{\theta}\right\|^2 + \frac{d}{m}. \tag{A.1.4}$$

It follows that

$$\mathbb{E}\left\|\xi_k\right\|^2 \leq \frac{2M^2}{n}W_2^2(\nu_k,\pi) + \frac{2M^2}{n}\left[\mathbb{E}\left\|\hat{\theta} - \bar{\theta}\right\|^2 + \frac{d}{m}\right]. \tag{A.1.5}$$

Now using that $W_2^2(\nu_{k+1},\pi) \leq \mathbb{E}\left\|\Delta_{k+1}\right\|^2$ we get the following

$$W_2^2(\nu_{k+1},\pi) \leq \left[(1-mh)W_2(\nu_k,\pi) + \alpha M(h^3d)^{\frac{1}{2}}\right]^2 + \frac{2h^2M^2}{n}W_2^2(\nu_k,\pi) + \frac{2h^2M^2}{n}\left[\mathbb{E}\left\|\hat{\theta} - \bar{\theta}\right\|^2 + \frac{d}{m}\right],$$

where $\alpha = 7\sqrt{2}/6$. Gathering like terms we can further bound $W_2^2(\nu_{k+1},\pi)$ to get the following recursive formula

$$W_2^2(\nu_{k+1},\pi) \leq \left[(1-A)W_2(\nu_k,\pi) + C\right]^2 + B^2$$

where

$$\begin{aligned}A &= 1 - \sqrt{\frac{2h^2M^2}{n} + (1-mh)^2} \\ B &= \sqrt{\frac{2h^2M^2}{n}\left[\mathbb{E}\left\|\hat{\theta} - \bar{\theta}\right\|^2 + \frac{d}{m}\right]} \\ C &= \alpha M(h^3d)^{\frac{1}{2}}.\end{aligned}$$

We can now apply Lemma 1 of Dalalyan and Karagulyan (2017), as stated below to solve this recurrence relation.

Lemma A.1.3. *Let A, B and C be non-negative numbers such that $A \in (0,1)$. Assume that the sequence of non-negative numbers x_k, $k = 0, 1, \ldots,$ satisfies the recursive inequality*

$$x_k^2 \leq [(1-A)x_k + C]^2 + B^2$$

for every integer $k > 0$. Then for all integers $k \geq 0$

$$x_k \leq (1-A)^k x_0 + \frac{C}{A} + \frac{B^2}{C + \sqrt{A}B}$$

To complete the proof all that remains is to check $A \in (0,1)$ so that Lemma A.1.3 can be applied. Clearly $A < 1$, since $n \geq 1$ we have

$$A \geq 1 - \sqrt{2h^2M^2 - (1-mh)^2},$$

and the RHS is positive when $h \in (0, 2m/(2M^2 + m^2))$. □ □

Proof of Theorem 3.3.7

Proof. Starting from Proposition 3.3.6, we have that

$$W_2(\nu_K, \pi) \leq (1-A)^K W_2(\nu_0, \pi) + \frac{C}{A} + \frac{B^2}{C + \sqrt{A}B}. \tag{A.1.6}$$

where

$$A = 1 - \sqrt{\frac{2h^2M^2}{n} + (1-mh)^2}, \qquad B = \sqrt{\frac{2h^2M^2}{n}\left[\mathbb{E}\left\|\hat{\theta} - \bar{\theta}\right\|^2 + \frac{d}{m}\right]}, \qquad C = \alpha M (h^3 d)^{\frac{1}{2}},$$

Suppose we stop the algorithm at iteration K. Using (A.1.6), the following are sufficient conditions that ensure $W_2^2(\nu_K, \pi) < \epsilon_0/\sqrt{m}$,

$$(1-A)^K W_2(\nu_0, \pi) \leq \frac{\epsilon_0}{2\sqrt{m}}, \tag{A.1.7}$$

$$\frac{C}{A} \leq \frac{\epsilon_0}{4\sqrt{m}}, \tag{A.1.8}$$

$$\frac{B^2}{C + \sqrt{A}B} \leq \frac{\epsilon_0}{4\sqrt{m}}. \tag{A.1.9}$$

The starting point θ_0 is deterministic, so from Theorem 1 of Durmus and Moulines (2017b)

$$W_2^2(\nu_0, \pi) \leq \mathbb{E}\left\|\theta_0 - \bar{\theta}\right\|^2 + \frac{d}{m}. \tag{A.1.10}$$

If we rewrite

$$h = \frac{\gamma}{m}\left[\frac{2n}{2R^2 + n}\right], \tag{A.1.11}$$

where $\gamma \in (0,1)$ is some constant and $R := M/m$ as defined in the theorem statement, then it follows that we can write

$$A = 1 - \sqrt{1 - 2mh(1-\gamma)}. \tag{A.1.12}$$

Since we have the condition

$$h \leq \frac{1}{m}\left[\frac{n}{2R^2 + n}\right],$$

then $\gamma \leq \frac{1}{2}$.

Now suppose, using (A.1.12), we set

$$Kh \geq \frac{1}{m}\log\left[\frac{4m}{\epsilon_0^2}\left(\mathbb{E}\left\|\theta_0 - \bar{\theta}\right\|_2^2 + d/m\right)\right] \tag{A.1.13}$$

Then using the result for the deterministic starting point θ_0 (A.1.10), we find that

(A.1.13) implies that

$$\begin{aligned}\frac{\epsilon_0}{2\sqrt{m}} &\geq \exp\left[-mhK/2\right]\sqrt{\mathbb{E}\left\|\theta_0-\bar{\theta}\right\|^2+\frac{d}{m}}\\ &\geq \left[1-mh\right]^{\frac{K}{2}} W_2(\nu_0,\pi)\\ &\geq (1-A)^K W_2(\nu_0,\pi),\end{aligned}$$

Using (A.1.12) and that our conditions imply $\gamma < 1/2$. Hence (A.1.7) holds.

Using that for some real number $y \in [0,1]$, $\sqrt{1-y} \leq 1-y/2$, we can bound A by

$$A \geq 1-\sqrt{1-2mh(1-\gamma)} \geq mh(1-\gamma) := A_0. \tag{A.1.14}$$

As $\gamma \leq 1/2$, for (A.1.8) to hold it is sufficient that

$$\frac{\epsilon_0}{4\sqrt{m}} \geq \frac{C}{A_0},$$

where $C/A_0 \geq 2\alpha M\sqrt{hd}/m$. This leads to the following sufficient condition on h,

$$h \leq \frac{1}{m}\left[\frac{\epsilon_0^2}{64R^2\alpha^2 d}\right] \tag{A.1.15}$$

Similarly for (A.1.9) it is sufficient that

$$\frac{\epsilon_0}{4\sqrt{m}} \geq \frac{B}{\sqrt{A_0}}$$

Now

$$\frac{B}{\sqrt{A_0}} \geq \frac{2\sqrt{h}M\sqrt{\mathbb{E}\left\|\hat{\theta}-\bar{\theta}\right\|^2+d/m}}{\sqrt{mn}}$$

Leading to the following sufficient condition on n

$$n \geq \frac{64hM^2}{\epsilon_0^2}\left[\mathbb{E}\left\|\hat{\theta}-\bar{\theta}\right\|^2+\frac{d}{m}\right].$$

Now due to the conditions on h, define

$$\beta := \max\left\{\frac{1}{2L^2+1}, \frac{\epsilon_0^2}{64L^2\alpha^2 d}\right\}.$$

Then (A.1.9) will hold when

$$n \geq \frac{64L^2\beta}{\epsilon_0^2} m \left[\mathbb{E}\left\|\hat{\theta} - \bar{\theta}\right\|^2 + \frac{d}{m}\right] \tag{A.1.16}$$

□ □

Proof of Lemma 3.3.3

Proof. Our proof follows similarly to Dubey et al. (2016),

$$\begin{aligned}
\mathbb{E}\left\|\xi_k\right\|^2 &= \mathbb{E}\left\|\nabla\tilde{f}(\theta_k) - \nabla f(\theta_k)\right\|^2 \\
&= \mathbb{E}\left\|\nabla f_0(\theta_k) - \nabla f_0(\hat{\theta}) + \frac{1}{n}\sum_{i\in S_k}\frac{1}{p_i}\left[\nabla f_i(\theta_k) - \nabla f_i(\hat{\theta})\right] - \left[\nabla f(\theta_k) - \nabla f(\hat{\theta})\right]\right\|^2 \\
&\leq \frac{1}{n^2}\mathbb{E}\sum_{i\in S_k}\left\|\left[\nabla f(\theta_k) - \nabla f(\hat{\theta})\right] - \left(\nabla f_0(\theta_k) - \nabla f_0(\hat{\theta}) + \frac{1}{p_i}\left[\nabla f_i(\theta_k) - \nabla f_i(\hat{\theta})\right]\right)\right\|^2.
\end{aligned}$$

Where the third line follows due to independence. For any random variable R, we have that $\mathbb{E}\left\|R - \mathbb{E}R\right\|^2 \leq \mathbb{E}\left\|R\right\|^2$. Using this, the smoothness results of Assumption 3.3.2 and our choice of p_i, gives the following, where $\mathbb{E}_I$ refers to expectation with respect to the sampled datum index, I,

$$\begin{aligned}
\mathbb{E}\left\|\xi_k\right\|^2 &\leq \frac{1}{n}\mathbb{E}_I\left(\frac{1}{p_I}\left\|\nabla f_I(\theta_k) - \nabla f_I(\hat{\theta})\right\|^2\right) \\
&\leq \frac{1}{n}\sum_{i=1}^{N}\frac{\sum_{j=1}^{N} L_j}{L_i}\left(L_i\left\|\theta_k - \hat{\theta}\right\|\right)^2 \\
&= \frac{1}{n}\left\{\sum_{i=1}^{N}\left(\sum_{j=1}^{N} L_j\right)L_i\right\}\left\|\theta_k - \hat{\theta}\right\|^2,
\end{aligned}$$

from which the required bound follows trivially. □ □

Proof of Lemma 3.3.5

Proof. Smoothness condition: By the triangle inequality

$$\begin{aligned}\|\nabla f(\theta) - \nabla f(\theta')\| &\leq \sum_{i=0}^{N} \left\| \nabla f_i(\theta) - \sum_{i=0}^{N} f_i(\theta') \right\| \\ &\leq (N+1)L \|\theta - \theta'\| .\end{aligned}$$

Strong convexity: We have that

$$\begin{aligned}f(\theta) - f(\theta') - \nabla f(\theta')^\top(\theta - \theta') &= \sum_{i=0}^{N} \left[f_i(\theta) - f_i(\theta') - \nabla f_i(\theta')^\top(\theta - \theta') \right] \\ &\geq \frac{(N+1)l}{2} \|\theta - \theta'\|_2^2 .\end{aligned}$$

□ □

A.2 Post-processing Proofs

Proof that $\mathbb{E}[\hat{h}(\theta)] = 0$

Proof due to Friel et al. (2016). Let $S \subset \{1, \ldots, N\}$ be the minibatch chosen to estimate $\hat{\mathbf{z}}$, then using the law of total expectation

$$\begin{aligned}\mathbb{E}[\hat{h}(\theta)] &= \mathbb{E}\left[\Delta Q(\theta) + \nabla Q(\theta) \cdot \mathbb{E}[\hat{\mathbf{z}}|S]\right] \\ &= \mathbb{E}\left[\Delta Q(\theta) + \nabla Q(\theta) \cdot \mathbf{z}\right] = \mathbb{E}[h(\theta)] = 0.\end{aligned}$$

Proof of Theorem 3.4.2

Proof. We start from the bound in Theorem 6.1 of Mira et al. (2013), stating for some control variate h, the optimal variance reduction R is given by

$$R = \frac{\left(\mathbb{E}_{\theta|\mathbf{x}}\left[g(\theta)h(\theta)\right]\right)^2}{\mathbb{E}_{\theta|\mathbf{x}}\left[h(\theta)\right]^2},$$

so that in our case we have

$$\begin{aligned}\hat{R} &= \frac{\left(\mathbb{E}_{\theta|\mathbf{x}}\left[g(\theta)\hat{h}(\theta)\right]\right)^2}{\mathbb{E}_{\theta|\mathbf{x}}\left[\hat{h}(\theta)\right]^2} \\ &= \frac{\left(\mathbb{E}_{\theta|\mathbf{x}}\left[g(\theta)h(\theta)\right]\right)^2}{\mathbb{E}_{\theta|\mathbf{x}}\left[h(\theta)\right]^2 + \frac{1}{4}\mathbb{E}_{\theta|\mathbf{x}}\left[\mathbf{a}\cdot\xi_S(\theta)\right]^2} \\ &= \frac{R}{1 + \frac{\frac{1}{4}\mathbb{E}_{\theta|\mathbf{x}}[\mathbf{a}\cdot\xi_S(\theta)]^2}{\mathbb{E}_{\theta|\mathbf{x}}[h(\theta)]^2}}.\end{aligned}$$

Then we can apply Lemmas A.2.1, A.2.2, defined in Section A.2, to get the desired result

$$\hat{R} \geq \frac{R}{1 + [\sigma(N+1)]^{-1}\mathbb{E}_{\theta|\mathbf{x}}\mathbb{E}_S\left\|\xi_S(\theta)\right\|^2}. \tag{A.2.1}$$

□ □

Lemmas

Lemma A.2.1. *Define* $A = \sum_{i=1}^{d} a_i^2$, *and let* $\xi_S(\theta) = \nabla\widehat{\log p(\theta}|\mathbf{x}) - \nabla \log p(\theta|\mathbf{x})$ *be the noise in the gradient estimate. Then*

$$\mathbb{E}_{\theta|\mathbf{x}}\left[\mathbf{a}\cdot\xi_S(\theta)\right]^2 \leq A\mathbb{E}_{\theta|\mathbf{x}}\mathbb{E}_S\left\|\xi_S(\theta)\right\|^2.$$

Proof. We can condition on the gradient noise, and then immediately apply the Cauchy-Schwarz inequality to get

$$\begin{aligned}\mathbb{E}_{\theta|\mathbf{x}}\left[\mathbf{a}\cdot\xi_S(\theta)\right]^2 &= \mathbb{E}_{\theta|\mathbf{x}}\mathbb{E}_S\left[\mathbf{a}\cdot\xi_S(\theta)\right]^2 \\ &\leq \left(\sum_{i=1}^d a_i^2\right)\mathbb{E}_{\theta|\mathbf{x}}\mathbb{E}_S\left\|\xi_S(\theta)\right\|^2\end{aligned}$$

□ □

Lemma A.2.2. *Under Assumption 3.4.1, define* $A = \sum_{i=1}^d a_i^2$. *Then* $\mathbb{E}_{\theta|x}\left[h(\theta)\right]^2 \leq A\sigma(N+1)/4$.

Proof. Applying the Cauchy-Schwarz inequality

$$\begin{aligned}\mathbb{E}_{\theta|x}\left[h(\theta)\right]^2 &\leq \frac{1}{4}\left(\sum_{i=1}^d a_i^2\right)\mathbb{E}_{\theta|\mathbf{x}}\left\|\nabla f(\theta)\right\|^2 \\ &\leq \frac{A(N+1)}{4}\sigma\end{aligned}$$

□ □

A.3 Experiments

Minibatch Sizes

The minibatch sizes were kept fixed for each of the dataset sizes. They are given in the following table:

Stepsize Tuning and Hyperparameters

When tuning the experiments, initially a wide grid search was used to obtain a stepsize to first order. Then, if convergence was insufficient, a more precise grid search was

Model	Minibatch Size
Logistic Regression	500
Probabilistic Matrix Factorisation	5000
LDA	50

Table A.3.1: Minibatch sizes for each of the experiments in 3.5 (they were fixed for SGLD, SGLD-CV and SAGA).

used. In the tables to follow, we detail the optimal stepsizes found in the different experiments. For SGLD-CV we list two sets of stepsizes: SGD and SGLD-CV. SGD corresponds to the stepsizes for the initial optimisation step, SGLD-CV corresponds to the stepsizes for the SGLD-CV algorithm itself.

Logistic Regression

Method	$0.01N$	$0.1N$	N
SGLD	5×10^{-4}	5×10^{-5}	5×10^{-6}
SGLD-CV	5×10^{-4}	5×10^{-5}	5×10^{-6}
SGLD-CV (SGD)	7×10^{-5}	5×10^{-5}	5×10^{-6}
SAGA	1×10^{-3}	1×10^{-4}	1×10^{-5}

Table A.3.2: Tuned stepsizes for the Logistic regression experiment in Section 3.5.1.

Alternative results using a decreasing stepsize scheme of the form $h_k = h(1+k/a)^{-b}$ are given in Figure A.3.1. We use the optimal value of $b = .33$ (Teh et al., 2016). We again tune using a grid search and find optimal h values are the same for the fixed

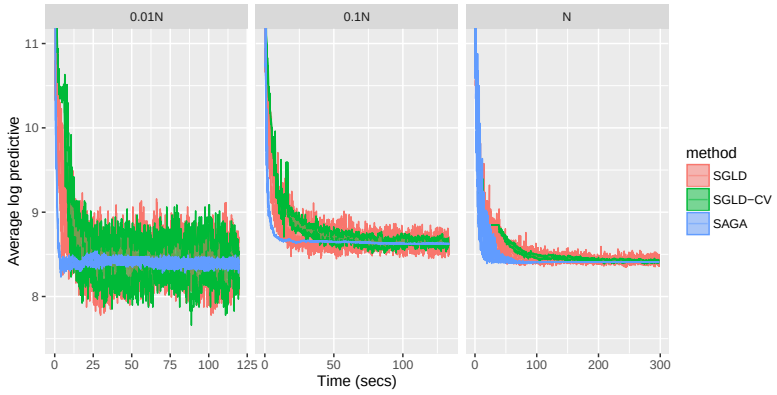


Figure A.3.1: Log predictive density over a test set every 10 iterations of SGLD (with a decreasing stepsize scheme), SGLD-CV and SAGA fit to a logistic regression model as the data size N is varied.

case, and $a = 1000$ is the best for all dataset sizes.

Matrix Factorisation

We use the formulation of BPMF as in Chen et al. (2014, Section H.2). We set $\lambda_U, \lambda_V, \lambda_a, \lambda_b \sim \text{Gamma}(1, 300)$ and $\tau = 3$. This formulation has two matrix parameters U, V and two vector parameters a and b that are learnt by the chosen SGMCMC algorithm. We found these parameters tend to have quite different scales, so setting one global tuning parameter for all mixed poorly. However having four separate stepsizes would also prove difficult to tune. We opted instead to set one global tuning parameter h, detailed in Table A.3.3, and then scale each of the stepsizes based on the relative size of $\mathbb{E}\left[\nabla f(\theta)\right]$. The scaling we opted for, for each parameter θ was $\mathbb{E}\left[\left\|\nabla \hat{f}(\theta)\right\|^2 / d\right]$ for SGLD and SGD; $\mathbb{E}\left[\left\|\nabla \tilde{f}(\theta)^2\right\| / d\right]$ for SGLD-CV; and for SAGA it was equivalent with the corresponding variance reduced gradient. Here, for a given matrix A, $\|A\|$ corresponds to the Frobenius norm. The expectation was estimated

using stochastic optimisation, for example at each iteration k the scaling S_k for SGLD would be estimated by

$$S_k = \frac{1}{k}\left[\frac{\left\|\nabla \hat{f}(\theta_k)\right\|^2}{d} - S_{k-1}\right].$$

Then the local stepsize h_θ for parameter θ would be set to $h_\theta = h/\sqrt{S_k}$. The global stepsizes are detailed in the table below.

Method	$0.1N$	$0.5N$	N
SGLD	1×10^{-3}	5×10^{-3}	5×10^{-3}
SGLD-CV	5×10^{-7}	1×10^{-6}	1×10^{-6}
SGLD-CV (SGD)	1×10^{-2}	1×10^{-2}	1×10^{-2}
SAGA	5×10^{-3}	1×10^{-2}	1×10^{-2}

Table A.3.3: Tuned stepsizes for the Bayesian probabilistic matrix factorisation experiment in Section 3.5.2.

Alternative results using a decreasing stepsize scheme of the form $h_k = h(1+k/a)^{-b}$ are given in Figure A.3.2. We use the optimal value of $b = .33$ (Teh et al., 2016). We again tune using a grid search and find optimal h values are the same for the fixed case, and $a = 1000$ is the best for all dataset sizes.

Latent Dirichlet Allocation

We used the LDA formulation of (Patterson and Teh, 2013), integrating out η, but without the Riemannian information, and using the expanded-natural parameterisation. We use uninformative hyperparameters $\alpha = \beta = 1$.

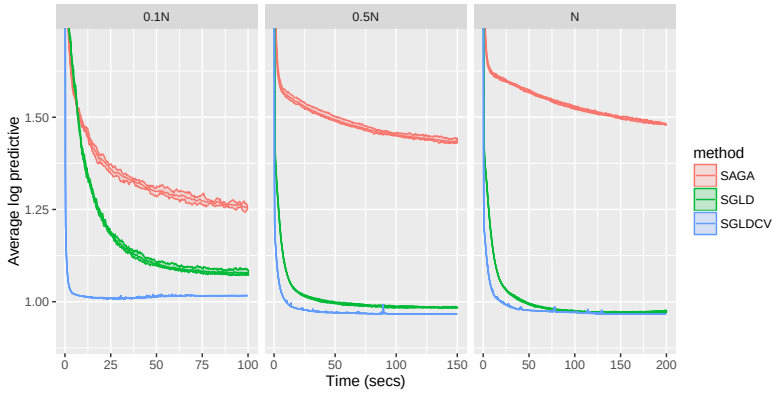


Figure A.3.2: Log predictive density over a test set of SGLD (with a decreasing stepsize scheme), SGLD-CV and SAGA fit to a Bayesian probabilistic matrix factorisation model as the number of users is varied, averaged over 5 runs. We used the Movielens ml-100k dataset.

Method	$0.1N$	$0.6N$	N
SGLD	8×10^{-4}	8×10^{-5}	5×10^{-5}
SGLD-CV	8×10^{-4}	8×10^{-5}	5×10^{-5}
SGLD-CV (SGD)	7×10^{-4}	1×10^{-4}	1×10^{-4}
SAGA	5×10^{-4}	5×10^{-5}	5×10^{-5}

Table A.3.4: Tuned stepsizes for the Bayesian probabilistic matrix factorisation experiment in Section 3.5.2.

Appendix B

Appendix to Chapter 5

B.1 Proofs

Proof of Proposition 5.2.1

Proof. Define the *local weak error* of SGLD, starting from θ_0 and with stepsize h, with test function ϕ by

$$\mathbb{E}\left|\phi(\theta_1) - \phi(\bar{\theta}_h)\right|,$$

where $\bar{\theta}_h$ is the true underlying Langevin diffusion (5.2.1), run for time h with starting point θ_0. Then it is shown by Vollmer et al. (2016) that if $\phi : \mathbb{R}^d \to \mathbb{R}$ is a smooth test function, and that SGLD applied with test function ϕ has local weak error $O(h)$, then

$$\mathbb{E}\left|\lim_{M\to\infty} 1/M \sum_{m=1}^{M} \phi(\theta_m) - \mathbb{E}_\pi[\phi(\theta)]\right|$$

is also $O(h)$. What remains to be checked is that using such a simple function for ϕ (the identity), does not cause things to disappear such that the local weak error

of SGLD is no longer $O(h)$. The identity function is infinitely differentiable, thus is sufficiently smooth. For SGLD, we find that

$$\mathbb{E}[\theta_1|\theta_0] = \theta_0 + hf'(\theta_0).$$

For the Langevin diffusion, we define the one step expectation using the weak Taylor expansion of Zygalakis (2011), which is valid since we have made Assumptions 3.1 and 3.2 of Vollmer et al. (2016). Define the infinitesimal operator $\mathcal{L}$ of the Langevin diffusion (5.2.1) by

$$\mathcal{L}\phi = f'(\theta) \cdot \partial_\theta \phi(\theta) + \partial_\theta^2 \phi(\theta).$$

Then Zygalakis (2011) shows that the weak Taylor expansion of Langevin diffusion (5.2.1) has the form

$$\mathbb{E}[\bar{\theta}_h|\theta_0] = \theta_0 + h\mathcal{L}\phi(\theta_0) + \frac{h^2}{2}\mathcal{L}^2\phi(\theta_0) + O(h^3).$$

This means when ϕ is the identity then

$$\mathbb{E}[\bar{\theta}_h|\theta_0] = \theta_0 + hf'(\theta_0) + \frac{h^2}{2}\left[f(\theta)f'(\theta) + f''(\theta)\right] + O(h^3).$$

Since the terms agree up to $O(h)$ then it follows that even when ϕ is the identity, SGLD still has local weak error of $O(h)$. This completes the proof. □

Proof of Theorem 5.3.1

Proof. Suppose we have a random variable U_∞ following a generalized gamma posterior with data $\mathbf{z}$ and the following density

$$f(u) \propto u^{2(\alpha+\sum_{i=1}^N z_i)-1} e^{-u^2/4}.$$

Set $a := 2(\alpha + \sum_{i=1}^{N} z_i)$, Then $\partial \log f(u) = (2a-1)/u - u/2$, so that the Langevin diffusion for U_∞ will have the following integral form

$$U_{t+h} \,|\, U_t = U_t + \int_t^{t+h} \left[\frac{2a-1}{U_s} - \frac{U_s}{2}\right] ds + \sqrt{2} \int_t^{t+h} dW_t.$$

Applying Ito's lemma to U_t to transform to $\theta_t = g^{-1}(U_t) = U_t^2/4$ (here $g(\cdot)$ has been stated in the proof), we find that

$$\theta_{t+h} \,|\, \theta_t = \theta_t + \int_t^{t+h} [a - \theta_s]\, ds + \int_t^{t+h} \sqrt{2\theta_t} dW_t.$$

This is exactly the integral form for the CIR process. This completes the proof. □

Now we give more details of the connection between SGLD and SCIR. Let us define an SGLD algorithm that approximately targets U_∞, but without the Euler discretization by

$$U_{(m+1)h} \,|\, U_{mh} = U_{mh} + \int_{mh}^{(m+1)h} \left[\frac{2\hat{a}_m - 1}{U_s} - \frac{U_s}{2}\right] ds + \sqrt{2} \int_{mh}^{(m+1)h} dW_t, \qquad \text{(B.1.1)}$$

where $\hat{a}_m$ is an unbiased estimate of a; for example, the standard SGLD estimate $\hat{a}_m = \alpha + N/n \sum_{i \in S_m} z_i$; also h is a tuning constant which determines how much time is simulated before resampling $\hat{a}_m$.

Again applying Ito's lemma to U_{mh} to transform to $\theta_{mh} = g(U_{mh}) = U_{mh}^2/4$, we find that

$$\theta_{(m+1)h} = \theta_{mh} + \int_{mh}^{(m+1)h} [\hat{a}_m - \theta_s]\, ds + \int_{mh}^{(m+1)h} \sqrt{2\theta_t} dW_t.$$

This is exactly the integral form for the update equation of an SCIR process.

Finally, to show SCIR has the desired approximate target, we use some properties of the gamma distribution. Firstly if $\theta_\infty \sim \text{Gamma}(a, 1)$ then $4\theta_\infty \sim \text{Gamma}(a, \frac{1}{4})$,

so that $U_\infty = 2\sqrt{\theta_\infty}$ will have a generalized gamma distribution with density proportional to $h(u) \propto u^{2a-1}e^{-u^2/4}$. This is exactly the approximate target of the discretization free SGLD algorithm (B.1.1) we derived earlier.

Proof of Theorem 5.4.1

First let us define the following quantities

$$r(s) = \frac{se^{-h}}{1 - s(1 - e^{-h})}, \qquad r^{(n)}(s) = \underbrace{r \circ \cdots \circ r}_{\text{n}}(s).$$

Then we will make use of the following Lemmas:

Lemma B.1.1. *For all $n \in \mathbb{N}$ and $s \in \mathbb{R}$*

$$r^{(n)}(s) = \frac{se^{-nh}}{1 - s(1 - e^{-nh})}.$$

Lemma B.1.2. *For all $n \in \mathbb{N}$, $s \in \mathbb{R}$, set $r^{(0)}(s) := s$, then*

$$\prod_{i=0}^{n-1} \left[1 - r^{(i)}(s)(1 - e^{-h})\right] = \left[1 - s(1 - e^{-nh})\right].$$

Both can be proved by induction, which is shown in Section B.2.

Suppose that $\theta_1|\theta_0$ is a CIR process, starting at θ_0 and run for time h. Then we can immediately write down the MGF of θ_1, $M_{\theta_1}(s)$, using the MGF of a non-central chi-squared distribution

$$M_{\theta_1}(s) = \mathbb{E}\left[e^{s\theta_1}|\theta_0\right] = \left[1 - s(1 - e^{-h})\right]^{-a} \exp\left[\frac{s\theta_0 e^{-h}}{1 - s(1 - e^{-h})}\right].$$

We can use this to find $\mathbb{E}\left[e^{s\theta_M} | \, \theta_{M-1}\right]$, and then take expectations of this with respect to θ_{M-2}, i.e. $\mathbb{E}\left[\mathbb{E}\left[e^{s\theta_M} | \, \theta_{M-1}\right] | \, \theta_{M-2}\right]$. This is possible because $\mathbb{E}\left[e^{s\theta_M} | \theta_{M-1}\right]$ has the form $C(s)\exp[\theta_{M-1} r(s)]$, where $C(s)$ is a function only involving s, and $r(s)$ is as defined earlier. Thus repeatedly applying this and using Lemmas B.1.1 and B.1.2 we find

$$M_{\theta_M}(s) = \left[1 - s(1 - e^{-Mh})\right]^{-a} \exp\left[\frac{s\theta_0 e^{-Mh}}{1 - s(1 - e^{-Mh})}\right]. \tag{B.1.2}$$

Although this was already known, we can use the same idea to find the MGF of the SCIR process.

The MGF of SCIR immediately follows using the same logic as before, as well as using the form of $M_{\theta_M}(s)$ and Lemmas B.1.1 and B.1.2. Leading to

$$\begin{aligned} M_{\hat{\theta}_M}(s) &= \prod_{m=1}^{M} \left[1 - r^{(m-1)}(s)(1 - e^{-h})\right]^{-\hat{a}_m} \exp\left[\theta_0 r^{(M)}(s)\right] \\ &= M_{\theta_M}(s) \prod_{m=1}^{M} \left[\frac{1 - s(1 - e^{-mh})}{1 - s(1 - e^{-(m-1)h})}\right]^{-(\hat{a}_m - a)} \end{aligned}$$

Proof of Theorem 5.4.2

Proof. From Theorem 5.4.1, we have

$$M_{\hat{\theta}_M}(s) = M_{\theta_M}(s) \underbrace{\prod_{m=1}^{M} \left[1 - s(1 - e^{-mh})\right]^{-(\hat{a}_m - a)}}_{e_0(s)} \underbrace{\prod_{m=1}^{M} \left[1 - s(1 - e^{-(m-1)h})\right]^{-(a - \hat{a}_m)}}_{e_1(s)}.$$

We clearly have $M_{\theta_M}(0) = e_0(0) = e_1(0) = 1$. Differentiating we find

$$e_0'(s) = \sum_{i=1}^{M} (\hat{a}_i - a)(1 - e^{-ih}) \left[1 - s(1 - e^{-ih})\right]^{-1} e_0(s),$$

similarly

$$e_1'(s) = \sum_{i=1}^{M} (a - \hat{a}_i)(1 - e^{-(i-1)h}) \left[1 - s(1 - e^{-(i-1)h})\right]^{-1} e_1(s).$$

It follows that, labeling the minibatch noise up to iteration M by $\mathcal{B}_M$, and using the fact that $\mathbb{E}\hat{a}_i = a$ for all $i = 1, \ldots, M$ we have

$$\begin{aligned}
\mathbb{E}\hat{\theta}_M &= \mathbb{E}\left[\mathbb{E}\left(\hat{\theta}_M | \mathcal{B}_M\right)\right] \\
&= \mathbb{E}\left[M'_{\hat{\theta}_M}(0)\right] \\
&= \mathbb{E}\left[M'_{\theta_M}(0)e_0(0)e_1(0) + M_{\theta_M}(0)e'_0(0)e_1(0) + M_{\theta_M}(0)e_0(0)e'_1(0)\right] \\
&= \mathbb{E}\theta_M.
\end{aligned}$$

Now taking second derivatives we find

$$e''_0(s) = \sum_{i=1}^{M}(\hat{a}_i - a)(\hat{a}_i - a - 1)(1 - e^{-ih})^2\left[1 - s(1 - e^{-ih})\right]^{-2} e_0(s)$$

$$+\sum_{i\neq j}(\hat{a}_i - a)(\hat{a}_j - a)(1 - e^{-ih})(1 - e^{-jh})\left[1 - s(1 - e^{-ih})\right]^{-1}\left[1 - s(1 - e^{-jh})\right]^{-1} e_0(s).$$

Now taking expectations with respect to the minibatch noise, noting independence of $\hat{a}_i$ and $\hat{a}_j$ for $i \neq j$,

$$\mathbb{E}\left[e''_0(0)\right] = \sum_{i=1}^{M}(1 - e^{-ih})^2 \mathbb{V}\mathrm{ar}(\hat{a}_i).$$

By symmetry

$$\mathbb{E}\left[e''_1(0)\right] = \sum_{i=1}^{M}(1 - e^{-(i-1)h})^2 \mathbb{V}\mathrm{ar}(\hat{a}_i).$$

We also have

$$\mathbb{E}\left[e'_0(0)e'_1(0)\right] = -\sum_{i=1}^{M}(1 - e^{-ih})(1 - e^{-(i-1)h})\mathbb{V}\mathrm{ar}(\hat{a}_i).$$

Now we can calculate the second moment using the MGF as follows, note that $\mathbb{E}(e_0'(0)) = \mathbb{E}(e_1'(0)) = 0$,

$$
\begin{aligned}
\mathbb{E}\hat{\theta}_M^2 &= \mathbb{E}\left[M''_{\hat{\theta}_M}(0)\right] \\
&= \mathbb{E}\left[M''_{\theta_M}(0)e_0(0)e_1(0) + M_{\theta_M}(0)e_0''(0)e_1(0) + M_{\theta_M}(0)e_0(0)e_1''(0) + 2M_{\theta_M}(0)e_0'(0)e_1'(0)\right] \\
&= \mathbb{E}\theta_M^2 + \sum_{i=1}^{M}(1-e^{-ih})^2\mathbb{V}\mathrm{ar}(\hat{a}_i) + \sum_{i=1}^{M}(1-e^{-(i-1)h})^2\mathbb{V}\mathrm{ar}(\hat{a}_i) - 2\sum_{i=1}^{M}(1-e^{-ih})(1-e^{-(i-1)h})\mathbb{V}\mathrm{ar} \\
&= \mathbb{E}\theta_M^2 + \mathbb{V}\mathrm{ar}(\hat{a})\left[e^{-2Mh} - 1 + 2\sum_{i=1}^{M}\left(e^{-2(i-1)h} - e^{-(2i-1)h}\right)\right] \\
&= \mathbb{E}\theta_M^2 + \mathbb{V}\mathrm{ar}(\hat{a})\left[e^{-2Mh} - 1 + 2\sum_{i=0}^{2M-1}(-1)^i e^{-ih}\right] \\
&= \mathbb{E}\theta_M^2 + \mathbb{V}\mathrm{ar}(\hat{a})\left[e^{-2Mh} - 1 + \frac{2-2e^{-2Mh}}{1+e^{-h}}\right] \\
&= \mathbb{E}\theta_M^2 + \mathbb{V}\mathrm{ar}(\hat{a})(1-e^{-2Mh})\left[\frac{1-e^{-h}}{1+e^{-h}}\right]
\end{aligned}
$$

□

B.2 Proofs of Lemmas

Proof of Lemma B.1.1

Proof. We proceed by induction. Clearly the result holds for $n = 1$. Now assume the result holds for all $n \leq k$, we prove the result for $n = k + 1$ as follows

$$\begin{aligned} r^{(k+1)}(s) &= r \circ r^{(k)}(s) \\ &= r\left(\frac{se^{-kh}}{1 - s(1 - e^{-kh})}\right) \\ &= \frac{se^{-kh}}{1 - s(1 - e^{-kh})} \cdot \frac{e^{-h}(1 - s(1 - e^{-kh}))}{1 - s(1 - e^{-kh}) - se^{-kh}(1 - e^{-h})} \\ &= \frac{se^{-(k+1)h}}{1 - s(1 - e^{-(k+1)h})}. \end{aligned}$$

Thus the result holds for all $n \in \mathbb{N}$ by induction. □

Proof of Lemma B.1.2

Proof. Once again we proceed by induction. Clearly the result holds for $n = 1$. Now assume the result holds for all $n \leq k$. Using Lemma B.1.1, we prove the result for $n = k + 1$ as follows

$$\begin{aligned} \prod_{i=0}^{k} \left[1 - r^{(i)}(s)(1 - e^{-h})\right] &= \left[1 - s(1 - e^{-kh})\right] \left[1 - \frac{se^{-kh}(1 - e^{-h})}{1 - s(1 - e^{-kh})}\right] \\ &= \left[1 - s(1 - e^{-kh})\right] \left[\frac{1 - s(1 - e^{-(k+1)h})}{1 - s(1 - e^{-kh})}\right] \\ &= \left[1 - s(1 - e^{-(k+1)h})\right] \end{aligned}$$

Thus the result holds for all $n \in \mathbb{N}$ by induction. □

B.3 CIR Parameter Choice

As mentioned in Section 5.3, the standard CIR process has more parameters than those presented. The full form for the CIR process is as follows

$$d\theta_t = b(a - \theta_t)dt + \sigma\sqrt{\theta_t}dW_t, \tag{B.3.1}$$

where a, b and σ are parameters to be chosen. This leads to a Gamma$(2ab/\sigma^2, 2b/\sigma^2)$ stationary distribution. For our purposes, the second parameter of the gamma stationary distribution can be set arbitrarily, thus it is natural to set $2b = \sigma^2$ which leads to a Gamma$(a, 1)$ stationary distribution and a process of the following form

$$d\theta_t = b(a - \theta_t)dt + \sqrt{2b\theta_t}dW_t.$$

Fix the stepsize h, and use the slight abuse of notation that $\theta_m = \theta_{mh}$. The process has the following transition density

$$\theta_{m+1} \,|\, \theta_m = \vartheta_m \sim \frac{1 - e^{-bh}}{2} W, \qquad W \sim \chi^2\left(2a, 2\vartheta_m \frac{e^{-bh}}{1 - e^{-bh}}\right).$$

Using the MGF of a non-central chi-square distribution we find

$$M_{\theta_M}(s) = \left[1 - s(1 - e^{-Mbh})\right]^{-a} \exp\left[\frac{s\theta_0 e^{-Mbh}}{1 - s(1 - e^{-Mbh})}\right].$$

Clearly b and h are unidentifiable. Thus we arbitrarily set $b = 1$.

B.4 Stochastic Slice Sampler for Dirichlet Processes

Dirichlet Processes

The Dirichlet process (DP) (Ferguson, 1973) is parameterised by a scale parameter $\alpha \in \mathbb{R}_{>0}$ and a base distribution G_0 and is denoted $DP(G_0, \alpha)$. A formal definition

is that G is distributed according to $DP(G_0, \alpha)$ if for all $k \in \mathbb{N}$ and k-partitions $\{B_1, \ldots, B_k\}$ of the space of interest Ω

$$(G(B_1), \ldots, G(B_k)) \sim \text{Dir}(\alpha G_0(B_1), \ldots, \alpha G_0(B_k)).$$

More intuitively, suppose we simulate $\theta_1, \ldots \theta_N$ from G. Then integrating out G (Blackwell and MacQueen, 1973) we can represent θ_N conditional on θ_{-N} as

$$\theta_N \mid \theta_1, \ldots, \theta_{N-1} \sim \frac{1}{N-1+\alpha} \sum_{i=1}^{N-1} \delta_{\theta_i} + \frac{\alpha}{N-1+\alpha} G_0, \tag{B.4.1}$$

where δ_θ is the distribution concentrated at θ.

An explicit construction of a DP exists due to Sethuraman (1994), known as the *stick-breaking construction*. The slice sampler we develop in this section is based on this construction. For $j = 1, 2, \ldots,$ set $V_j \sim \text{Beta}(1, \alpha)$ and $\theta_j \sim G_0$. Then the stick breaking construction is given by

$$\omega_j := V_j \prod_{k=1}^{j-1} (1 - V_k) \tag{B.4.2}$$

$$G \sim \sum_{j=1}^{\infty} \omega_j \delta_{\theta_j}, \tag{B.4.3}$$

and we have $G \sim DP(G_0, \alpha)$.

Slice sampling Dirichlet process mixtures

We focus on sampling from Dirichlet process mixture models defined by

$$X_i \mid \theta_i \sim F(\theta_i)$$

$$\theta_i \mid G \sim G$$

$$G \mid G_0, \alpha \sim DP(G_0, \alpha).$$

A popular MCMC algorithm for sampling from this model is the *slice sampler*, originally developed by Walker (2007) and further developed by Papaspiliopoulos (2008); Kalli et al. (2011). The slice sampler is based directly on the stick-breaking construction (B.4.2), rather than the sequential (Pólya urn) formulation of (B.4.1). This makes it a more natural approach to develop a stochastic sampler from; since the stochastic sampler relies on conditional independence assumptions. The slice sampler can be extended to other Bayesian nonparametric models quite naturally, from their corresponding stick breaking construction.

We want to make inference on a Dirichlet process using the stick breaking construction directly. Suppose the mixture distribution F, and the base distribution G_0 admit densities f and g_0. Introducing the variable z, which determines which component x is currently allocated to, we can write the density as follows

$$p(x|\omega, \theta, z) \propto \omega_z f(x|\theta_z).$$

Theoretically we could now use a Gibbs sampler to sample conditionally from z, θ and ω. However this requires updating an infinite number of weights, similarly z is drawn from a categorical distribution with an infinite number of categories. To get around this Walker (2007) introduces another latent variable u, such that the density is now

$$p(x|\omega, \theta, z, u) \propto \mathbf{1}(u < \omega_z) f(x|\theta_z),$$

so that the full likelihood is given by

$$p(\mathbf{x}|\omega, \theta, \mathbf{z}, \mathbf{u}) \propto \prod_{i=1}^{N} \mathbf{1}(u_i < \omega_{z_i}) f(x_i|\theta_{z_i}). \tag{B.4.4}$$

Walker (2007) shows that in order for a standard Gibbs sampler to be valid given (B.4.4), the number of weights ω_j that needs to be sampled given this new latent variable is now finite, and given by k^*, where k^* is the smallest value such that $\sum_{j=1}^{k^*} \omega_j > 1 - u_i$.

The Gibbs algorithm can now be stated as follows, note we have included an improvement suggested by Papaspiliopoulos (2008), in how to sample v_j.

- Sample the slice variables $\mathbf{u}$, given by $u_i \mid \omega, \mathbf{z} \sim U(0, \omega_{z_i})$ for $i = 1, \ldots, N$. Calculate $u^* = \min \mathbf{u}$.
- Delete or add components until the number of current components k^* is the smallest value such that $u^* < 1 - \sum_{j=1}^{k^*} \omega_j$.
- Draw new component allocations z_i for $i = 1, \ldots, N$, using $p(z_i = j | x_i, u_i, \omega, \theta) \propto \mathbf{1}(\omega_j > u_i) f(x_i|\theta)$.
- For $j \leq k^*$, sample new component parameters θ_j from $p(\theta_j | \mathbf{x}, \mathbf{z}) \propto g_0(\theta_j) \prod_{i:\, z_i = j} f(x_i|\theta_j)$
- For $j \leq k^*$ calculate simulate new stick breaks v from $v_j \mid \mathbf{z}, \alpha \sim \text{Beta}\left(1 + m_j, \alpha + \sum_{l=j+1}^{k^*} m_l\right)$. Here $m_j := \sum_{i=1}^{N} \mathbf{1}_{z_i = j}$.
- Update ω using the new v: $\omega_j = v_j \prod_{l<j}(1 - v_j)$.

Stochastic Sampler

The conditional independence of each update of the slice sampler introduced in Section B.4 makes it possible to adapt it to a stochastic variant. Suppose we update θ and

v given a minibatch of the $\mathbf{z}$ and $\mathbf{u}$ parameters. Then since the $\mathbf{z}$ and $\mathbf{u}$ parameters are just updated from the marginal of the posterior, only updating a minibatch of these parameters at a time would leave the posterior as the invariant distribution. Our exact MCMC procedure is similar to that in the `R` package `PReMiuM` (Liverani et al., 2015), though they do not use a stochastic sampler. First define the following: $Z^* = \max \mathbf{z}$; $S \subset \{1, \ldots, N\}$ is the current minibatch; $u^* = \min \mathbf{u}_S$; k^* is the smallest value such that $\sum_{j=1}^{k^*} \omega_j > 1 - u^*$. Then our updates proceed as follows:

- Recalculate Z^* and S (note this can be done in $O(n)$ time since only n $\mathbf{z}$ values changed).
- For $j = 1, \ldots, Z^*$ sample v_j stochastically with SCIR from $v_j \,|\, \mathbf{z}, \alpha \sim \text{Beta}(1 + \hat{m}_j, \alpha + \sum_{l=j+1}^{k^*} \hat{m}_l)$. Here $\hat{m}_j = N/n \sum_{i \in S} \mathbf{1}_{z_i = j}$.
- Update ω_j using the new v: $\omega_j = v_j \prod_{l<j}(1 - v_j)$.
- For $j = 1, \ldots, Z^*$ sample θ_j stochastically with SGMCMC from $p(\theta_j | \mathbf{x}, \mathbf{z}) \propto g_0(\theta_j) \prod_{S_j} f(x_i | \theta_j)$. Here $S_j = \{i : z_i = j \text{ and } i \in S\}$.
- For $i \in S$ sample the slice variables $u_i \,|\, \omega, \mathbf{z} \sim U(0, \omega_{z_i})$.
- Sample α if required. Using Escobar and West (1995), for our example we assume a Gamma(b_1, b_2) prior so that $\alpha \,|\, v_{1:Z^*} \sim \text{Gamma}(b_1 + Z^*, b_2 - \sum_{j=1}^{K^*} \log(1 - v_j))$.
- Recalculate u^*. Sample additional ω_j from the prior, until k^* is reached. For $j = (Z^* + 1), \ldots, k^*$ sample additional θ_j from the prior.

- For $i \in S$, sample z_i, where $\mathbb{P}(z_i = j|u_i, \omega, \theta, \mathbf{x}) \propto \mathbf{1}(\omega_j > u_i) f(x_i|\theta_j)$.

Note that for our particular example, we have the following conditional update for θ (ignoring minibatching for simplicity):

$$\theta_j \,|\, z_j, \mathbf{x} \sim \text{Dirichlet}\left(a + \sum_{i \in S_j} x_{i1}, \ldots, a + \sum_{i \in S_j} x_{id}\right).$$

B.5 Experiments

Synthetic

We now fully explain the distance measure used in the synthetic experiments. Suppose we have random variables X taking values in $\mathbb{R}$ with cumulative density function (CDF) F. We also have an approximate sample from X, $\hat{X}$ with empirical density function $\hat{F}$. The Kolmogorov-Smirnov distance d_{KS} between X and $\hat{X}$ is defined by $d_{KS}(X, \hat{X}) = \sup_{x \in \mathbb{R}} \left\| \hat{F}(x) - F(x) \right\|$. However the Dirichlet distribution is multi-dimensional, so we measure the average Kolmogorov-Smirnov distance across dimensions by using the Rosenblatt transform (Rosenblatt, 1952).

Suppose now that X takes values in $\mathbb{R}^d$. Define the conditional CDF of $X_k = x_k | X_{k-1} = x_{k-1}, \ldots, X_1 = x_1$ to be $F(x_k|\mathbf{x}_{1:(k-1)})$. Suppose we have an approximate sample from X, which we denote $\mathbf{x}^{(m)}$, for $m = 1, \ldots M$. Define $\hat{F}_j$ to be the empirical CDF defined by the samples $F(x_j^{(m)}|\mathbf{x}_{1:(j-1)}^{(m)})$. Then Rosenblatt (1952) showed that if $\hat{X}$ is a true sample from X then $\hat{F}_j$ should be the uniform distribution and independent of $\hat{F}_k$ for $k \neq j$. This allows us to define a Kolmogorov-Smirnov distance measure

across multiple dimensions as follows

$$d_{KS}(X, \hat{X}) = \frac{1}{K} \sum_{j=1}^{K} \sup_{x \in \mathbb{R}} \left\| \hat{F}_j(x) - F_j(x) \right\|.$$

Where here applying Rosenblatt (1952), $F_j(X)$ is just the uniform distribution.

The full posterior distributions for the sparse and dense experiments are as follows:

$$\omega_{\text{sparse}} \,|\, \mathbf{z} \sim \text{Dir}\left[800.1, 100.1, 100.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1\right],$$

$$\omega_{\text{dense}} \,|\, \mathbf{z} \sim \text{Dir}\left[112.1, 119.1, 92.1, 98.1, 95.1, 96.1, 102.1, 92.1, 91.1, 103.1\right].$$

For each of the five random seeds, we pick the stepsize giving the best d_{KS} for SGRLD and SCIR from the following options:

Method	h							
SCIR	1.0	5e-1	1e-1	5e-2	1e-2	5e-3	1e-3	
SGRLD	5e-1	1e-1	5e-2	1e-2	5e-3	1e-3	5e-4	1e-4

Table B.5.1: Stepsizes for the synthetic experiment

Latent Dirichlet Allocation

As mentioned in the main body, we use a decreasing stepsize scheme of the form $h_m = h(1 + m/\tau)^{-\kappa}$. We do this to be fair to SGRLD, where the best performance is found by using this decreasing scheme (Patterson and Teh, 2013; Ma et al., 2015); and this will probably reduce some of the bias due to the stepsize h. We find a decreasing stepsize scheme of this form also benefits SCIR, so we use it as well. Notice that we find similar optimal hyperparameters for SGRLD to Patterson and Teh (2013). Table B.5.2 fully details the hyperparameter settings we use for the LDA experiment.

Method	h	τ	κ	α	β	K	n	Gibbs Samples
CIR	0.5	10.	.33	0.1	0.5	100	50	200
SGRLD	0.01	1000.	.6	0.01	0.0001	100	50	200

Table B.5.2: Hyperparameters for the LDA experiment

Bayesian Nonparametric Mixture

For details of the stochastic slice sampler we use, please refer to Section B.4. Figure B.5.3 details full hyperparameter settings for the Bayesian nonparametric mixture experiment. Note that h_θ corresponds to the stepsizes assigned for sampling the θ parameters; while h_{DP} corresponds to the stepsizes assigned for sampling from the weights ω for the Dirichlet process.

Method	h_θ	h_{DP}	a	K	n
CIR	0.1	0.1	0.5	20	1000
SGRLD	0.001	0.005	0.001	30	1000

Table B.5.3: Hyperparameters for the Bayesian nonparametric mixture experiment

Bibliography

R. Abraham, J. E. Marsden, and R. Ratiu. *Manifolds, Tensor Analysis, and Applications*, volume 2. Springer-Verlag, 1988.

Sungjin Ahn, Anoop Korattikara, and Max Welling. Bayesian posterior sampling via stochastic gradient Fisher scoring. In *Proceedings of the 29th International Conference on Machine Learning*, pages 1591–1598, 2012.

Sungjin Ahn, Anoop Korattikara, Nathan Liu, Suju Rajan, and Max Welling. Large-scale distributed Bayesian matrix factorization using stochastic gradient MCMC. In *Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 9–18, 2015.

JJ Allaire, Dirk Eddelbuettel, Nick Golding, and Yuan Tang. *TensorFlow: R Interface to TensorFlow*, 2016. URL `https://github.com/rstudio/tensorflow`.

Jack Baker, Paul Fearnhead, Emily B. Fox, and Christopher Nemeth. Control variates for stochastic gradient MCMC. *Statistics and Computing*, 2018. URL `https://doi.org/10.1007/s11222-018-9826-2`. To Appear.

A. D. Barbour. Stein's method and poisson process convergence. *Journal of Applied Probability*, 25:175–184, 1988.

Rémi Bardenet, Arnaud Doucet, and Chris Holmes. On markov chain monte carlo methods for tall data. *Journal of Machine Learning Research*, 18(47):1–43, 2017.

Bo Martin Bibby, Ib Michael Skovgaard, and Michael Sørensen. Diffusion-type models with given marginal distribution and autocorrelation function. *Bernoulli*, 11(2): 191–220, 2005.

Joris Bierkens, Paul Fearnhead, and Gareth Roberts. The zig-zag process and super-efficient sampling for Bayesian analysis of big data. *The Annals of Statistics*, 2018a. To Appear.

Joris Bierkens, Gareth Roberts, and Pierre-Andr Zitt. Ergodicity of the zigzag process. Available at `https://arxiv.org/abs/1712.09875`, 2018b.

Jock A Blackard and Denis J Dean. Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and electronics in agriculture*, 24(3):131–151, 1999.

David Blackwell and James B. MacQueen. Ferguson distributions via Polya urn schemes. *The Annals of Statistics*, 1(2):353–355, 1973.

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

Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of the 19th International Conference on Computational Statistics*, pages 177–187. Springer, 2010.

Alexandre Bouchard-Côté, Sebastian J. Vollmer, and Arnaud Doucet. The Bouncy Particle Sampler: A nonreversible rejection-free Markov chain Monte Carlo method. *Journal of the American Statistical Association*, 113(522):855–867, 2018.

John S. Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pages 43–52, 1998.

François Caron and Emily B Fox. Sparse graphs using exchangeable random measures. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 79(5): 1295–1366, 2017.

Bob Carpenter, Andrew Gelman, Matthew Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. Stan: A probabilistic programming language. *Journal of Statistical Software*, 76 (1), 2017.

Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3):1–27, 2011. URL `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Niladri S Chatterji, Nicolas Flammarion, Yi-An Ma, Peter L Bartlett, and Michael I

Jordan. On the theory of variance reduction for stochastic gradient Monte Carlo. Available at `https://arxiv.org/abs/1802.05431v1`, 2018.

Changyou Chen, Nan Ding, and Lawrence Carin. On the convergence of stochastic gradient MCMC algorithms with high-order integrators. In *Advances in Neural Information Processing Systems 28*, pages 2278–2286, 2015.

Changyou Chen, Wenlin Wang, Yizhe Zhang, Qinliang Su, and Lawrence Carin. A convergence analysis for a class of practical variance-reduction stochastic gradient MCMC. Available at `https://arxiv.org/abs/1709.01180`, 2017.

Tianqi Chen, Emily Fox, and Carlos Guestrin. Stochastic gradient Hamiltonian Monte Carlo. In *Proceedings of the 31st International Conference on Machine Learning*, pages 1683–1691, 2014.

Xiang Cheng, Niladri S. Chatterji, Yasin Abbasi-Yadkori, Peter L. Bartlett, and Michael I. Jordan. Sharp convergence rates for Langevin dynamics in the non-convex setting. Available from `https://arxiv.org/abs/1805.01648`, 2018.

John C. Cox, Jonathan E. Ingersoll, and Stephen A. Ross. A theory of the term structure of interest rates. *Econometrica*, 53(2):385–407, 1985.

Arnak S. Dalalyan. Theoretical guarantees for approximate sampling from smooth and log-concave densities. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 79(3):651–676, 2016.

Arnak S Dalalyan and Avetik G Karagulyan. User-friendly guarantees for the

Langevin Monte Carlo with inaccurate gradient. Available at `https://arxiv.org/abs/1710.00095`, 2017.

Mark HA Davis. Piecewise-deterministic markov processes: A general class of non-diffusion stochastic models. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, pages 353–388, 1984.

George Deligiannidis, Alexandre Bouchard-Côté, and Arnaud Doucet. Exponential Ergodicity of the Bouncy Particle Sampler. *Annals of Statistics*, 2018. To Appear.

Nan Ding, Youhan Fang, Ryan Babbush, Changyou Chen, Robert D Skeel, and Hartmut Neven. Bayesian sampling using stochastic gradient thermostats. In *Advances in Neural Information Processing Systems 27*, pages 3203–3211, 2014.

Kumar Avinava Dubey, Sashank J Reddi, Sinead A Williamson, Barnabas Poczos, Alexander J Smola, and Eric P Xing. Variance reduction in stochastic gradient Langevin dynamics. In *Advances in Neural Information Processing Systems 29*, pages 1154–1162. Curran Associates, Inc., 2016.

David B Dunson and Chuanhua Xing. Nonparametric Bayes modeling of multivariate categorical data. *Journal of the American Statistical Association*, 104(487):1042–1051, 2009.

Alain Durmus and Eric Moulines. Nonasymptotic convergence analysis for the unadjusted Langevin algorithm. *The Annals of Applied Probability*, 27(3):1551–1587, 2017a.

Alain Durmus and Eric Moulines. High-dimensional Bayesian inference via the unadjusted Langevin algorithm. 2017b. Available at `https://hal.archives-ouvertes.fr/hal-01304430/`.

Michael D Escobar and Mike West. Bayesian density estimation and inference using mixtures. *Journal of the American Statistical Association*, 90(430):577–588, 1995.

Paul Fearnhead, Joris Bierkens, Murray Pollock, and Gareth O. Roberts. Piecewise deterministic Markov processes for continuous-time Monte Carlo. *Statistical Science*, 33(3):386–412, 2018.

Thomas S. Ferguson. A Bayesian analysis of some nonparametric problems. *The Annals of Statistics*, 1(2):209–230, 1973.

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The Elements of Statistical Learning*, volume 1. Springer-Verlag, 2001.

Nial Friel, Antonietta Mira, and Chris Oates. Exploiting multi-core architectures for reduced-variance estimation with intractable likelihoods. *Bayesian Analysis*, 11(1): 215–245, 2016.

S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(6):721–741, 1984.

Charles J Geyer. Markov chain Monte Carlo lecture notes, 2005. Unpublised lecture notes. Available from `http://www.stat.umn.edu/geyer/f05/8931/n1998.pdf`.

Mike Giles, Tigran Nagapetyan, Lukasz Szpruch, Sebastian Vollmer, and Konstantinos Zygalakis. Multilevel Monte Carlo for scalable Bayesian computations. Available from `https://arxiv.org/abs/1609.06144`, 2016.

Mark Girolami and Ben Calderhead. Riemann manifold Langevin and Hamiltonian Monte Carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 73(2):123–214, 2011.

Jack Gorham, Andrew B Duncan, Sebastian J Vollmer, and Lester Mackey. Measuring sample quality with diffusions. Available from `https://arxiv.org/abs/1611.06972`, 2016.

Peter J. Green. Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. *Biometrika*, 82(4):711–732, 1995.

Andreas Griewank and Andrea Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, volume 2. SIAM, 2008.

W. K. Hastings. Monte Carlo sampling methods using Markov Chains and their applications. *Biometrika*, 57(1):97–109, 1970.

Matthew D. Hoffman and Andrew Gelman. The No-U-Turn Sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research*, 15:1593–1623, 2014.

Alan M. Horowitz. A generalized guided Monte Carlo algorithm. *Physics Letters B*, 268(2):247 – 252, 1991.

Maria Kalli, Jim E Griffin, and Stephen G Walker. Slice sampling mixture models. *Statistics and Computing*, 21(1):93–105, 2011.

Rafail Khasminskii. *Stochastic stability of differential equations*, volume 66. Springer-Verlag, 2011.

Peter E. Kloeden and Eckhard Platen. *Numerical Solution of Stochastic Differential Equations*, volume 1. Springer-Verlag, 1992.

Damien Lamberton and Gilles Pagès. Recursive computation of the invariant distribution of a diffusion. *Bernoulli*, 8(3):367–405, 2002.

Lucien Le Cam. *Asymptotic methods in statistical decision theory*. Springer, 2012.

Yann LeCun and Corinna Cortes. *MNIST Handwritten Digit Database*, 2010. URL `http://yann.lecun.com/exdb/mnist/`.

B. Leimkuhler and X. Shang. Adaptive thermostats for noisy gradient systems. *SIAM Journal on Scientific Computing*, 38(2):A712–A736, 2016.

P. A. W Lewis and G. S. Shedler. Simulation of nonhomogeneous Poisson processes by thinning. *Naval Research Logistics Quarterly*, 26(3):403–413.

Cheng Li, Sanvesh Srivastava, and David B. Dunson. Simple, scalable and accurate posterior interval estimation. *Biometrika*, 104(3):665–680, 2017.

S Li, A Beygelzimer, S Kakadet, J Langford, S Arya, and D Mount. FNN: fast nearest neighbor search algorithms and applications. R package version 1.1. Available at `https://cran.r-project.org/web/packages/FNN/`, 2013.

Wenzhe Li, Sungjin Ahn, and Max Welling. Scalable MCMC for mixed membership stochastic blockmodels. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, pages 723–731, 2016.

Silvia Liverani, David Hastie, Lamiae Azizi, Michail Papathomas, and Sylvia Richardson. PReMiuM: An R package for profile regression mixture models using Dirichlet processes. *Journal of Statistical Software*, 64(7):1–30, 2015.

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.

Yi-An Ma, Tianqi Chen, and Emily Fox. A complete recipe for stochastic gradient MCMC. In *Advances in Neural Information Processing Systems 28*, pages 2917–2925. 2015.

J. Mattingly, A. Stuart, and M. Tretyakov. Convergence of numerical time-averaging and stationary measures via Poisson equations. *SIAM Journal on Numerical Analysis*, 48(2):552–577, 2010.

J.C. Mattingly, A.M. Stuart, and D.J. Higham. Ergodicity for SDEs and approximations: locally Lipschitz vector fields and degenerate noise. *Stochastic Processes and their Applications*, 101(2):185 – 232, 2002. ISSN 0304-4149.

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.

Sean P. Meyn and R. L. Tweedie. Stability of Markovian processes I: criteria for discrete-time chains. *Advances in Applied Probability*, 24(3):542574, 1992.

Sean P Meyn and Richard L Tweedie. *Markov Chains and Stochastic Stability*, volume 1. Springer-Verlag, 1993a.

Sean P Meyn and Richard L Tweedie. Stability of Markovian processes II: Continuous-time processes and sampled chains. *Advances in Applied Probability*, 25(3):487–517, 1993b.

Sean P Meyn and Richard L Tweedie. Stability of Markovian processes III: Foster–Lyapunov criteria for continuous-time processes. *Advances in Applied Probability*, 25(3):518–548, 1993c.

Stanislav Minsker, Sanvesh Srivastava, Lizhen Lin, and David Dunson. Scalable and robust Bayesian inference via the median posterior. In *Proceedings of the 31st International Conference on Machine Learning*, volume 32, pages 1656–1664, 2014.

Stanislav Minsker, Sanvesh Srivastava, Lizhen Lin, and David B. Dunson. Robust and scalable Bayes via a median of subset posterior measures. *Journal of Machine Learning Research*, 18(124):1–40, 2017.

Antonietta Mira, Reza Solgi, and Daniele Imparato. Zero variance Markov chain Monte Carlo for Bayesian estimators. *Statistics and Computing*, 23(5):653–662, 2013.

Andriy Mnih and Ruslan R Salakhutdinov. Probabilistic matrix factorization. In *Advances in Neural Information Processing Systems 20*, pages 1257–1264, 2008.

Tigran Nagapetyan, Andrew Duncan, Leonard Hasenclever, Sebastian J Vollmer, Lukasz Szpruch, and Konstantinos Zygalakis. The true cost of stochastic gradient Langevin dynamics. Available at `https://arxiv.org/abs/1706.02692`, 2017.

Radford M Neal. MCMC using Hamiltonian Dynamics. In *Handbook of Markov Chain Monte Carlo*. Chapman & Hall, 2010.

Willie Neiswanger, Chong Wang, and Eric P. Xing. Asymptotically exact, embarrassingly parallel MCMC. In *Proceedings of the 30th Conference on Uncertainty in Artificial Intelligence*, pages 623–632. AUAI Press, 2014.

Christopher Nemeth and Chris Sherlock. Merging MCMC subposteriors through Gaussian-process approximations. *Bayesian Analysis*, 13(2):507–530, 2018.

A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro. Robust stochastic approximation approach to stochastic programming. *SIAM Journal on Optimization*, 19(4):1574–1609, 2009.

Bernt Øksendal. *Stochastic Differential Equations*, volume 6. Springer-Verlag, 2003.

Theodore Papamarkou, Antonietta Mira, Mark Girolami, et al. Zero variance differential geometric Markov chain Monte Carlo algorithms. *Bayesian Analysis*, 9(1): 97–128, 2014.

Omiros Papaspiliopoulos. A note on posterior sampling from Dirichlet mixture models. Technical Report. Available at `http://wrap.warwick.ac.uk/35493/1/WRAP_papaspliiopoulos_08-20wv2.pdf`, 2008.

Sam Patterson and Yee Whye Teh. Stochastic gradient Riemannian Langevin dynamics on the probability simplex. In *Advances in Neural Information Processing Systems 26*, pages 3102–3110, 2013.

I G Petrovskii. *Lectures on Partial Differential Equations. GITTL, Moscow (1950). English translation.* Interscience, New York, 1954.

TensorFlow Development Team. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*, 2015. URL `http://tensorflow.org`.

Martyn Plummer. *JAGS: A program for analysis of Bayesian graphical models using Gibbs sampling*, 2003. URL `http://mcmc-jags.sourceforge.net/`.

Murray Pollock, Paul Fearnhead, Adam M Johansen, and Gareth O Roberts. The scalable Langevin exact algorithm: Bayesian inference for big data. Available at `https://arxiv.org/abs/1609.03436`, 2016.

Maxim Raginsky, Alexander Rakhlin, and Matus Telgarsky. Non-convex learning via stochastic gradient Langevin dynamics: a nonasymptotic analysis. In *Proceedings of the 2017 Conference on Learning Theory*, volume 65, pages 1674–1703, 2017.

R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, 2008. URL `http://www.R-project.org`.

Brian D Ripley. *Stochastic simulation*. John Wiley & Sons, 2009.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.

Christian Robert and George Casella. *Monte Carlo Statistical Methods*, volume 2. Springer-Verlag, 2004.

Gareth O Roberts and Jeffrey S Rosenthal. Optimal scaling of discrete approximations to Langevin diffusions. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 60(1):255–268, 1998.

Gareth O. Roberts and Jeffrey S. Rosenthal. General state space Markov chains and MCMC algorithms. *Probability Surveys*, 1:20–71, 2004.

Gareth O. Roberts and Richard L. Tweedie. Exponential convergence of Langevin distributions and their discrete approximations. *Bernoulli*, 2(4):341–363, 1996.

Murray Rosenblatt. Remarks on a multivariate transformation. *The Annals of Mathematical Statistics*, 23(3):470–472, 1952.

Issei Sato and Hiroshi Nakagawa. Approximation analysis of stochastic gradient Langevin dynamics by using Fokker–Planck equation and Ito process. In *Proceedings of the 31st International Conference on Machine Learning*, pages 982–990. PMLR, 2014.

Steven L. Scott, Alexander W. Blocker, Fernando V. Bonassi, Hugh A. Chipman, Edward I. George, and Robert E. McCulloch. Bayes and big data: The consensus Monte Carlo algorithm. *International Journal of Management Science and Engineering Management*, 11(2):78–88, 2016.

Jayaram Sethuraman. A constructive definition of Dirichlet priors. *Statistica Sinica*, 4(2):639–650, 1994.

Sanvesh Srivastava, Volkan Cevher, Quoc Dinh, and David Dunson. WASP: Scalable Bayes via barycenters of subset posteriors. In *Proceedings of the 18th International Conference on Artificial Intelligence and Statistics*, volume 38, pages 912–920, 2015.

Denis Talay and Luciano Tubaro. Expansion of the global error for numerical schemes solving stochastic differential equations. *Stochastic Analysis and Applications*, 8(4): 483–509, 1990.

Yee Whye Teh, Alexandre H Thiéry, and Sebastian J Vollmer. Consistency and fluctuations for stochastic gradient Langevin dynamics. *Journal of Machine Learning Research*, 17(7):1–33, 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. Available at `https://arxiv.org/abs/1610.09787`, 2016.

Sebastian J Vollmer, Konstantinos C Zygalakis, et al. (Non-) asymptotic properties of stochastic gradient Langevin dynamics. *Journal of Machine Learning Research*, 17(159):1–48, 2016.

Stephen G Walker. Sampling the Dirichlet mixture model with slices. *Communications in Statistics*, 36(1):45–54, 2007.

Hanna M Wallach, Iain Murray, Ruslan Salakhutdinov, and David Mimno. Evaluation

methods for topic models. In *Proceedings of the 26th International Conference on Machine Learning*, pages 1105–1112, 2009.

Ming Chen Wang and George Eugene Uhlenbeck. On the theory of the Brownian motion ii. *Reviews of Modern Physics*, 17(2-3):323, 1945.

Xiangyu Wang and David B Dunson. Parallelizing MCMC via Weierstrass sampler. Available at `https://arxiv.org/abs/1312.4605`, 2013.

Max Welling and Yee W Teh. Bayesian learning via stochastic gradient Langevin dynamics. In *Proceedings of the 28th International Conference on Machine Learning*, pages 681–688, 2011.

David Williams. *Probability with Martingales*. Cambridge University Press, 1991.

Minjie Xu, Balaji Lakshminarayanan, Yee Whye Teh, Jun Zhu, and Bo Zhang. Distributed Bayesian posterior sampling via moment sharing. In *Advances in Neural Information Processing Systems 27*, pages 3356–3364. 2014.

Pan Xu, Jinghui Chen, Difan Zou, and Quanquan Gu. Global convergence of Langevin dynamics based algorithms for nonconvex optimization. In *Advances in Neural Information Processing Systems 31*, pages 3126–3137. 2018.

K. C. Zygalakis. On the existence and the applications of modified equations for stochastic differential equations. *SIAM Journal on Scientific Computing*, 33(1): 102–130, 2011.