Scalable Monte Carlo for Bayesian Learning

Paul Fearnhead, Christopher Nemeth, Chris J. Oates and Chris Sherlock

Contents

Preface 1

1 Background 4
1.1 Monte Carlo Methods 5
1.1.1 What is Monte Carlo Integration? 5
1.1.2 Importance Sampling 7
1.1.3 Monte Carlo or Quadrature? 7
1.1.4 Control Variates 9
1.1.5 Monte Carlo Integration and Bayesian Statistics 11
1.2 Example Applications 14
1.2.1 Logistic Regression 14
1.2.2 Bayesian Matrix Factorisation 15
1.2.3 Bayesian Neural Networks for Classification 16
1.3 Markov Chains 17
1.3.1 Reversible Markov chains 18
1.3.2 Convergence, Averages, and Variances 19
1.4 Stochastic Differential Equations 22
1.4.1 The Ornstein–Uhlenbeck Process 23
1.4.2 The Infinitesimal Generator 25
1.4.3 Langevin Diffusions 25
1.5 The Kernel Trick 28
1.5.1 Finite-Dimensional Inner Product Spaces 28
1.5.2 Kernels in a Finite-Dimensional Inner Product Space 31
1.5.3 A New Inner Product and the Kernel Trick in Finite Dimensions 33
1.5.4 General Kernels 35
1.5.5 The Power of the Kernel Trick 38
1.6 Chapter Notes 40

2 Reversible MCMC and its Scaling 42
2.1 The Metropolis–Hastings Algorithm 43
2.1.1 Component-wise updates and Gibbs moves 48
2.1.2 The Metropolis–Hastings Independence Sampler 49
2.1.3 The Random Walk Metropolis Algorithm 50

2.1.4 The Metropolis-Adjusted Langevin Algorithm 53
2.2 Hamiltonian Monte Carlo 57
2.3 Chapter Notes 63

3 Stochastic Gradient MCMC Algorithms 65
3.1 The Unadjusted Langevin Algorithm 65
3.2 Approximate vs. Exact MCMC 67
3.3 Stochastic Gradient Langevin Dynamics 69
3.3.1 Controlling Stochasticity in the Gradient Estimator 72
3.3.2 Example: The Value of Control Variates 78
3.3.3 Convergence Results for Stochastic Gradient Langevin Dynamics 80
3.4 A General Framework for stochastic gradient MCMC 85
3.5 Guidance for Efficient Scalable Bayesian Learning 90
3.5.1 Experiments on a Logistic Regression Model 92
3.5.2 Experiments on a Bayesian Neural Network Model 97
3.6 Generalisations and Extensions 100
3.6.1 Scalable Inference for Models in Constrained Spaces 100
3.6.2 Scalable Inference with Time Series Data 102
3.7 Chapter Notes 105

4 Non-Reversible MCMC 106
4.1 The Benefits of Non-Reversibility 106
4.2 Hamiltonian Monte Carlo Revisited 109
4.3 Lifting Schemes for MCMC 112
4.3.1 Non-Reversible HMC 112
4.3.2 Gustafson's Algorithm and Multidimensional Generalisations 113
4.4 Improving Non-reversibility: Delayed Rejection 119
4.4.1 The Discrete Bouncy Particle Sampler 121
4.5 Chapter Notes 124

5 Continuous-Time MCMC 126
5.1 Continuous-Time MCMC as the Limit of Non-Reversible MCMC 126
5.2 Piecewise Deterministic Markov Processes 129
5.2.1 What is a PDMP? 129
5.2.2 Simulating PDMPs 130
5.2.3 The Generator and Invariant Distribution of a PDMP 134
5.2.4 The Limiting Process of Section 5.1 as a PDMP 136
5.3 Continuous-time MCMC via PDMPs 139
5.3.1 Different Samplers 140
5.3.2 Use of PDMP Output 154
5.3.3 Comparison of Samplers 155
5.4 Efficient Simulation of PDMP Samplers 159
5.4.1 Simulating PDMPs 159
5.4.2 Exploiting Model Sparsity 165

5.4.3 Data Subsampling Ideas 168
5.5 Extensions 175
5.5.1 Discontinuous Target Distribution 176
5.5.2 Reversible Jump PDMP Samplers 179
5.5.3 More General Velocity Models 184
5.6 Chapter Notes 191

6 Assessing and Improving MCMC 192
6.1 Diagnostics for MCMC 192
6.1.1 Convergence Diagnostics 193
6.1.2 Bias Diagnostics 194
6.1.3 Improved Bias Diagnostics via the Kernel Trick 197
6.2 Convergence Bounds for MCMC 200
6.2.1 Bounds on Integral Probability Metrics 200
6.2.2 Choice of Auxiliary Markov Process 203
6.2.3 Kernel Stein Discrepancy 205
6.2.4 Convergence Control 210
6.2.5 Stochastic Gradient Stein Discrepancy 216
6.3 Optimal Weights for MCMC 218
6.4 Optimal Thinning for MCMC 222
6.5 Chapter Notes 224

References 227
Index 237

Preface

At the time of writing, science, industry, and society are being transformed by the emergence of a new generation of powerful machine learning and artificial intelligence (AI) methodologies. The safe use of such algorithms demands a probabilistic viewpoint, enabling reasoning in settings where data are noisy or limited, and endowing predictions with an appropriate degree of confidence for downstream decision-making and mitigation of risk. Yet, it remains true that fundamental probabilistic operations, such as conditioning on an observed dataset, are not easily performed at the scale required.

The aim of this book is to provide a graduate-level introduction to advanced topics in Markov chain Monte Carlo (MCMC), as applied broadly in the Bayesian computational context. Most, if not all of these topics (stochastic gradient MCMC, non-reversible MCMC, continuous time MCMC, and new techniques for convergence assessment) have emerged as recently as the last decade, and have driven substantial recent practical and theoretical advances in the field. A particular focus is on methods that are *scalable* with respect to either the amount of data, or the data dimension, motivated by the emerging high-priority application areas in machine learning and AI. Throughout this book, the clear presentation of ideas is prioritised over a rigorous technical treatment of all mathematical details; appropriate references for further reading are provided in the end-notes of each chapter. In particular, we will limit the use of measure theory; the reader should assume that all sets and functions are measurable with respect to an appropriate sigma-algebra, and all continuous distributions should be assumed to be absolutely continuous with respect to Lebesgue measure and all densities should be assumed to be densities with respect to Lebesgue measure.

This book has been indirectly shaped by the researchers and colleagues – too numerous to name individually – who have contributed to recent progress in the field. Special gratitude must go to Rebekah Fearnhead, Heishiro Kanagawa, Tamás Papp and Lorenzo Rimella, for proof-reading

the manuscript, to Richard Howey for typesetting the figures, and to Natalie Tomlinson and Anna Scriven for their encouragement and typesetting support. The authors are grateful for financial support from the Engineering and Physical Sciences Council (through grants EP/W019590/1, EP/R018561/1, EP/R034710/1, EP/V022636/1 and EP/Y028783/1), the Alan Turing Institute, and the Leverhulme Trust.

Paul Fearnhead
Christopher Nemeth
Chris Sherlock
Lancaster University, UK

Chris J. Oates
Newcastle University, UK

Common Notation

n	total number of iterations of an algorithm or Monte Carlo sample size.
d	dimension (of parameter space).
N	number of elements in the dataset.
$\mathcal{D}$	the dataset $\{\mathbf{y}_1, \ldots, \mathbf{y}_N\}$,.
$\boldsymbol{\theta}$	the parameter.
$L(\boldsymbol{\theta}; \mathcal{D})$	the likelihood function.
$\ell(\boldsymbol{\theta}; \mathcal{D})$	the log-likelihood function.
$\pi_0(\boldsymbol{\theta})$	the prior density.
$\pi(\boldsymbol{\theta}\|\mathcal{D})$	the posterior density, often abbreviated to $\pi(\boldsymbol{\theta})$.
$\mathbb{I}$	the indicator function.
$\mathbf{I}_d$	the $d \times d$ identity matrix.
x_i	the ith component of the vector $\mathbf{x}$.
$\mathbf{x}_k$	the kth vector in a sequence $(\mathbf{x}_k)_{k=1,2,\ldots}$.
$x_k^{(i)}$	the ith component of the vector $\mathbf{x}_k$.
i.i.d.	independent and identically distributed.
$\overset{\mathsf{D}}{=}$	equal in distribution.
$\overset{\mathsf{D}}{\rightarrow}$	converges in distribution.
$\delta_{\mathbf{x}}$	the Dirac distribution, which places all mass at $\mathbf{x}$.
$\mathsf{N}(\cdot; \boldsymbol{\mu}, \mathbf{V})$	the density of a normal random variable with mean $\boldsymbol{\mu}$ and covariance $\mathbf{V}$.
$\mathsf{U}_d(\cdot)$	the density for a uniform random variable on the d-dimensional sphere, $\mathbb{S}^{d-1} \subset \mathbb{R}^d$.
$\mathcal{L}^p(\pi)$	the set of measurable functions f with $\int \lvert f(\mathbf{x})\rvert^p \, \mathrm{d}\pi(\mathbf{x}) < \infty$.
$C^s(\mathbb{R}^d, \mathbb{R}^p)$	the set of functions $f : \mathbb{R}^d \rightarrow \mathbb{R}^p$ for which continuous derivatives exist of orders up to $s \in \{0, 1, \ldots\} \cup \{\infty\}$.
$x_n = O(a_n)$	there exist n_0 and M such that for all $n \geq n_0$, $\lvert x_n\rvert \leq M a_n$.
$X_n = O_p(a_n)$	for any $\epsilon > 0$ there exist n_0 and M such that for all $n \geq n_0$, $\mathbb{P}(\lvert X_n\rvert > M a_n) < \epsilon$.

1

Background

This book describes some recent developments in *scalable Monte Carlo* algorithms and their applications within Bayesian learning: what exactly does this mean?

Monte Carlo methods are a class of computational methods that involve repeated sampling to numerically approximate quantities of interest. We specifically focus on Monte Carlo integration methods, which are sampling-based methods for evaluating or approximating the value of integrals. Such methods are widely used across science and engineering, but our motivation comes particularly from Bayesian statistics. One of the key quantities in Bayesian statistics is the posterior distribution, which encapsulates our belief regarding unknown parameters of a model given our prior belief and an observed dataset. We can then obtain estimates of the parameters, or quantify our uncertainty about the parameters, in terms of expectations with respect to the posterior distribution. For example, a common estimate of a parameter is the posterior expectation of that parameter; the predictive probability of future observations is the expectation of the density/mass function of the future observation taken with respect to the posterior distribution. Calculating these expectations involves evaluating an integral, and the idea of Monte Carlo is to use samples from the posterior to estimate such integrals.

The main challenge with using Monte Carlo in Bayesian statistics is often in deriving an efficient algorithm to sample from the posterior distribution. Markov chain Monte Carlo is a general and widely-used class of methods for sampling from a distribution, based on simulating a Markov process that has the posterior distribution as its stationary distribution.

In recent years, there has been interest in applying Markov chain Monte Carlo to ever-increasingly complex and challenging problems. For example, the dimension, d say, of the parameter space of the models we wish to fit to data, or the number of data points, N say, in our data set can be large. As either d or N increases, the efficiency of Markov chain Monte Carlo

methods may reduce. For example, as d increases we may need to have more iterations of our Markov chain Monte Carlo algorithm to achieve the required level of accuracy, while as N increases, the computational cost per iteration of a standard algorithm will increase. *Scalable Markov chain Monte Carlo* methods are specifically those methods which can scale well as either or both of d and N increase.

The remainder of this introductory chapter will cover background relevant to scalable Markov chain Monte Carlo. The next section will introduce Monte Carlo methods, explain why Monte Carlo integration is widely-used, and explain how it is relevant to Bayesian statistics. This will be followed by an introduction to some of the statistical models and applications that will be used to demonstrate the methods in this book, as well as an informal and brief introduction to some of the concepts from stochastic processes that will be used in later chapters. Finally, the chapter ends with a short introduction to kernel methods in preparation for a deeper exposition in Chapter 6.

1.1 Monte Carlo Methods

1.1.1 What is Monte Carlo Integration?

Assume we have a distribution of interest. For simplicity of presentation, here and for the remainder of this chapter, we assume that the distribution is continuous on $\mathbb{R}^d$. Let $\mathbf{X}$ denote a random variable with this distribution and let $\pi(\mathbf{x})$ denote the corresponding probability density function for $\mathbf{X}$; we will also use π to refer to the distribution itself when that is necessary. Then the expectation of some function h of $\mathbf{X}$ is an integral

$$I = \mathbb{E}\left[h(\mathbf{X})\right] = \int h(\mathbf{x})\pi(\mathbf{x})\,\mathrm{d}\mathbf{x}.$$

This expectation is *well-defined*, that is, h is integrable with respect to π, if $\int |h(\mathbf{x})|\pi(\mathbf{x})\,\mathrm{d}\mathbf{x} < \infty$. We abbreviate this to $h \in \mathcal{L}^1(\pi)$ and throughout this section we assume that this holds true. If we can sample from $\pi(\cdot)$ then we can estimate this expectation/integral by (i) drawing n independent realisations, $\mathbf{x}_1, \ldots, \mathbf{x}_n$, from $\pi(\cdot)$ and (ii) calculating the sample average of the values $h(\mathbf{x}_1), \ldots, h(\mathbf{x}_n)$. This gives an estimate of I, namely

$$\hat{I} = \frac{1}{n}\sum_{k=1}^{n} h(\mathbf{x}_k).$$

This is called a *Monte Carlo* estimate of I, as it is obtained from independent, random samples from $\pi(\cdot)$.

The Monte Carlo estimator can be interpreted as being based on n independent random variables $\mathbf{X}_1, \ldots, \mathbf{X}_n$, of which $\mathbf{x}_1, \ldots, \mathbf{x}_n$ are realisations. Is it a good estimator? This is impossible to answer in generality, but we can at least describe some good properties that the estimator can admit. First, since the $\mathbf{X}_i$ are i.i.d, $\mathbb{E}\left[\hat{I}\right] = \mathbb{E}\left[h(\mathbf{X}_1)\right] = I$, so the estimator is *unbiased*. Secondly, and more importantly, the strong law of large numbers tells us that we can make our estimate arbitrarily accurate, with high probability, if we choose n large enough. Formally, provided I is well defined, that is $h \in \mathcal{L}^1(\pi)$, and our samples from $\pi(\cdot)$ are independent, then as $n \to \infty$,

$$\frac{1}{n}\sum_{k=1}^{n} h(\mathbf{X}_k) \to I \quad \text{almost surely.} \tag{1.1}$$

Almost sure convergence means that the collection of outcomes where the convergence does not occur has a combined probability of 0.

Thus, with high probability, our Monte Carlo estimator will be accurate if we choose n large enough, but the result does not tell us how large n needs to be, nor how accurate the estimator will be for a given value of n. However, provided that $\int h(\mathbf{x})^2\pi(\mathbf{x})\ \mathrm{d}\mathbf{x} < \infty$, which we abbreviate to $h \in \mathcal{L}^2(\pi)$, we can use the central limit theorem to answer these questions. Again assume that our samples from $\pi(\cdot)$ are independent, and define

$$V = \int \{h(\mathbf{x}) - I\}^2\pi(\mathbf{x})\ \mathrm{d}\mathbf{x}.$$

Then, the central limit theorem states that

$$\sqrt{n}\left(\frac{\frac{1}{n}\sum_{k=1}^{n} h(\mathbf{X}_k) - I}{\sqrt{V}}\right) \xrightarrow{\mathsf{D}} \mathsf{N}(0, 1),$$

as $n \to \infty$. Here the convergence is in distribution, and we have convergence to a standard normal distribution in the limit.

One way of interpreting this result is that, for large enough n, approximately,

$$\frac{1}{n}\sum_{k=1}^{n} h(\mathbf{X}_k) \sim \mathsf{N}\left(I, \frac{V}{n}\right).$$

That is our estimator will be approximately normally distributed, with mean equal to the integral, I, and a variance that is V/n. This shows that the quantity V governs how easy it is to estimate I via Monte Carlo integration, and the accuracy depends on both V and n. The order of the error of a Monte Carlo estimator is $\sqrt{V/n}$ and, thus, the Monte Carlo error decays with sample size at a rate of $n^{-1/2}$.

1.1.2 Importance Sampling

What if we are interested in calculating or approximating a more general integral, $I = \int_\Omega g(\mathbf{x})\mathrm{d}\mathbf{x}$, of some function g over a region Ω? We can use Monte Carlo sampling to estimate this integral by re-writing the integral as an expectation with respect to some density function $q(\cdot)$ defined on Ω as follows,

$$I = \int_\Omega \frac{g(\mathbf{x})}{q(\mathbf{x})} q(\mathbf{x})\ \mathrm{d}\mathbf{x} = \mathbb{E}\left[h(\mathbf{X})\right],$$

where $h(\mathbf{x}) = g(\mathbf{x})/q(\mathbf{x})$. If I is well-defined, that is $\int |g(\mathbf{x})|\mathrm{d}\mathbf{x} < \infty$, then $h \in \mathcal{L}^1(q)$, and I can be estimated using Monte Carlo integration as above, based on independent realised samples $\mathbf{x}_1, \ldots, \mathbf{x}_n$ from $q(\cdot)$ by calculating the arithmetic mean of $h(\mathbf{x}_1), \ldots, h(\mathbf{x}_n)$. This process is called *Importance Sampling*, and $q(\cdot)$ is known as the proposal distribution.

For this Monte Carlo estimator to be feasible, we have two constraints on q. First, we need $q(\mathbf{x}) > 0$ whenever $g(\mathbf{x}) > 0$, in order for $h(\mathbf{x})$ to be well-defined. Second, we need to be able to easily sample from $q(\cdot)$. The choice of $q(\cdot)$ will affect the accuracy of the estimator, with the variance of our estimator for a Monte Carlo sample of size n being V/n where

$$V = \int \left(\frac{g(\mathbf{x})}{q(\mathbf{x})} - I\right)^2 q(\mathbf{x})\ \mathrm{d}\mathbf{x}.$$

This variance will be small if $g(\mathbf{x})/q(\mathbf{x})$ is roughly constant, and one can show that the optimal choice of $q(\cdot)$ in terms of minimising V is $q(\mathbf{x}) \propto |g(\mathbf{x})|$. If $g(\mathbf{x})$ is non-negative everywhere (or non-positive everywhere) then such a choice of q will give an estimator that has zero-variance, that is an exact estimator. More generally the variance V will be large if there are values of $\mathbf{x}$ for which $g(\mathbf{x})/q(\mathbf{x})$ is large. This leads to a rule-of-thumb that, if Ω is unbounded, one wants $q(\mathbf{x})$ to have heavier tails than $|g(\mathbf{x})|$ to avoid this ratio blowing up as $\|\mathbf{x}\| \to \infty$.

1.1.3 Monte Carlo or Quadrature?

It is natural to ask why one should use Monte Carlo integration when there are alternative numerical integration methods, such as quadrature. To see the potential benefits of Monte Carlo methods, consider estimating an integral on the unit hyper-cube $[0, 1]^d$. We can then compare quadrature with Monte Carlo integration based on samples from a uniform distribution on $[0, 1]^d$.

First, consider $d = 1$. In this case, quadrature methods tend to be much

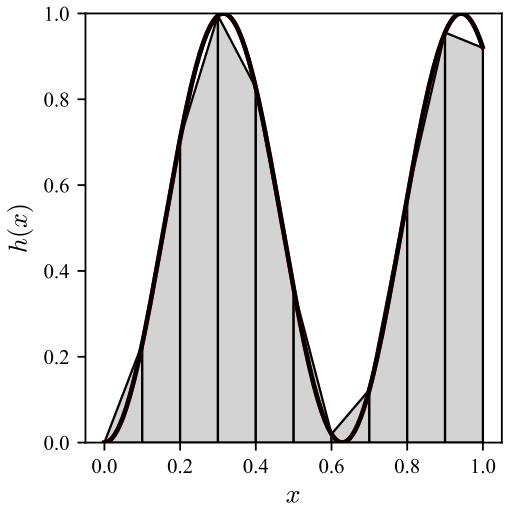

Figure 1.1 Example of trapezoid rule. We can estimate the integral, by (i) setting $x_1, \ldots, x_n$ to be evenly spaced points on $[0, 1]$; (ii) creating $n - 1$ trapezoids based on joining up the points $(x_k, h(x_k))$ (shaded in regions); and (iii) estimating the integral by the sum of the areas of the trapezoids.

more accurate than Monte Carlo methods. We have seen that the Monte Carlo variance, if we have n Monte Carlo samples, is $O(1/n)$, which means that the error of our Monte Carlo estimator will be $O_p(n^{-1/2})$.

By comparison, a simple numerical method is the trapezoidal rule. This involves evaluating the integrand, $h(x)$ at a set of equally spaced points, $x_1, \ldots, x_n$, on $[0, 1]$, and approximating the integral using the total area of the trapezoids formed by joining up the points $(x_k, h(x_k))$ for $k = 1, \ldots, n$, see Figure 1.1. Assuming our integrand has a bounded second derivative $|h''(x)| < L$ for some L, then we can bound the error in the estimate of the integral as $L\delta^2/12$, where $\delta = 1/(n-1)$ is the width of each trapezoid. This gives an error that decays like $O(1/n^2)$, which is much better than the Monte Carlo method. Furthermore, higher-order quadrature methods, such as Simpson's rule, can obtain even faster decay of the approximate error with n, if the integrand is sufficiently smooth.

So, quadrature methods can be more accurate for 1-dimensional integrals, at least for functions whose second derivatives are bounded. However,

now consider higher-dimensional integrals involving functions $h(\mathbf{x})$, the only information about which we have is that the second-order (partial) derivatives are bounded. Then we can apply a cubature rule based on a grid of $m + 1$ equally spaced points in each dimension. The spacing of these points will be $\delta = 1/m$ and there will be $n = (m + 1)^d$ points in total. If we have a cubature whose error decays like δ^r, for some power r, for example, $r = 2$ for the trapezoidal rule, then the error decays at a rate of $m^{-r} \approx n^{-r/d}$. For large d, this convergence will be slower than the $n^{-1/2}$ rate of Monte Carlo integration, explaining why Monte Carlo is often the default method for numerically approximating high-dimensional integrals. To overcome this *curse of dimension* in cubature, it is usually necessary to identify a sense in which the integrand $h(\mathbf{x})$ is effectively low-dimensional, which can be difficult or impossible depending on the applied context.

1.1.4 Control Variates

Let us return to the problem of estimating the expectation of some function of a random variable,

$$I = \mathbb{E}\left[h(\mathbf{X})\right] = \int h(\mathbf{x})\pi(\mathbf{x})\,\mathrm{d}\mathbf{x},$$

where $\pi(\mathbf{x})$ is the density of $\mathbf{X}$. We have seen how we can estimate this using a sample from $\pi(\cdot)$, and that the accuracy of this estimator is proportional to

$$V = \int \left\{h(\mathbf{x}) - I\right\}^2 \pi(\mathbf{x})\,\mathrm{d}\mathbf{x} = \int h(\mathbf{x})^2\pi(\mathbf{x})\,\mathrm{d}\mathbf{x} - I^2.$$

The latter expression is just the standard expression for the variance of $h(\mathbf{X})$. This shows that it is easier to estimate expectations of functions that vary less when evaluated at $\mathbf{X}$.

Assume that we know the expectation of a set of random variables $g_1(\mathbf{X}), \ldots, g_J(\mathbf{X})$, each a transformation of $\mathbf{X}$. Without loss of generality, we can assume that these random variables have mean zero, *i.e.*,

$$\mathbb{E}\left[g_j(\mathbf{X})\right] = 0, \quad \text{for } j = 1, \ldots, J,$$

as, if this is not the case, we can define new random variables equal to the old random variables minus their expectations. Then, for any constants $\gamma_1, \ldots, \gamma_n$,

$$I = \mathbb{E}\left[h(\mathbf{X})\right] - \sum_{j=1}^{J} \gamma_j \mathbb{E}\left[g_j(\mathbf{X})\right] = \mathbb{E}\left[h(\mathbf{X}) - \sum_{j=1}^{J} \gamma_j g_J(\mathbf{X})\right]. \tag{1.2}$$

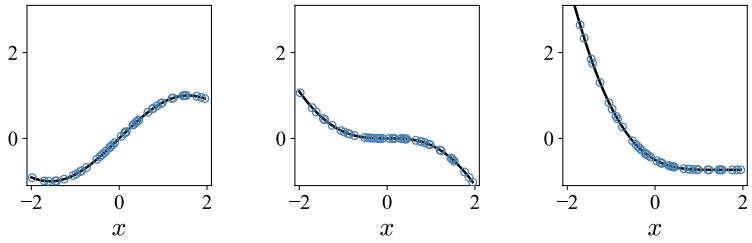

Figure 1.2 Example of control variates for estimating $\mathbb{E}[\sin(X)]$, where X has a standard normal distribution $\mathsf{N}(0, 1)$. Each plot shows the function whose expectation is being estimated and 50 values used in the Monte Carlo estimate (dots). From left to right the functions are respectively: $h(x) = \sin(x)$, $h(x) = \sin(x) - x$, and $h(x) = \sin(x) - \pi x/2 + (x^2 - 1)/2$. The expectation of each function is constructed to be the same. The effect of introducing control variates in the middle and right-hand plot is to flatten out the function we are integrating – in the middle plot, this happens for $x \approx 0$ and for the right-hand plot for $x \approx \pi/2$. The variability of the function values, i.e. the dots, is smallest for the middle plot and largest for the right-hand plot.

By suitable choice of the constants $\gamma_1, \ldots, \gamma_J$, the variability of the random variable $h(\mathbf{X}) - \sum_{j=1}^{J} \gamma_j g_j(\mathbf{X})$ can be made smaller than that of $h(\mathbf{X})$, and thus a Monte Carlo estimate of I based on (1.2) will have smaller Monte Carlo variance than the basic Monte Carlo estimator. We call $\sum_{j=1}^{J} \gamma_j g_j(\mathbf{X})$ a *control variate* for $h(\mathbf{X})$. Heuristically, we want to choose $\gamma_1, \ldots, \gamma_J$ so that $h(\mathbf{X}) \approx \gamma_0 + \sum_{j=1}^{J} \gamma_j g_j(\mathbf{X})$, which means that $h(\mathbf{X}) - \sum_{j=1}^{J} \gamma_j g_j(\mathbf{X})$ is approximately constant.

As a simple example, consider estimating the expectation of $\sin(X)$ where X has a standard normal distribution $\mathsf{N}(0, 1)$. We know that this expectation is 0 as the distribution of X is symmetric about 0 and $\sin(-x) = -\sin(x)$. We will compare the simple Monte Carlo estimator of the expectation with estimates using control variates with the functions $g_1(x) = x$ and $g_2(x) = x^2 - 1$. By using a Taylor expansion of $\sin(x)$ at $x = 0$ we have $\sin(x) \approx x$ for small x, and thus a simple choice of control variate is $g_1(x)$.

We show pictorially the benefit of using this control variate in Figure 1.2, where we see that $\sin(x) - x \approx 0$ for most x values sampled from the standard normal distribution. This reduces the Monte Carlo variance of the estimate of $\mathbb{E}[h(X)]$ by close to a factor of 2.

Care must be taken with control variates, however. For example, if we

perform a Taylor expansion of $\sin(x)$ at $x = \pi/2$ we get $\sin(x) \approx 1 - (x - \pi/2)^2/2$, which suggests using $-g_2(x)/2 + \pi g_1(x)/2$ as a control variate. However, this choice leads to an increase in the Monte Carlo variance by over a factor of 3. Figure 1.2 shows that the function $\sin(x) - \pi x/2 + (x^2 - 1)/2$ is roughly constant for $x \approx \pi/2$, but overall it is more variable across the range $x \in [-2, 2]$, where most of the probability mass of $\mathsf{N}(0, 1)$ lies.

This example shows that the choice of $\gamma_1, \ldots, \gamma_J$ is important when using control variates. In some situations, there may be a natural way of choosing these – for example, based on a Taylor expansion of the function of interest around the mode of the distribution of $\mathbf{X}$. However, it is also possible to choose these values based on simulation. Ideally, we would choose $\gamma_1, \ldots, \gamma_J$ to minimise the Monte Carlo variance

$$\int \left\{ h(\mathbf{x}) - \sum_{j=1}^{J} \gamma_j g_j(\mathbf{x}) \right\}^2 \pi(\mathbf{x}) \, \mathrm{d}\mathbf{x} - I^2,$$

and we can obtain a Monte Carlo estimate of this. If $\mathbf{x}_1, \ldots, \mathbf{x}_n$ are realised samples from $\pi(\cdot)$, then we can choose $\gamma_1, \ldots, \gamma_J$ to minimise

$$\sum_{k=1}^{n} \left(h(\mathbf{x}_k) - \sum_{j=1}^{J} \gamma_j g_j(\mathbf{x}_k) \right)^2,$$

which just involves minimising a sum of squares criterion. If we let $\mathbf{h}$ be the $n \times 1$ vector whose ith entry is $h(\mathbf{x}_i)$, $\boldsymbol{\gamma}$ be the $J \times 1$ vector whose ith entry is γ_i, and $\mathbf{Z}$ be the $n \times J$ matrix whose (i, j)th entry is $g_j(\mathbf{x}_i)$, then, assuming $\mathbf{Z}$ is of full rank, the least-squares estimate of $\boldsymbol{\gamma}$ is

$$\hat{\boldsymbol{\gamma}} = (\mathbf{Z}^\top \mathbf{Z})^{-1} \mathbf{Z}^\top \mathbf{h}.$$

These estimates $\hat{\boldsymbol{\gamma}}$ depend on the Monte Carlo samples, and thus for the Monte Carlo estimate of I to be unbiased we need to use a new set of Monte Carlo samples from $\mathbf{X}$ for estimating I using the $\hat{\boldsymbol{\gamma}}$.

While we have presented the idea of control variates for estimating expectations of functions, similar ideas can be used with importance sampling for estimating general integrals.

1.1.5 Monte Carlo Integration and Bayesian Statistics

One of the most important applications of Monte Carlo methods occurs within Bayesian statistics. To explain why, consider the problem of making inferences, from data, about the parameter of a statistical model. We will use the notation $\mathcal{D}$ to denote data in general. In some situations, we will

need to distinguish individual data points, and in those settings, we will assume $\mathcal{D} = \{\mathbf{y}_1, \ldots, \mathbf{y}_N\}$, with $\mathbf{y}_i$ being the ith data point and N being the size of our dataset.

We further assume that we have a model for the data. Let the model depend on a parameter $\boldsymbol{\theta}$, and denote the likelihood of the data under our model by $L(\boldsymbol{\theta}; \mathcal{D})$. The likelihood is the probability, or probability density, of observing data $\mathcal{D}$ under our model if the parameter is $\boldsymbol{\theta}$. In Bayesian statistics, we represent beliefs, or uncertainty, about the parameter, $\boldsymbol{\theta}$, through probability distributions. Our beliefs about $\boldsymbol{\theta}$ before seeing the data are given by a prior, $\pi_0(\boldsymbol{\theta})$, and, once we observe data, Bayes' Theorem provides the update to the posterior distribution:

$$\pi(\boldsymbol{\theta} \mid \mathcal{D}) \propto \pi_0(\boldsymbol{\theta})\, L(\boldsymbol{\theta}; \mathcal{D})\,. \tag{1.3}$$

Where it will not cause confusion, we may drop the explicit conditioning on the data in the posterior, and write $\pi(\boldsymbol{\theta})$ rather than $\pi(\boldsymbol{\theta} \mid \mathcal{D})$.

Assuming the correctness of our model, the posterior distribution contains all information about the parameter, $\boldsymbol{\theta}$, that can be logically deduced from our prior belief and the dataset. From it, we can then obtain a point estimate for $\boldsymbol{\theta}$, such as its posterior mean, and quantify uncertainty in terms of the posterior probability of $\boldsymbol{\theta}$ lying in a given set of values. However, in most applications, the posterior distribution is intractable, meaning that it cannot be explicitly calculated. The central challenge is that the posterior density $\pi(\boldsymbol{\theta} \mid \mathcal{D})$ is known, via Bayes' Theorem, only up to a normalising constant.

The intractability of the posterior distribution is a key motivator for Monte Carlo methods. If we can draw samples from $\pi(\boldsymbol{\theta} \mid \mathcal{D})$, then we can obtain simple, and often accurate, Monte Carlo estimates of posterior quantities of interest. Given realisations $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_n$ sampled from $\pi(\boldsymbol{\theta} \mid \mathcal{D})$, and a function $h(\boldsymbol{\theta})$ whose expectation

$$I := \mathbb{E}_\pi\left[h(\boldsymbol{\theta})\right] = \int h(\boldsymbol{\theta})\pi(\boldsymbol{\theta} \mid \mathcal{D})\, \mathrm{d}\boldsymbol{\theta}$$

is of interest, define

$$\widehat{\mu}_h^{(n)} := \frac{1}{n}\sum_{k=1}^{n} h(\boldsymbol{\theta}_k). \tag{1.4}$$

As mentioned earlier, for any function $h \in \mathcal{L}^1(\pi)$, the strong law of large numbers (1.1) tells us that we can estimate $\mathbb{E}_\pi\left[h(\boldsymbol{\theta})\right]$ as accurately as we desire using Monte Carlo integration, and provided enough samples

are taken: $\widehat{\mu}_h^{(n)} \to \mathbb{E}_\pi [h(\boldsymbol{\theta})]$ almost surely as $n \to \infty$. Moreover, if $h \in \mathcal{L}^2(\pi)$ then the central limit theorem states that the Monte Carlo error, $\widehat{\mu}_h^{(n)} - \mathbb{E}_\pi [h(\boldsymbol{\theta})]$ is $O_p(n^{-1/2})$.

For example, the vector of posterior means can be estimated by

$$\hat{\boldsymbol{\theta}} = \frac{1}{n} \sum_{k=1}^{n} \boldsymbol{\theta}_k,$$

and the posterior probability of $\boldsymbol{\theta} \in \mathcal{B}$ for some set $\mathcal{B}$ can be estimated by the proportion of Monte Carlo samples in $\mathcal{B}$

$$\hat{\mathbb{P}} (\boldsymbol{\theta} \in \mathcal{B}) = \frac{1}{n} \sum_{k=1}^{n} \mathbb{I} \{\boldsymbol{\theta}_k \in \mathcal{B}\},$$

where $\mathbb{I} \{\boldsymbol{\theta}_k \in \mathcal{B}\}$ is the indicator function of the event $\boldsymbol{\theta}_k \in \mathcal{B}$.

The challenge with this Monte Carlo approach to Bayesian statistics is the difficulty in sampling from $\pi(\boldsymbol{\theta})$, particularly if $\boldsymbol{\theta}$ is high-dimensional. Of the Monte Carlo integration methods we have mentioned so far, importance sampling offers an alternative when we are unable to sample from π directly. Consider estimating the posterior expectation for some function $h(\boldsymbol{\theta})$, so $h(\boldsymbol{\theta}) = \boldsymbol{\theta}$ would give us the posterior mean of $\boldsymbol{\theta}$ and $h(\boldsymbol{\theta}) = \mathbb{I} \{\boldsymbol{\theta} \in \mathcal{B}\}$ would give us the posterior probability of $\boldsymbol{\theta} \in \mathcal{B}$. Let $q(\boldsymbol{\theta})$ be a proposal distribution with the same support as the posterior. Then we have

$$\mathbb{E} [h(\boldsymbol{\theta}) \mid \mathcal{D}] = \int h(\boldsymbol{\theta}) \pi(\boldsymbol{\theta}) \, \mathrm{d}\boldsymbol{\theta} = \int \frac{h(\boldsymbol{\theta}) \pi(\boldsymbol{\theta})}{q(\boldsymbol{\theta})} q(\boldsymbol{\theta}) \, \mathrm{d}\boldsymbol{\theta}.$$

It is common to define *weights* $w(\boldsymbol{\theta}) := \pi(\boldsymbol{\theta}) / q(\boldsymbol{\theta})$. Then given an independent sample $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_n$ from $q(\boldsymbol{\theta})$, we can estimate the posterior expectation by the importance sampling estimator

$$\frac{1}{n} \sum_{k=1}^{n} w(\boldsymbol{\theta}_k) h(\boldsymbol{\theta}_k).$$

There are two issues with this estimator. The first is that as we only know the posterior up to a constant of proportionality, we only know the weights up to a constant of proportionality. However, the constant of proportionality can be estimated by setting $h(\boldsymbol{\theta}) = 1$, whence $\mathbb{E} [h(\boldsymbol{\theta})] = 1$ as the expectation of a constant is the constant. Thus we can use the unnormalised posterior density in the definition of the weights, and estimate the normalising constant as $(1/n) \sum_{k=1}^{n} w(\boldsymbol{\theta}_k)$. The posterior expectation of $h(\boldsymbol{\theta})$ is then estimated as

$$\sum_{k=1}^{n} \frac{w(\boldsymbol{\theta}_k)}{\sum_{j=1}^{n} w(\boldsymbol{\theta}_j)} h(\boldsymbol{\theta}_k),$$

which requires knowing the posterior density only up to a constant of proportionality. Often we define normalised weights $w^*(\boldsymbol{\theta}_k) = w(\boldsymbol{\theta}_k)/\sum_{j=1}^n w(\boldsymbol{\theta}_j)$, and we can then view the weighted samples $(\boldsymbol{\theta}_k, w^*(\boldsymbol{\theta}_k))$, for $k = 1, \ldots, n$, as a discrete approximation to the posterior.

The second issue is that the Monte Carlo variances of our estimators of posterior expectations depend on the variability of the weights. Often this will be large if $\boldsymbol{\theta}$ is high-dimensional. To see this, consider a toy example where the posterior has independent components. Assume each component is normal with mean 0 and variance σ^2, and the importance-sampling proposal distribution is also independent over components, but with a standard normal distribution, *i.e.*, with mean zero and unit variance, for each component. The importance sampling weight for a realisation $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_d)$ is

$$w(\boldsymbol{\theta}) = \sigma^{-d} \exp\left\{\frac{\sigma^2 - 1}{2\sigma^2} \sum_{i=1}^{d} \theta_i^2\right\}.$$

Now $\sum_{i=1}^d \theta_i^2$ has a χ_d^2 distribution under the proposal, and using the moment generating function of a χ_d^2 distribution, we obtain the Monte Carlo variance of the weight:

$$\mathrm{var}\{w(\boldsymbol{\theta})\} = \sigma^{-d}\left(2 - \sigma^2\right)^{-d/2} - 1.$$

Writing $\sigma^2 = 1 + \epsilon$, for some $\epsilon > 0$, this variance is $(1/\sqrt{1-\epsilon^2})^d - 1$, which increases exponentially with d. The focus of Markov chain Monte Carlo (MCMC) methods that we introduce in the next chapter is to produce sampling algorithms that avoid this exponential curse of dimensionality.

1.2 Example Applications

In later chapters, we will demonstrate the Monte Carlo methods on some example models which we now introduce. Whilst these models are somewhat simple to describe, their posteriors exhibit many of the features of more challenging posterior distributions, in particular, with respect to scalable sampling.

1.2.1 Logistic Regression

Logistic regression models the relationship between a binary response and a set of covariates. Denote the responses by $y_1, \ldots, y_N$ and the covariates by d-dimensional vectors $\mathbf{x}_1, \ldots, \mathbf{x}_N$. Then, logistic regression models the

data (the responses) as conditionally independent, given a d-dimensional parameter $\boldsymbol{\theta}$ and the covariates, and that

$$\mathbb{P}\left(Y = y_j | \mathbf{x}_j, \boldsymbol{\theta}\right) = \frac{\exp\{y_j \mathbf{x}_j^\top \boldsymbol{\theta}\}}{1 + \exp\{\mathbf{x}_j^\top \boldsymbol{\theta}\}}.$$

An intercept term can be included in the model by setting the first coordinate of each of $\mathbf{x}_1, \ldots, \mathbf{x}_N$ to be 1.

Our interest will be in sampling from the posterior distribution of $\boldsymbol{\theta}$. To define the posterior, we need to specify a prior $\pi_0(\boldsymbol{\theta})$, and we will assume that our prior is Gaussian with mean $\mathbf{0}$ and variance $\boldsymbol{\Sigma}_{\boldsymbol{\theta}}$. This leads to a posterior distribution, $\pi(\boldsymbol{\theta}|\mathcal{D})$, which can be written succinctly up to a multiplicative constant as

$$\pi(\boldsymbol{\theta}|\mathcal{D}) \propto \exp\left\{-\frac{1}{2}\boldsymbol{\theta}^\top \boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{-1} \boldsymbol{\theta}\right\} \prod_{j=1}^{N} \frac{\exp\{y_j \mathbf{x}_j^\top \boldsymbol{\theta}\}}{1 + \exp\{\mathbf{x}_j^\top \boldsymbol{\theta}\}}.$$

This is a canonical, albeit relatively simple, test problem for sampling methodologies. When we consider sampling methods for this model, we will drop the explicit conditioning on data $\mathcal{D}$ and use $\pi(\boldsymbol{\theta})$ to denote the target distribution of the sampler. The samplers we consider will often use gradient information about their target distribution, and we have

$$\frac{\partial \log \pi(\boldsymbol{\theta})}{\partial \theta_i} = -\left[\boldsymbol{\theta}^\top \boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{-1}\right]_i + \sum_{j=1}^{N} x_j^{(i)} \left\{ y_j - \frac{\exp\{\mathbf{x}_j^\top \boldsymbol{\theta}\}}{1 + \exp\{\mathbf{x}_j^\top \boldsymbol{\theta}\}} \right\}, \tag{1.5}$$

where $x_j^{(i)}$ indicates the ith component of $\mathbf{x}_j$.

1.2.2 Bayesian Matrix Factorisation

Bayesian matrix factorisation attempts to find a representation of a high-dimensional matrix as the product of two lower-dimensional matrices. Consider an $n \times m$ matrix $\mathbf{Y}$, and let $\boldsymbol{\theta} = \{\mathbf{U}, \mathbf{V}\}$ where $\mathbf{U}$ and $\mathbf{V}$ are $n \times d$ and $d \times m$ matrices respectively. Then the approximation is $\mathbf{Y} \approx \mathbf{UV}$. If $d \ll \min\{m, n\}$ then this can lead to a substantial reduction in dimension, and the model can be viewed as attempting to find low-dimensional structure in $\mathbf{Y}$.

The interpretation of this model is that each row of $\mathbf{V}$ is a *factor*, and we aim to approximate each row of $\mathbf{Y}$ as a linear combination of these factors. The entries in $\mathbf{U}$ are called *factor loadings*, and give the relative weight of each factor for each row of $\mathbf{Y}$.

One common approach to fitting these models is to use a Gaussian working model, thus up to additive constants, the log-likelihood is

$$L(\boldsymbol{\theta}; \mathcal{D}) = -nm \log \sigma - \frac{1}{2\sigma^2} \left\{ \sum_{i=1}^{n} \sum_{j=1}^{m} \left(Y_{i,j} - \sum_{k=1}^{d} U_{i,k} V_{k,j} \right)^2 \right\},$$

where σ^2 is the variance of the difference between entries of $\mathbf{Y}$ and $\mathbf{UV}$. In Bayesian matrix factorisation, we then introduce a prior on the parameters $\mathbf{U}$ and $\mathbf{V}$. Often, the prior for each entry is Gaussian, or is a mixture of a Gaussian and a point-mass at zero, as this encourages sparsity in the factors which potentially aids the interpretation of $\mathbf{U}$ and $\mathbf{V}$. It is also possible to introduce a prior over the number of factors, d, with the priors for the entries of $\mathbf{U}$ and $\mathbf{V}$ potentially depending on d.

1.2.3 Bayesian Neural Networks for Classification

Artificial neural networks are a flexible and popular class of models used in machine learning for solving supervised learning problems, such as regression and classification tasks. In the case of classification, assume that $y_1, y_2, \ldots, y_N$ are observed data, where each y_j represents one of G classes, i.e. $y_j \in \{1, 2, \ldots, G\}$. Assuming $d-$dimensional vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ for the covariates, then under a simple two-layer neural network model, the probability of a particular class y_j is

$$\mathbb{P}(Y = y_j | \mathbf{x}_j, \boldsymbol{\theta}) \propto \exp(\mathbf{A}_{y_j}^\top \sigma(\mathbf{B}^\top \mathbf{x}_j + \mathbf{b}) + a_{y_j}), \tag{1.6}$$

where $\mathbf{b}$ is a d_h-dimensional vector, $\mathbf{B}$ is a $d \times d_h$ matrix, with d_h the dimension of the variables in the hidden layer. The function $\sigma : \mathbb{R}^{d_h} \to (0,1)^{d_h}$ is a vector softmax function with $\sigma(\mathbf{z})_i = \exp(z_i)/\{\sum_{j=1}^{d_h} \exp(z_j)\}$ for $i = 1, \ldots, d_h$. The notation $\mathbf{A}_i$ refers to the i-th column of the $d_h \times G$ matrix $\mathbf{A}$. The parameters of the model $\boldsymbol{\theta} = \text{vec}(\mathbf{a}, \mathbf{A}, \mathbf{b}, \mathbf{B})$ are represented by vectors $\mathbf{a}, \mathbf{b}$, commonly referred to as *biases*, and matrices $\mathbf{A}, \mathbf{B}$, which are commonly referred to as *weights*.

Taking a Bayesian approach to parameter estimation, we can place independent Gaussian priors on each of the elements of the biases and weights in $\boldsymbol{\theta}$. Monte Carlo algorithms can be used to sample from the posterior,

$$\pi(\boldsymbol{\theta} | \mathcal{D}) \propto \pi_0(\boldsymbol{\theta}) \prod_{j=1}^{N} \mathbb{P}(y_j | \mathbf{x}_j, \boldsymbol{\theta}). \tag{1.7}$$

For Bayesian neural network models, the dataset sizes tend to be very

large and approximating the posterior distribution requires Monte Carlo methods which are scalable to large datasets. In Chapter 3, we will use stochastic gradient Markov chain Monte Carlo algorithms to approximate the Bayesian neural network posterior distribution.

1.3 Markov Chains

This section describes discrete-time Markov chains, focusing on the concepts that will be required to understand the Markov chain Monte Carlo method and its efficiency: the stationary distribution, reversibility, convergence to the stationary distribution, ergodic averages, integrated autocorrelation time and effective sample size.

Definition 1.1 A *discrete-time Markov chain* on a state space $\mathcal{X}$ is a collection of random variables $\{X_k\}_{k=0}^{\infty}$ with each $X_k \in \mathcal{X}$, such that for any $\mathcal{A} \subseteq \mathcal{X}$,

$$\mathbb{P}\left(X_{k+1} \in \mathcal{A} \mid X_k = x_k, \ldots, X_0 = x_0\right) = \mathbb{P}\left(X_{k+1} \in \mathcal{A} \mid X_k = x_k\right); \quad (1.8)$$

conditional on the current state, the distribution of the next state is independent of all previous states.

In this chapter, we will only consider *homogeneous* Markov chains, where the distribution of the next state given the current state does not depend on the value of k. Such a chain has a stationary distribution, ν, if $X_k \sim \nu \implies X_{k+1} \sim \nu$. If the chain also has a unique limiting distribution, then this must be ν since, by repeated induction, if $X_j \sim \nu$ then $X_k \sim \nu$ for all $k > j$, including as $k \to \infty$.

The following two examples of Markov chains on the vertices of an m-sided polygon illustrate different ways that a chain can be stationary. We label the vertices of the polygon from 0 to $m-1$, increasing in a clockwise direction; thus, $\mathcal{X} = \{0, 1, \ldots, m-1\}$.

Example 1.2 (See Figure 1.3, left.) Let $\{X_k\}_{k=0}^{\infty}$ be a Markov chain on the vertices of an m-sided polygon where the state at time $k+1$ is obtained from the state at time k by moving to the next vertex in a clockwise direction. If at time k the chain is equally likely to be at each of the vertices, then this is still the case at time $k+1$. The stationary distribution has $\mathbb{P}(X_k = x) = 1/m$ for $x \in \mathcal{X}$.

Example 1.3 (See Figure 1.3, right.) Let $\{X_k\}_{k=0}^{\infty}$ be a Markov chain on the vertices of an n-sided polygon where the state at time $k+1$ is obtained from the state at tim.e k by performing one of the following moves, each of

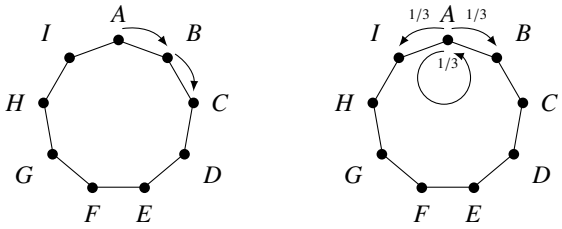

Figure 1.3 9-sided polygon where the Markov chain only moves clockwise (left figure), as in Example 1.2 or moves either a clockwise or anti-clockwise direction with probability $1/3$ (right figure), as in Example 1.3.

which has a probability of $1/3$: move to the next vertex in an anti-clockwise direction; do not move; move to the next vertex in a clockwise direction. As with Example 1.2 the stationary distribution has $\mathbb{P}(X_k = x) = 1/m$ for $x \in \mathcal{X}$.

1.3.1 Reversible Markov chains

Example 1.2 has a clear flow in a clockwise direction and, because of this, is an example of a non-reversible Markov chain; these will be discussed in detail in Chapter 4. By contrast, in Example 1.3, consider any two adjacent vertices: at stationarity, the probability of being at the first and moving to the second is the same as the probability of being at the second and moving to the first. Indeed, this is true of any pair of vertices, with the probability being 0 if they are not adjacent. This is an example of a reversible Markov chain.

Definition 1.4 A Markov chain $\{X_k\}_{k=1}^{\infty}$ with a state space of $\mathcal{X}$ is *reversible* with respect to a distribution ν when, for any two sets $\mathcal{B}, C \subseteq \mathcal{X}$, if $X_k \sim \nu$ then $\mathbb{P}(X_k \in \mathcal{B}, X_{k+1} \in C) = \mathbb{P}(X_k \in C, X_{k+1} \in \mathcal{B})$.

Consider the decomposition

$$\mathbb{P}(X_k \in \mathcal{B}, X_{k+1} \in C) = \mathbb{P}(X_k \in \mathcal{B})\,\mathbb{P}(X_{k+1} \in C | X_k \in \mathcal{B}).$$

The first term on the right-hand side is the amount of probability mass in $\mathcal{B}$ at time k and the second term is the fraction of that mass which moves to C at time $k+1$, so the product is the amount of probability mass moving from $\mathcal{B}$ to C. If the chain is reversible with respect to ν and $X_k \sim \nu$, then this is

also the amount of mass moving from C to $\mathcal{B}$. Given this balance, referred to as *detailed balance*, we would expect the total amount of probability mass in any set to remain constant. Indeed, setting $C = \mathcal{X}$ in Definition 1.4, we see that reversibility implies that if $X_k \sim \nu$, $\mathbb{P}(X_k \in \mathcal{B}) = \mathbb{P}(\mathcal{X}_{k+1} \in \mathcal{B})$. Since this is also true for all $\mathcal{B}$, $X_{k+1} \sim \nu$.

1.3.2 Convergence, Averages, and Variances

In Example 1.3, whatever the value or distribution of X_0, as $k \to \infty$ the distribution of X_k converges to the stationary distribution. For simplicity of presentation, we show this when $m = 2m' + 1$ is odd. For all $x_0, x \in \mathcal{X}$, $\mathbb{P}(X_{m'} = x | X_0 = x_0) \geq 1/3^{m'}$ since it takes at most m' moves in a single direction to reach x, and if the chain arrives earlier, we include the probability of it staying at x until time m'. Thus, the transition probability after m' steps can be written as a mixture:

$$\mathbb{P}(X_{m'} = x | X_0 = x_0) = \delta \nu(x) + (1 - \delta) q(x | x_0), \tag{1.9}$$

for some conditional probability mass function q and with $\delta = m/3^{m'}$. The distribution at the start of a given iteration can always be thought of as a mixture of ν and some other distribution, where the mixture probability for ν could be 0. We can imagine that there is a hidden coin, and if it is showing "heads" then the distribution of the chain is ν. Since ν is the stationary distribution, if the coin is currently showing "heads" it will still be showing heads after a further m' moves. If the coin is showing "tails" then (1.9) tells us that there is a probability of at least δ that it will be showing heads after the next m' moves. Equivalently, the mixture probability of the component that is not ν has been multiplied by $1 - \delta$ or less. After km' iterations, it is, therefore at most $(1 - \delta)^k \to 0$ as $k \to \infty$.

However, convergence to a stationary distribution does not occur for all Markov chains. The chain in Example 1.2 is deterministic: if $X_0 = 0$, then $X_{km} = 0$ for all integers, k. The following examples illustrate two further cases.

Example 1.5 (See Figure 1.4.) Alter Example 1.3 so that the chain cannot remain at its current vertex but must move either clockwise or anticlockwise by a single vertex, each with a probability of $1/2$. As with the Examples 1.2 and 1.3, the stationary distribution has $\mathbb{P}(X_k = x) = 1/m$ for $x \in \mathcal{X}$.

If n is an even number, and X_0 is even then the chain in Example 1.5 only visits even-numbered states at even-numbered times, and odd-numbered

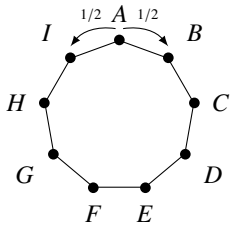

Figure 1.4 9-sided polygon with Markov transitions described in Example 1.5.

states at odd-numbered times. Such chains are termed *periodic* and clearly do not converge to their stationary distribution.

Example 1.6 Consider a Markov chain of the form in Example 1.3, but on two separate m-sided polygons with no movement between the two. A chain with separate regions between which there can be no movement is termed *reducible* because it can be reduced to simpler component parts.

A reducible chain does not even have a single stationary distribution. In Example 1.6 for any $\beta \in [0, 1]$ the distribution with probabilities β/m for each vertex on the first polygon and $(1-\beta)/m$ for each vertex on the second polygon is stationary.

A chain which is not reducible is termed *irreducible*, and a chain which is not periodic is termed *aperiodic*.

Ergodic Averages

The *ergodic theorem* for a Markov chain on a general state-space, $\mathcal{X}$, states that provided the chain satisfies natural generalisations of irreducibility and aperiodicity, and has a proper stationary distribution, ν, then as $k \to \infty$, the distribution of X_k converges to that stationary distribution. Furthermore, subject to the same conditions, for any $h \in \mathcal{L}^1(\nu)$, samples from the Markov chain satisfy a strong law of large numbers:

$$\widehat{I}_n(h) := \frac{1}{n} \sum_{k=1}^{n} h(X_k) \to \mathbb{E}_\nu \left[h(X) \right] \tag{1.10}$$

almost surely as $n \to \infty$.

Integrated Auto-Correlation Time and Effective Sample Size

Let us assume that X_0 is, in fact, drawn from the stationary distribution. Define $\sigma_h^2 := \mathrm{Var}_\nu \left[h(X) \right]$ and assume $\sigma_h^2 < \infty$. For $k \in \{0, 1, 2, \dots\}$, the

lag-k auto-correlation is $\rho_k := \text{Cor}\left[h(X_0), h(X_k)\right] = \text{Cor}\left[h(X_j), h(X_{j+k})\right]$ since the Markov chain is time-homogeneous. If the X_j were independent samples from ν then $n\text{Var}\left[\widehat{I}_n(h)\right] = \sigma_h^2$. For a stationary Markov chain with

$$\sum_{k=1}^{\infty} |\rho_k| < \infty, \tag{1.11}$$

it holds that

$$\lim_{n\to\infty} n\text{Var}\left[\widehat{I}_n(h)\right] = \sigma_h^2 \, \mathsf{IACT}_h, \tag{1.12}$$

where

$$\mathsf{IACT}_h := 1 + 2\sum_{k=1}^{\infty} \rho_k, \tag{1.13}$$

is the *integrated auto-correlation time*. To see why this is the case, firstly, without loss of generality, take h to have $\mathbb{E}_\nu\left[h(X)\right] = 0$; if this is not true initially, we subtract off the expectation: the variance properties are unchanged. Then

$$n\text{Var}\left[\widehat{I}_n(h)\right] = \frac{1}{n}\mathbb{E}\left[\sum_{k=1}^{n}\sum_{j=1}^{n} h(X_j)h(X_k)\right] = \frac{\sigma_h^2}{n}\sum_{k=1}^{n}\sum_{j=1}^{n}\rho_{|k-j|}. \tag{1.14}$$

But $\sum_{k=1}^{n}\sum_{j=1}^{n}\rho_{|k-j|} = n\rho_0 + 2\sum_{k=1}^{n}(n-k)\rho_k$, so

$$\frac{n}{\sigma_h^2}\text{Var}\left[\widehat{I}_n(h)\right] = 1 + 2\sum_{k=1}^{n}\left(1 - \frac{k}{n}\right)\rho_k = 1 + 2\sum_{k=0}^{\infty}\max\left(0, 1 - \frac{k}{n}\right)\rho_k.$$

Given (1.11), the dominated converge theorem permits us to exchange the ordering of the limit as $n \to \infty$ and the sum over k, which provides the limit (1.12).

The practical consequence of (1.12) is that, for finite n,

$$\text{Var}\left[\widehat{I}_h\right] \approx \frac{\sigma_h^2}{n/\mathsf{IACT}_h}, \tag{1.15}$$

the same as the variance if n/IACT_h i.i.d. samples from ν had been used. The quantity n/IACT_h is, therefore, known as the *effective sample size*. Since they relate directly to the inverse variance of $\widehat{I}_n(h)$, effective sample size and the inverse of the integrated auto-correlation time are useful measures of the efficiency of a Markov chain for estimating $\mathbb{E}_\nu\left[h(X)\right]$.

1.4 Stochastic Differential Equations

The Langevin stochastic differential equation is the basis for the Metropolis Adjusted Langevin Algorithm (Section 2.1.4) and for stochastic gradient Langevin methods (Chapter 3). It is also key to understanding the efficiency of various Metropolis–Hastings algorithms when the dimension, d, is high (see Chapter 2). We start with a heuristic introduction to stochastic differential equations before considering a special case of the Langevin diffusion known as the Ornstein–Uhlenbeck process and then moving onto the general Langevin diffusion.

Consider a differential equation of the form

$$\frac{\mathrm{d}x_t}{\mathrm{d}t} = a(x_t, t),$$

with a known initial value for x_0. Discretising time leads to the simple Euler approximation

$$\delta x_t \approx a(x_t, t)\delta t,$$

where $\delta x_t = x_{t+\delta t} - x_t$. Setting $\delta t = T/m$, starting with x_0, and recursively applying the Euler update m times leads to an approximation $\widehat{x}_T$, which approaches the true value x_T as $m \to \infty$.

Instead of deterministic updates, we might wish to allow for the addition of random noise with scale proportional to $b(x_t, t)$. The initial value, X_0, may now be random and setting $\delta X_t := X_{t+\delta t} - X_t$ leads to one possible update

$$\delta X_t \approx a(X_t, t)\delta t + b(X_t, t)\epsilon_t, \qquad \epsilon_t \sim \mathsf{N}(0, \delta t),$$

where the Gaussian noise terms ϵ_t are independent of all previous randomness, and X_t has become a random variable. A noise distribution of the form $\mathsf{N}(0, \delta t)$ is chosen because it is self-consistent. For example, with $a(X_t, t) = a$ and $b(x_t, t) = b$, after two time steps initialised at $X_0 = x_0$, we have

$$X_{2\delta t} \approx x_0 + a\delta t + b\epsilon_{\delta t} + a\delta t + b\epsilon_{2\delta t} = x_0 + 2a\delta t + b\tilde{\epsilon}_{2\delta t}$$

where $\tilde{\epsilon}_{2\delta t} \sim \mathsf{N}(0, 2\delta t)$, since the two noise terms $\epsilon_{\delta t}, \epsilon_{2\delta t}$ are independent. However, the right-hand side of this expression is exactly of the same form we would get from a single time step of size $2\delta t$ to obtain $X_{2\delta t}$ from X_0.

The process with $a = 0$, $b = 1$ and $X_0 = 0$ consists of a sequence of mean-zero Gaussian increments, each with a variance of δt. This is a discretisation of a process known as *Brownian motion*, which is often denoted by W_t. In

particular, we have that

$$\delta W_t = W_{t+\delta t} - W_t \sim \mathsf{N}(0, \delta t),$$

and $W_t \sim \mathsf{N}(0, t)$. From the definition of W_t, we may rewrite the noisy update as

$$\delta X_t \approx a(X_t, t)\delta t + b(X_t, t)\delta W_t. \tag{1.16}$$

Consider this process on some interval $[0, T]$, with $\delta t = T/m$ and $X_0 = x_0$, for some initial value x_0. Subject to some regularity conditions, the limit as $m \to \infty$ exists and is written:

$$\mathrm{d}X_t = a(X_t, t)\mathrm{d}t + b(X_t, t)\mathrm{d}W_t.$$

This is known as a *stochastic differential equation* (SDE), and (1.16) is the Euler–Maruyama approximation to it. Subject to the initial condition, the *solution* to this SDE is the stochastic process $\{X_t\}_{t\in[0,T]}$ obtained from the limit $\delta t \to 0$ of the discrete-time process defined through (1.16).

The above heuristic describes a one-dimensional SDE and its Euler–Maruyama discretisation; however, it is straightforward to extend these to higher dimensions with $\mathbf{X}_t \in \mathbb{R}^d$, $\mathbf{a} : \mathbb{R}^d \times [0, \infty) \to \mathbb{R}^d$, $\mathbf{W}_t \in \mathbb{R}^k$ and the $d \times k$ matrix $\mathbf{b} : \mathbb{R}^d \times [0, \infty) \to \mathbb{R}^{dk}$.

A stochastic process that satisfies an SDE is called a *diffusion*. For the most part, we will deal with time-homogeneous diffusions, where a and b have no explicit time dependence; however, time-inhomogeneous diffusions will be used in Chapter 3.

1.4.1 The Ornstein–Uhlenbeck Process

Consider the SDE

$$\mathrm{d}X_t = -\frac{1}{2\sigma^2}b^2 X_t \mathrm{d}t + b\mathrm{d}W_t.$$

The Euler–Maruyama discretisation gives

$$X_{t+\delta t} \approx X_t + \delta X_t = \left(1 - \frac{b^2}{2\sigma^2}\delta t\right) X_t + b\delta W_t.$$

Since δW_t is Gaussian distributed and independent of X_t, if X_t is Gaussian so is $X_{t+\delta t}$. Moreover, if $\mathbb{E}[X_t] = 0$, then $\mathbb{E}[X_{t+\delta t}] = 0$. Finally, if $\mathrm{Var}[X_t] = \sigma^2$ then

$$\mathrm{Var}[X_{t+\delta t}] = \left(1 - \frac{b^2}{2\sigma^2}\delta t\right)^2 \sigma^2 + b^2\delta t = \sigma^2 + \frac{1}{4\sigma^4}b^4\delta t^2.$$

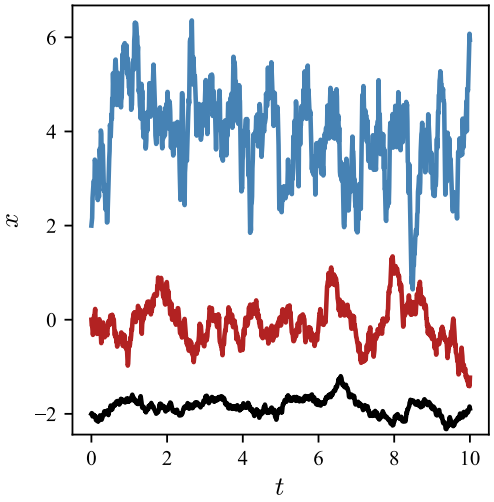

Figure 1.5 Three realisations of the Ornstein–Uhlenbeck processes, all with $\sigma = 1$, and on the time interval $[0, 10]$. Other parameter settings are $x_0 = 2$, $m = 4$ and $b = 3$; $x_0 = m = 0$ and $b = 1$; $x_0 = -2$, $m = -4$ and $b = 1/3$.

In the limit $m \to \infty$, as the number of increments is increased, with $\delta t = T/m \downarrow 0$, the term in δt^2 becomes irrelevant: the variance does not change. Thus, if $X_0 \sim \mathsf{N}(0, \sigma^2)$ then $X_t \sim \mathsf{N}(0, \sigma^2)$ for all $t > 0$; the SDE is stationary. Shifting the coordinate system by m we see that the slightly more general SDE

$$\mathrm{d}X_t = -\frac{1}{2\sigma^2} b^2 (X_t - m)\mathrm{d}t + b\mathrm{d}W_t \tag{1.17}$$

has a stationary distribution of $\mathsf{N}(m, \sigma^2)$. The process arising from the SDE (1.17) is known as the *Ornstein–Uhlenbeck* (OU) process. Substituting $s = b^2 t$, the SDE becomes $\mathrm{d}X_s = -(X_s - m)/(2\sigma^2)\mathrm{d}s + \mathrm{d}W_s$, which explains why b^2 is termed the *speed* of the diffusion. Figure 1.5 presents three realisations of OU processes with stationary distribution $\mathsf{N}(m, 1)$ and started from the corresponding $m/2$. Each diffusion has a different speed, and the effect of this on the convergence to, and mixing within, the stationary distribution is clearly visible.

1.4.2 The Infinitesimal Generator

The *infinitesimal generator* (or, simply, *generator*) of a continuous-time stochastic process acts on a function h of the process:

$$(\mathcal{L}h)(\mathbf{x}) := \frac{\partial}{\partial t}\mathbb{E}\left[h(\mathbf{X}_t)|\mathbf{X}_0 = \mathbf{x}\right]\Big|_{t=0} = \lim_{\delta t \downarrow 0} \frac{\mathbb{E}\left[h(\mathbf{X}_{\delta t})\right] - h(\mathbf{x})}{\delta t}. \quad (1.18)$$

The set of functions for which the limit exists for all $\mathbf{x}$ is called the *domain* of the infinitesimal generator. Subject to regularity conditions, this includes the set of compactly supported functions with a second derivative that is continuous, denoted C_0^2.

For processes defined by an SDE, we can gain some insight into their generator by considering a Taylor expansion. For simplicity of presentation, we consider $x \in \mathbb{R}$:

$$\frac{1}{\delta t}\mathbb{E}\left[h(X_{\delta t}) - h(x)\right] = \frac{1}{\delta t}\mathbb{E}\left[(X_{\delta t} - x)h'(x) + \frac{1}{2}(X_{\delta t} - x)^2 h''(x) + \ldots\right].$$

The Euler–Maruyama approximation of the SDE is $X_{\delta t} - x \approx a(x)\delta t + b(x)\delta W_t$. Thus $\mathbb{E}\left[X_{\delta t} - x\right] \approx a(x)\delta t$ and $\mathbb{E}\left[(X_{\delta t} - x)^2\right] \approx b(x)^2\delta t + a(x)^2[\delta t]^2$, with all higher order expectations at most $o(\delta t)$. Thus, we might expect that

$$(\mathcal{L}h)(x) = a(x)\frac{\mathrm{d}h}{\mathrm{d}x} + \frac{1}{2}b(x)^2\frac{\mathrm{d}^2 h}{\mathrm{d}x^2},$$

and this is indeed the case. For a multivariate diffusion, the generator is

$$(\mathcal{L}h)(\mathbf{x}) = \sum_{i=1}^{d} a_i \frac{\partial h}{\partial x_i}\bigg|_{\mathbf{x}} + \frac{1}{2}\sum_{i=1}^{d}\sum_{j=1}^{d}(bb^\top)_{i,j}\frac{\partial^2 h}{\partial x_i \partial x_j}\bigg|_{\mathbf{x}}. \quad (1.19)$$

Generators of diffusion processes are used in the next subsection to derive the stationary density of two classes of diffusion that appear repeatedly in Chapters 2 and 3. Generators of diffusions are also used in Chapter 6 for the assessment and improvement of algorithms. Finally, Chapter 5 employs the generators of another class of continuous-time stochastic processes to determine the processes' stationary distributions.

1.4.3 Langevin Diffusions

We now describe two classes of diffusion, the overdamped and underdamped Langevin diffusions, where the stationary density forms an explicit part of the SDE formulation.

The Overdamped Langevin Diffusion

Consider a positive, differentiable density function $f(\mathbf{x})$ for $\mathbf{x} \in \mathbb{R}^d$, and the following SDE:

$$\mathrm{d}\mathbf{X}_t = \frac{1}{2}\nabla \log f(\mathbf{X}_t) b^2 \mathrm{d}t + b\mathrm{d}\mathbf{W}_t. \tag{1.20}$$

A solution to this SDE is known as an *overdamped Langevin diffusion*. The OU process (1.17) with $f(x) = \mathsf{N}(x; m, 1/a)$ is a special case of this class of diffusions, and in this case, as seen in Section 1.4.1, f is the density of the stationary process. In fact, this is true in general: the stationary density of the overdamped Langevin diffusion (1.20) is f. To see this in one dimension, consider the infinitesimal generator of the diffusion:

$$(\mathcal{L}h)(x) = \frac{1}{2}b^2 \frac{f'(x)}{f(x)} h'(x) + \frac{1}{2}b^2 h''(x).$$

This is the rate of change of the expectation of $h(X_t)$ at $t = 0$, when started from $X_0 = x$. Suppose instead that X_0 has a density of f. Then, the rate of change of the expectation of X_t at $t = 0$ can be calculated by taking expectations with respect to X_0. This is

$$\frac{1}{2}b^2 \int \left\{ \frac{f'(x)}{f(x)} h'(x) + h''(x) \right\} f(x)\, \mathrm{d}x = \frac{1}{2}b^2 \int \{f(x)h'(x)\}'\, \mathrm{d}x = 0$$

for all sufficiently smooth h with compact support. Thus, if $X_0 \sim f$, $\frac{\mathrm{d}}{\mathrm{d}t}\mathbb{E}\left[h(X_t)\right]\big|_{t=0} = 0$. This is true for *all* $h \in C_0^2$, and so the distribution of X_t does not change as t increases from 0. The distribution at time 0 must, therefore, be the stationary distribution of the Langevin diffusion (1.20), and f is the corresponding stationary density.

When Langevin diffusions are employed in a Bayesian setting, $f(\mathbf{x})$ is often a posterior density whose normalising constant is, typically, intractable. The fact that the calculation of $\nabla \log f(\mathbf{X}_t)$ does not require this normalising constant is crucial to the practical use of these diffusions.

The Underdamped Langevin Diffusion

The *underdamped Langevin diffusion* extends the state space to include a velocity component, $\mathbf{P}_t$:

$$\mathrm{d}\mathbf{X}_t = \mathbf{P}_t \mathrm{d}t, \tag{1.21}$$

$$\mathrm{d}\mathbf{P}_t = -\gamma \mathbf{P}_t \mathrm{d}t + c\nabla \log f(\mathbf{X}_t)\mathrm{d}t + \sqrt{2\gamma c}\, \mathrm{d}\mathbf{W}_t \tag{1.22}$$

Intuitively, dividing (1.22) through by γ and taking the limit as $\gamma \to \infty$ and $c \to \infty$ with $c/\gamma = b^2/2$ fixed, we obtain the overdamped Langevin diffusion, so the latter is a limiting case of the underdamped diffusion.

The underdamped Langevin diffusion targets $f(\mathbf{x})g(\mathbf{p})$, where

$$g(\mathbf{p}) = \frac{1}{\sqrt{2\pi c}} \exp\left(-\frac{1}{2c}\|\mathbf{p}\|^2\right).$$

To see this we, again, restrict ourselves to the one-dimensional case to simplify the presentation and, again, we start from the generator:

$$(\mathcal{L}h)(x,p) = ph_x(x,p) - \gamma p h_p(x,p) + c\frac{f'(x)}{f(x)}h_p(x,p) + \gamma c h_{p,p}(x,p),$$

where we have used subscripts to denote differentiation of h with respect to x or p. The quantity $(\mathcal{L}h)(x,p)$ is the rate of change of the expectation of $h(X_t, P_t)$ at $t = 0$, when started at $X_0 = x$ and $P_0 = p$. Thus if X_0 and P_0 have respective densities of $f(x)$ and $g(p)$, then the rate of change of $\mathbb{E}\left[h(X_t, P_t)\right]$ at $t = 0$ is

$$\iint \left\{ph_x - \gamma p h_p + c\frac{f'(x)}{f(x)}h_p + \gamma c h_{p,p}\right\} f(x)g(p)\mathrm{d}p\mathrm{d}x.$$

In the manipulations that follow, we will twice use the fact that $g'(p) = -pg(p)/c$. Firstly, integration by parts gives

$$\int \gamma c h_{p,p} g(p)\ \mathrm{d}p = \int p\gamma h_p g(p)\ \mathrm{d}p,$$

so the second and fourth terms cancel. Secondly, two integrations by parts, first with respect to p and then with respect to x, give

$$\begin{aligned}\iint c f'(x) h_p g(p)\ \mathrm{d}p\mathrm{d}x &= \iint f'(x) h p g(p)\ \mathrm{d}p\mathrm{d}x \\ &= -\iint f(x) h_x p g(p)\ \mathrm{d}p\mathrm{d}x,\end{aligned}$$

so the first and third terms cancel. The argument is completed analogously to that for the overdamped Langevin diffusion.

In Chapter 3, we explore further the overdamped and underdamped Langevin diffusions as practical algorithms for scalable Monte Carlo inference in the large-data setting and show that the discretisation of these diffusion processes leads to important special cases of the general framework for stochastic gradient MCMC algorithms.

1.5 The Kernel Trick

Chapter 6 introduces the *kernel Stein discrepancy* and uses it to measure the discrepancy between a sample of points and a distribution of interest. Practical use of the methodology is made feasible by the ability to reduce what appears to be an infinite amount of computation – maximising a quantity over an uncountably infinite set of possible functions – to only a finite number of arithmetic operations. The key mechanism for this simplification is often called *the kernel trick*, and the setting for its use is a *reproducing kernel Hilbert space*.

This section first explains the kernel trick in the more familiar setting of a finite-dimensional inner-product space, before extending to the more general setting required for Chapter 6. Whilst many of the concepts introduced are much more general, our presentation focuses on the specific setting of relevance: the vectors of our inner-product space are functions, the associated field is $\mathbb{R}$ and the inner product is an integral with respect to a probability distribution.

Throughout, $f(\cdot)$, $g(\cdot)$ *etc*. are functions from $\mathcal{X} \to \mathbb{R}$, where $\mathcal{X}$ is $\mathbb{R}^d$ or some closed or open subset of $\mathbb{R}^d$; $f(\mathbf{x})$, $g(\mathbf{x})$ *etc* denote the function evaluated at $\mathbf{x} \in \mathcal{X}$. The probability distribution ν is assumed to have a density $\nu(\mathbf{x})$ on $\mathcal{X}$.

1.5.1 Finite-Dimensional Inner Product Spaces

Let $0(\cdot)$ be the function such that $0(\mathbf{x}) = 0$ for all $\mathbf{x} \in \mathcal{X}$. A set, $\mathbb{V}$, of functions from $\mathcal{X} \to \mathbb{R}$ is a *vector space* over $\mathbb{R}$ if the following axioms are satisfied:

1. $0(\cdot) \in \mathbb{V}$.
2. $f(\cdot) \in \mathbb{V} \implies -f(\cdot) \in \mathbb{V}$.
3. $f(\cdot), g(\cdot) \in \mathbb{V} \implies f(\cdot) + g(\cdot) \in \mathbb{V}$.
4. $f(\cdot) \in \mathbb{V}$ and $a \in \mathbb{R} \implies af(\cdot) \in \mathbb{V}$.

Aside: The associativity, commutativity and distributativity axioms of a general vector space are satisfied automatically when the elements are functions and from $\mathcal{X}$ to $\mathbb{R}$ and the field is $\mathbb{R}$.

Every finite-dimensional vector space has a dimension, n, such that there is a set of n vectors $\{b_1(\cdot), \ldots, b_n(\cdot)\}$ which satisfy two properties:

1. **Linear independence**: If there are $a_1, \ldots, a_n \in \mathbb{R}$ such that $\sum_{i=1}^n a_i b_i(\cdot) = 0$ then $a_i = 0$ for all $i \in \{1, \ldots, n\}$.

2. **Spanning** $\mathbb{V}$: for each $f(\cdot) \in \mathbb{V}$ there are $a_1, \ldots, a_n \in \mathbb{R}$ such that $f(\cdot) = \sum_{i=1}^{n} a_i b_i(\cdot)$.

The set $\{b_1(\cdot), \ldots, b_n(\cdot)\}$ is called a *basis*.

Example 1.7 It is straightforward to check that the set

$$\begin{aligned}\mathbb{V} &= \{f(\cdot) : f(x) = c\sin(x+\theta) : c \in \mathbb{R}, \theta \in [0, 2\pi)\} \\ &= \{f(\cdot) : f(x) = a\sin x + b\cos x; a, b \in \mathbb{R}\}\end{aligned}$$

satisfies Axioms 1–4, whatever the domain, $\mathcal{X} \subseteq \mathbb{R}$. We may take $b_1(\cdot) = \sin(\cdot)$ and $b_2(\cdot) = \cos(\cdot)$. However, we may also take $b_1(\cdot) = \sin(\cdot)+3\cos(\cdot)$ and $b_2(\cdot) = \cos(\cdot)$, for example.

For any vector space $\mathbb{V}$ of functions from $\mathcal{X} \to \mathbb{R}$ and any distribution ν with a probability density function on $\mathcal{X}$ of $\nu(x)$, we define the *inner product*

$$\langle f(\cdot), g(\cdot)\rangle_\nu = \int f(\mathbf{x})g(\mathbf{x})\nu(\mathbf{x})\,\mathrm{d}\mathbf{x}, \tag{1.23}$$

where here and throughout this section, if the integral range is not specified then it is $\mathcal{X}$. We refer to this inner product as $\langle\cdot,\cdot\rangle_\nu$.

The inner product defined by (1.23) clearly satisfies two of the three defining properties of an inner product: $\langle f(\cdot), g(\cdot)\rangle = \langle g(\cdot), f(\cdot)\rangle$ and $\langle f(\cdot) + g(\cdot), h(\cdot)\rangle = \langle f(\cdot), h(\cdot)\rangle + \langle g(\cdot), h(\cdot)\rangle$. However, we have only that $\langle f(\cdot), f(\cdot)\rangle = 0 \Leftrightarrow f(\mathbf{x}) = 0(\mathbf{x})$ ν-almost everywhere, rather than $\langle f(\cdot), f(\cdot)\rangle = 0 \Leftrightarrow f(\cdot) = 0(\cdot)$. Each f belongs to an equivalence class of functions that are equal ν-almost everywhere. This set of equivalence classes forms a vector space and (1.23) defines an inner product on this space, not on the space of functions, $\mathbb{V}$. To keep the presentation in this section as straightforward as possible our wording ignores this distinction, but the more rigorous reader may wish to replace any vector space of functions and inner product between these functions with the corresponding vector space of equivalence classes of functions and inner products between these equivalence classes.

The inner product provides a norm, called the *induced norm*, the square of which is

$$\|f(\cdot)\|_\nu^2 = \langle f(\cdot), f(\cdot)\rangle_\nu = \int f(\mathbf{x})^2\nu(\mathbf{x})\,\mathrm{d}\mathbf{x}.$$

Example 1.8 (Example 1.7 continued) Let $\mathcal{X} = [0, 2\pi]$ and let ν be the

uniform distribution on $[0, 2\pi]$. For any $f(\cdot), g(\cdot) \in \mathbb{V}$,

$$\langle f(\cdot), g(\cdot)\rangle_\nu = \frac{1}{2\pi}\int_0^{2\pi} f(x)g(x)\,\mathrm{d}x \quad \text{and} \quad \|f(\cdot)\|_\nu^2 = \frac{1}{2\pi}\int_0^{2\pi} f(x)^2\,\mathrm{d}x.$$

Example 1.9 For a general vector space $\mathbb{V}$ of functions of the form $\mathcal{X} \to \mathbb{R}$, let $\mathbb{V}_\nu$ be the elements of $\mathbb{V}$ which have a finite norm induced by ν:

$$\mathbb{V}_\nu = \left\{ f(\cdot) \in \mathbb{V} : \int f(\mathbf{x})^2 \nu(\mathbf{x})\,\mathrm{d}\mathbf{x} < \infty \right\}.$$

Then $\mathbb{V}_\nu$ is also a vector space, since Axioms 1, 2 and 4 are satisfied trivially, and Axiom 3 is satisfied since for any $f(\cdot), g(\cdot) \in \mathbb{V}_\nu$,

$$\begin{aligned}\|f(\cdot) + g(\cdot)\|_\nu^2 &= \langle f(\cdot) + g(\cdot), f(\cdot) + g(\cdot)\rangle_\nu \\ &= \|f(\cdot)\|_\nu^2 + 2\langle f(\cdot), g(\cdot)\rangle_\nu + \|g(\cdot)\|_\nu^2 \\ &\le \|f(\cdot)\|_\nu^2 + 2\|f(\cdot)\|_\nu\|g(\cdot)\|_\nu + \|g(\cdot)\|_\nu^2 < \infty,\end{aligned}$$

where the third line uses the Cauchy–Schwarz inequality, which, in this case, is the familiar inequality $\mathbb{E}\left[f(\mathbf{X})g(\mathbf{X})\right]^2 \le \mathbb{E}[f(\mathbf{X})^2]\mathbb{E}\left[g(\mathbf{X})^2\right]$, where $\mathbf{X}$ has a density ν on $\mathcal{X}$.

Henceforth, for narrative simplicity, we will assume that $\mathbb{V}$ is a finite-dimensional vector space with dimension n. Section 1.5.4 extends the narrative to potentially infinite-dimensional spaces.

When considering the inner product $\langle\cdot,\cdot\rangle_\nu$, two vectors $f(\cdot), g(\cdot) \in \mathbb{V}$ are said to be *orthogonal* if $\langle f(\cdot), g(\cdot)\rangle_\nu = 0$ and the basis vectors, $e_1(\cdot), \ldots, e_n(\cdot)$ are said to be *orthonormal* if they are orthogonal and each has a norm of 1: for each $j, k \in \{1, \ldots, n\}$,

$$\|e_j(\cdot)\|_\nu = 1 \quad \text{and} \quad \langle e_j(\cdot), e_k(\cdot)\rangle_\nu = 0$$

whenever $j \neq k$. We will reserve the symbols $\{e_k(\cdot)\}_{k=1}^n$ for any set of n orthonormal basis functions.

The representation of $f(\cdot)$ in terms of an orthonormal basis

$$f(\cdot) = \sum_{j=1}^n f_j e_j(\cdot)$$

is termed an *orthonormal decomposition* of $f(\cdot)$. Since the $e_i(\cdot)$ are orthonormal, the projection of $f(\cdot)$ onto $e_k(\cdot)$ is f_k:

$$\begin{aligned}\langle f(\cdot), e_k(\cdot)\rangle_\nu &= \left\langle \sum_{j=1}^n f_j e_j(\cdot), e_k(\cdot) \right\rangle_\nu = \sum_{j=1}^n f_j \langle e_j(\cdot), e_k(\cdot)\rangle_\nu \\ &= f_k. \end{aligned} \tag{1.24}$$

Furthermore, the squared norm of $f(\cdot)$ is the sum of the squares of the orthonormal projections:

$$\|f(\cdot)\|^2 = \left\langle \sum_{j=1}^{n} f_j e_j(\cdot), \sum_{k=1}^{n} f_k e_k(\cdot) \right\rangle = \sum_{j=1}^{n} \sum_{k=1}^{n} f_j f_k \langle e_j(\cdot), e_k(\cdot) \rangle$$
$$= \sum_{j=1}^{n} f_j^2. \tag{1.25}$$

Example 1.10 In Example 1.7, since $\int_0^{2\pi} \sin^2 x \, \mathrm{d}x = \int_0^{2\pi} \cos^2 x \, \mathrm{d}x = \pi$ and $\int_0^{2\pi} \sin x \cos x \, \mathrm{d}x = 0$,

$$e_1(\cdot) = \sqrt{2}\sin(\cdot) \quad \text{and} \quad e_2(\cdot) = \sqrt{2}\cos(\cdot)$$

form an orthonormal basis for $\mathbb{V}$ when ν is the uniform distribution on $[0, 2\pi]$. Any function $f(\cdot) \in \mathbb{V}$ can be written as $f(\cdot) = f_1 e_1(\cdot) + f_2 e_2(\cdot)$. For example set

$$f(x) = \sin(x + \pi/6) = \frac{\sqrt{3}}{2}\sin x + \frac{1}{2}\cos x. \tag{1.26}$$

So $f_1 = \sqrt{3}/(2\sqrt{2})$ and $f_2 = 1/(2\sqrt{2})$. Also

$$\|f(\cdot)\|_\nu^2 = f_1^2 + f_2^2 = \frac{3}{8} + \frac{1}{8} = \frac{1}{2} = \frac{1}{2\pi}\int_0^{2\pi} \sin^2(x + \pi/6) \, \mathrm{d}x.$$

1.5.2 Kernels in a Finite-Dimensional Inner Product Space

As in the previous subsection, let $\mathbb{V}$ be an n-dimensional vector space of functions from $\mathcal{X}$ to $\mathbb{R}$ and let ν be a probability distribution on $\mathcal{X}$ with a probability density of $\nu(\mathbf{x})$, $\mathbf{x} \in \mathcal{X}$. Finally, let $\{e_k(\cdot)\}_{k=1}^n$ be a set of basis functions which is orthonormal with respect to the inner product (1.23).

Let $\lambda_1, \ldots, \lambda_n$ be a set of non-negative scalars and consider the following real-valued function on $\mathcal{X} \times \mathcal{X}$:

$$\mathsf{k}(\mathbf{x}, \mathbf{y}) = \sum_{j=1}^{n} \lambda_j e_j(\mathbf{x}) e_j(\mathbf{y}). \tag{1.27}$$

Clearly, $\mathsf{k}(\cdot, \cdot)$ is *symmetric*: $\mathsf{k}(\mathbf{y}, \mathbf{x}) = \mathsf{k}(\mathbf{x}, \mathbf{y})$. Moreover, $\mathsf{k}(\cdot, \cdot)$ is *positive*

semidefinite: for any finite $J < \infty$, $c_1, \ldots, c_J \in \mathbb{R}$ and $\mathbf{x}_1, \ldots \mathbf{x}_J \in \mathbb{R}^d$,

$$\begin{aligned}\sum_{j=1}^{J}\sum_{k=1}^{J} c_j c_k \mathsf{k}(\mathbf{x}_j, \mathbf{x}_k) &= \sum_{j=1}^{J}\sum_{k=1}^{J} c_j c_k \sum_{l=1}^{n} \lambda_l e_l(\mathbf{x}_j) e_l(\mathbf{x}_k) \\ &= \sum_{l=1}^{n} \lambda_l \sum_{j=1}^{J}\sum_{k=1}^{J} c_j c_k e_l(\mathbf{x}_j) e_l(\mathbf{x}_k) \\ &= \sum_{l=1}^{n} \lambda_l \left\{\sum_{j=1}^{J} c_j e_l(\mathbf{x}_j)\right\}^2 \geq 0.\end{aligned}$$

Any function $\mathsf{k}(\cdot,\cdot) : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ which is both symmetric and positive semidefinite is called a *kernel*.

Example 1.11 Continuing Example 1.7, let $\mathsf{k} : [0, 2\pi] \times [0, 2\pi] \to \mathbb{R}$ be

$$\begin{aligned}\mathsf{k}(x, y) &= \frac{1}{2} e_1(x) e_1(y) + \frac{3}{2} e_2(x) e_2(y) = \sin x \sin y + 3 \cos x \cos y \\ &= 2\cos(y - x) + \cos(y + x).\end{aligned}$$

This is symmetric and positive definite by construction.

Given the definition of $\mathsf{k}(\cdot,\cdot)$ in (1.27), define

$$\mathsf{k}(\mathbf{x}, \cdot) = \sum_{j=1}^{n} \lambda_j e_j(\mathbf{x}) e_j(\cdot), \tag{1.28}$$

and $\mathsf{k}(\cdot, \mathbf{x}) = \mathsf{k}(\mathbf{x}, \cdot)$. Since $e_j(\mathbf{x}) \in \mathbb{R}$, $\mathsf{k}(\mathbf{x}, \cdot) \in \mathbb{V}$. Furthermore, for $f(\cdot) \in \mathbb{V}$ define the operator T_k via

$$T_\mathsf{k} f(\cdot) = \int \mathsf{k}(\cdot, \mathbf{y}) f(\mathbf{y}) \nu(\mathbf{y})\, \mathrm{d}\mathbf{y}. \tag{1.29}$$

Then T_k is a *linear operator*, since for any $a, b \in \mathbb{R}$ and $f(\cdot), g(\cdot) \in \mathbb{V}$,

$$T_\mathsf{k}\{a f(\cdot) + b g(\cdot)\} = a T_\mathsf{k} f(\cdot) + b T_\mathsf{k} g(\cdot).$$

Now, writing $f(\cdot) = \sum_{k=1}^{n} f_k e_k(\cdot)$,

$$\begin{aligned}(T_\mathsf{k} f(\cdot))(\mathbf{x}) &= \int \mathsf{k}(\mathbf{x}, \mathbf{y}) f(\mathbf{y}) \nu(\mathbf{y}) \mathrm{d}\mathbf{y} \\ &= \langle \mathsf{k}(\mathbf{x}, \cdot), f(\cdot)\rangle_\nu = \left\langle \sum_{j=1}^{n} \lambda_j e_j(\mathbf{x}) e_j(\cdot), \sum_{k=1}^{n} f_k e_k(\cdot) \right\rangle_\nu \\ &= \sum_{j=1}^{n}\sum_{k=1}^{n} \lambda_j e_j(\mathbf{x}) f_k \langle e_j(\cdot), e_k(\cdot)\rangle_\nu = \sum_{k=1}^{n} \lambda_k f_k e_k(\mathbf{x}).\end{aligned}$$

So $T_{\mathsf{k}} f(\cdot) = \sum_{k=1}^{n} \lambda_k f_k e_k(\cdot)$ and, hence, $T_{\mathsf{k}} f(\cdot) \in \mathbb{V}$, too. Moreover, considering $f(\cdot) = e_j(\cdot)$, we see that $T_{\mathsf{k}} e_j(\cdot) = \lambda_j e_j(\cdot)$; each $e_j(\cdot)$ is an eigenfunction of T_{k} with a corresponding eigenvalue of λ_j.

Example 1.12 Continuing Example 1.7, with the kernel from Example 1.11,

$$\mathsf{k}(x, \cdot) = \sin x \sin(\cdot) + 3 \cos x \cos(\cdot) = 2 \cos(\cdot - x) + \cos(\cdot + x).$$

Let $f(\cdot)$ be as defined in (1.26). Then, using the definite integrals at the start of Example 1.10,

$$\begin{aligned} T_{\mathsf{k}} f(\cdot) &= \frac{1}{2\pi} \int_0^{2\pi} \{\sin(\cdot) \sin y + 3 \cos(\cdot) \cos y\} \left\{ \frac{\sqrt{3}}{2} \sin y + \frac{1}{2} \cos y \right\} \, \mathrm{d}y \\ &= \frac{1}{4\pi} \int_0^{2\pi} \sqrt{3} \sin(\cdot) \sin^2 y + 3 \cos(\cdot) \cos^2 y \, \mathrm{d}y \\ &= \frac{1}{4} \left\{ \sqrt{3} \sin(\cdot) + 3 \cos(\cdot) \right\}. \end{aligned}$$

Since

$$e_1(\cdot) = \sqrt{2} \sin(\cdot), \; e_2(\cdot) = \sqrt{2} \cos(\cdot), \; f_1 = \frac{\sqrt{3}}{2\sqrt{2}}, \; f_2 = \frac{1}{2\sqrt{2}},$$

$\lambda_1 = 1/2$ and $\lambda_2 = 3/2$, $T_{\mathsf{k}} f(\cdot)$ is, therefore,

$$\lambda_1 f_1 e_1(\cdot) + \lambda_2 f_2 e_2(\cdot),$$

as we would hope.

1.5.3 A New Inner Product and the Kernel Trick in Finite Dimensions

Let $\{e_j(\cdot)\}_{j=1}^{n}$ be an orthonormal basis for $\mathbb{V}$ and let k be defined through (1.27) with respect to this basis. For $f(\cdot), g(\cdot) \in \mathbb{V}$ with

$$f(\cdot) = \sum_{j=1}^{n} f_j e_j(\cdot) \quad \text{and} \quad g(\cdot) = \sum_{j=1}^{n} g_j e_j(\cdot), \tag{1.30}$$

the inner product with respect to ν is the sum of the products of the orthogonal projections:

$$\langle f(\cdot), g(\cdot)\rangle_\nu = \left\langle \sum_{j=1}^{n} f_j e_j(\cdot), \sum_{k=1}^{n} g_k e_k(\cdot) \right\rangle_\nu = \sum_{j=1}^{n}\sum_{k=1}^{n} f_j g_k \langle e_j(\cdot), e_k(\cdot)\rangle_\nu$$
$$= \sum_{j=1}^{n} f_j g_j.$$

We now define a new inner product

$$\langle f(\cdot), g(\cdot)\rangle_{\mathsf{k}} = \sum_{j=1}^{n} \frac{f_j g_j}{\lambda_j}, \tag{1.31}$$

where the $\{\lambda_j\}_{j=1}^n$, are exactly those from the definition of k and are the eigenvalues of the operator T_{k}.

This inner product may be rephrased in terms of a set of eigenfunctions which are orthonormal with respect to $\langle\cdot,\cdot\rangle_{\mathsf{k}}$: $\{e'_j(\cdot)\}_{j=1}^n$ with $e'_j(\cdot) = \sqrt{\lambda_j} e_j(\cdot)$. With respect to this basis, the vector

$$f(\cdot) = \sum_{j=1}^{n} f'_j e'_j(\cdot)$$

with $f'_j = f_j/\sqrt{\lambda_j}$. Using an analogous decomposition for $g(\cdot)$,

$$\langle f(\cdot), g(\cdot)\rangle_{\mathsf{k}} = \sum_{j=1}^{n} f'_j g'_j,$$

as expected. Finally,

$$\mathsf{k}(\mathbf{x}, \mathbf{y}) = \sum_{j=1}^{n} e'_j(\mathbf{x}) e'_j(\mathbf{y}).$$

Example 1.13 In Example 1.11, $e'_1(\cdot) = \sin(\cdot)$ and $e'_2(\cdot) = \sqrt{3}\cos(\cdot)$. Clearly,

$$\mathsf{k}(x, y) = \sin x \sin y + 3\cos x \cos y = e'_1(x)e'_1(y) + e'_2(x)e'_2(y).$$

For $f(\cdot)$ as in Example 1.26,

$$f(\cdot) = \frac{\sqrt{3}}{2}\sin(\cdot) + \frac{1}{2}\cos(\cdot) = \frac{\sqrt{3}}{2}\sin(\cdot) + \frac{\sqrt{3}}{6}\sqrt{3}\cos(\cdot),$$

so $f_1' = \sqrt{3}/2$ and $f_2' = \sqrt{3}/6$. Thus

$$\|f(\cdot)\|_{\mathsf{k}}^2 = \frac{3}{4} + \frac{3}{36} = \frac{5}{6}.$$

The Kernel Trick

From the definition (1.28) and with $f(\cdot)$ decomposed as in (1.30),

$$\begin{aligned}\langle \mathsf{k}(\mathbf{x}, \cdot), f(\cdot) \rangle_{\mathsf{k}} &= \left\langle \sum_{j=1}^{n} \lambda_j e_j(\mathbf{x}) e_j(\cdot), \sum_{k=1}^{n} f_k e_k(\cdot) \right\rangle_{\mathsf{k}} = \sum_{j=1}^{n} \frac{\lambda_j e_j(\mathbf{x}) f_j}{\lambda_j} \\ &= f(\mathbf{x}). \end{aligned} \tag{1.32}$$

Moreover, choosing $f(\cdot)$ to be $\mathsf{k}(\mathbf{y}, \cdot)$, $f_j = \lambda_j e_j(\mathbf{y})$ from (1.28), and, hence,

$$\langle \mathsf{k}(\mathbf{x}, \cdot), \mathsf{k}(\mathbf{y}, \cdot) \rangle_{\mathsf{k}} = \sum_{j=1}^{n} \lambda_j e_j(\mathbf{x}) e_j(\mathbf{y}) = \mathsf{k}(\mathbf{x}, \mathbf{y}). \tag{1.33}$$

Together, (1.32) and (1.33) enable the evaluation of inner products in $\langle \cdot, \cdot \rangle_{\mathsf{k}}$ *without needing to know the original basis functions* $e_1(\cdot), \ldots, e_n(\cdot)$ *nor the associated values* $\lambda_1, \ldots, \lambda_n$. Indeed, we do not even need to know ν. This is known as *the kernel trick*, and we will exemplify its use in Section 1.5.5. First, we generalise to a much broader class of kernels.

1.5.4 General Kernels

In Section 1.5.2 we created a kernel via (1.27) using a known orthonormal basis for the inner-product space $\mathbb{V}$, with the inner product specifed by (1.23) according to the density ν. However, a kernel is *any* positive-definite symmetric function and we are interested in kernels $\mathsf{k} : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$.

Example 1.14 The Gaussian kernel is

$$\mathsf{k}(\mathbf{x}, \mathbf{y}) = \exp\left(-\|\mathbf{y} - \mathbf{x}\|^2\right),$$

where $\|\cdot\|$ represents the standard Euclidean norm. This is clearly symmetric. To see that k is also positive semidefinite on $\mathcal{X} = \mathbb{R}^d$, note that

$$\mathbf{Z} \sim \mathsf{N}_d\left(\mathbf{x}, \frac{1}{4}\mathbf{I}_d\right) \quad \text{and} \quad \mathbf{Y}|\mathbf{Z} \sim \mathsf{N}_d\left(\mathbf{Z}, \frac{1}{4}\mathbf{I}_d\right) \implies \mathbf{Y} \sim \mathsf{N}_d\left(\mathbf{x}, \frac{1}{2}\mathbf{I}_d\right),$$

from which

$$\exp\left(-\|\mathbf{y} - \mathbf{x}\|^2\right) = \gamma \int \exp\left(-2\|\mathbf{y} - \mathbf{z}\|^2\right) \exp\left(-2\|\mathbf{x} - \mathbf{z}\|^2\right) \mathrm{d}\mathbf{z},$$

where $\gamma = 2^d/\pi^{d/2}$. Hence $\sum_{j=1}^{J}\sum_{k=1}^{J} c_j c_k \mathsf{k}(\mathbf{x}_j, \mathbf{x}_k)$ is

$$\begin{aligned}
&\gamma \sum_{j=1}^{J}\sum_{k=1}^{J} c_j c_k \int \exp\left(-2\|\mathbf{x}_j - \mathbf{z}\|^2\right) \exp\left(-2\|\mathbf{x}_k - \mathbf{z}\|^2\right) \mathrm{d}\mathbf{z} \\
&\qquad = \gamma \int \sum_{j=1}^{J}\sum_{k=1}^{J} c_j c_k \exp\left(-2\|\mathbf{x}_j - \mathbf{z}\|^2\right) \exp\left(-2\|\mathbf{x}_k - \mathbf{z}\|^2\right) \mathrm{d}\mathbf{z} \\
&\qquad = \gamma \int \left\{\sum_{j=1}^{J} c_j \exp\left(-2\|\mathbf{x}_j - \mathbf{z}\|^2\right)\right\}^2 \mathrm{d}\mathbf{z} \\
&\qquad \geq 0.
\end{aligned}$$

When specifying k in Example 1.14, we have not specified a vector space, nor a density ν, nor an associated inner product. However, since k is a kernel, we might hope that if we do specify ν and the inner product $\langle\cdot,\cdot\rangle_\nu$ in (1.23), then there might be a vector space with a basis that is orthonormal with respect to $\langle\cdot,\cdot\rangle_\nu$ such that k has the decomposition (1.27). If this were the case then we would know that there was a new inner product $\langle\cdot,\cdot\rangle_{\mathsf{k}}$ such that (1.32) and (1.33) held. Hence we could evaluate inner products with respect to k without knowing the basis itself nor the eigenvalues of T_{k}, nor, even, the details about ν.

The decomposition in (1.23) does not hold in general, but Mercer's Theorem and generalisations of it tell us that an analogous decomposition but with n potentially infinite holds widely.

Specifically, let $\mathcal{X}$ be $\mathbb{R}^d$ or a closed or open subset of $\mathbb{R}^d$, $\mathsf{k}(\cdot,\cdot) : \mathcal{X}\times\mathcal{X} \to \mathbb{R}$ be a kernel and $\nu(\mathbf{x})$, $\mathbf{x}\in\mathcal{X}$, be a probability density on $\mathcal{X}$. Then, provided $\mathsf{k}(\mathbf{x},\mathbf{y})$ is a continuous function of $\mathbf{x}$ and $\mathbf{y}$, and

$$\int \mathsf{k}(\mathbf{x},\mathbf{y})^2 \nu(\mathbf{y})\, \mathrm{d}\mathbf{y} < \infty \quad \text{for all } \mathbf{x}\in\mathcal{X}, \tag{1.34}$$

the linear operator T_{k} defined in (1.29) has at most countably many positive (and no negative) eigenvalues $\lambda_1, \lambda_2, \ldots$ with corresponding eigenfunctions $e_1(\cdot), e_2(\cdot), \ldots$ which are orthonormal with respect to the inner product $\langle\cdot,\cdot\rangle_\nu$ defined in (1.23). Furthermore, k can be decomposed as

$$\mathsf{k}(\mathbf{x},\mathbf{y}) = \sum_{j=1}^{\infty} \lambda_k e_j(\mathbf{x}) e_j(\mathbf{y})$$

and the set $\{\sqrt{\lambda_j} e_j(\cdot)\}_{j=1}^{\infty}$ forms an orthonormal basis with respect to the

inner product

$$\langle f(\cdot), g(\cdot)\rangle_{\mathsf{k}} = \sum_{j=1}^{\infty} \frac{f_j g_j}{\lambda_j},$$

where

$$f(\cdot) = \sum_{j=1}^{\infty} f_j e_j(\cdot) \quad \text{and} \quad g(\cdot) = \sum_{j=1}^{\infty} g_j e_j(\cdot). \tag{1.35}$$

The space in which $e_1(\cdot), e_2(\cdot), \ldots$ lie is a generalisation of the vector space of Example 1.9 to the *Hilbert space*, $\mathcal{H}_\nu$, of functions $f(\cdot) : \mathcal{X} \to \mathbb{R}$ with the inner product $\langle \cdot, \cdot \rangle_\nu$ and such that $\|f(\cdot)\|_\nu^2 = \int f(\mathbf{x})^2 \nu(\mathbf{x})\, \mathrm{d}\mathbf{x} < \infty$. Likewise the orthonormal basis $\{\sqrt{\lambda_j} e_j(\cdot)\}_{j=1}^{\infty}$ lies in the *reproducing kernel Hilbert space*, $\mathcal{H}_{\mathsf{k}}$, of functions with the inner product $\langle \cdot, \cdot \rangle_{\mathsf{k}}$ and such that $\|f(\cdot)\|_{\mathsf{k}} < \infty$. A *Hilbert space* $\mathcal{H}$ is an inner product space with a potentially infinite set of basis vectors that is *complete*; informally, it contains no "holes", so that for any sequence $f_1, f_2, \ldots$ with $\sum_{j=1}^{\infty} f_j^2 < \infty$ then (considering $\mathcal{H}_{\mathsf{k}}$, for example) $f(\cdot) = \lim_{n\to\infty} \sum_{j=1}^{n} f_j e'_j(\cdot)$ exists, with distance measured through the norm induced by the inner product, and is in $\mathcal{H}_{\mathsf{k}}$.

Thus, the simplifications of the inner products in (1.32) and (1.33) continue to hold; in general, the intermediate steps must replace n with ∞.

Example 1.15 For the Gaussian kernel of Example 1.14, $\mathsf{k}(\mathbf{x}, \cdot) = \exp(-\|\mathbf{x} - \cdot\|^2)$ and

$$\left\langle \exp(-\|\mathbf{x} - \cdot\|^2), \exp(-\|\mathbf{y} - \cdot\|^2) \right\rangle_{\mathsf{k}} = \exp(-\|\mathbf{y} - \mathbf{x}\|^2).$$

Also, for any $f(\cdot) \in \mathcal{H}_{\mathsf{k}}$,

$$\left\langle \exp(-\|\mathbf{x} - \cdot\|^2), f(\cdot) \right\rangle_{\mathsf{k}} = f(\mathbf{x}).$$

Trace-Class Kernels

A kernel where $\int \mathsf{k}(\mathbf{x}, \mathbf{x}) \nu(\mathbf{x})\, \mathrm{d}\mathbf{x} = c < \infty$ is referred to as *trace class*. This property has important consequences for the set of eigenvalues, $\lambda_1, \lambda_2, \ldots$, of T_{k}, since

$$\begin{aligned} \int \mathsf{k}(\mathbf{x}, \mathbf{x}) \nu(\mathbf{x})\, \mathrm{d}\mathbf{x} &= \int \sum_{k=1}^{\infty} \lambda_k e_k(\mathbf{x}) e_k(\mathbf{x}) \nu(\mathbf{x})\, \mathrm{d}\mathbf{x} \\ &= \sum_{k=1}^{\infty} \lambda_k \int e_k(\mathbf{x})^2 \nu(\mathbf{x})\, \mathrm{d}\mathbf{x} = \sum_{k=1}^{\infty} \lambda_k. \end{aligned}$$

Thus $\sum_{k=1}^{\infty} \lambda_k = c$. Since each $\lambda_k \geq 0$, we have $\lim_{k\to\infty} \lambda_k = 0$.

The Gaussian kernel of Example 1.14 is trace class with $c = 1$ since $\mathsf{k}(\mathbf{x}, \mathbf{x}) = 1$ for all $\mathbf{x} \in \mathbb{R}^d$. The kernel we will meet in in Chapter 6 is also of trace class, following a similar reasoning.

Without loss of generality, we label the eigenvalues $\lambda_1, \lambda_2 \ldots$ in order of decreasing size (choosing any one of the possibilities if some of the λ_j are not unique). With the decomposition of $f(\cdot)$ in (1.35),

$$\|f(\cdot)\|_{\mathsf{k}}^2 = \sum_{j=1}^{\infty} \frac{f_j^2}{\lambda_j} \geq \frac{1}{\lambda_1} \sum_{j=1}^{\infty} f_j^2 = \frac{1}{\lambda_1} \|f(\cdot)\|_{\nu}^2.$$

Thus $\|f(\cdot)\|_{\mathsf{k}} < \infty \implies \|f(\cdot)\|_{\nu} < \infty$ and hence $\mathcal{H}_{\mathsf{k}} \subseteq \mathcal{H}_{\nu}$. In general, $\mathcal{H}_{\mathsf{k}}$ is strictly smaller than $\mathcal{H}_{\nu}$ and the more quickly the eigenvalues of T_{k} decay the smaller the space $\mathcal{H}_{\mathsf{k}}$.

1.5.5 The Power of the Kernel Trick

Suppose we have values $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \mathcal{X}$ and we are interested in

$$\mathbb{V}^* = \left\{ g(\cdot) : g(\cdot) = \sum_{j=1}^{m} g_j \mathsf{k}(\mathbf{x}_j, \cdot),\ g_1, \ldots, g_m \in \mathbb{R} \right\}.$$

Firstly, for any $g(\cdot) = \sum_{j=1}^{m} g_j \mathsf{k}(\mathbf{x}_j, \cdot)$,

$$\begin{aligned} \|g(\cdot)\|_{\mathsf{k}}^2 &= \left\langle \sum_{j=1}^{m} g_j \mathsf{k}(\mathbf{x}_j, \cdot), \sum_{k=1}^{m} g_k \mathsf{k}(\mathbf{x}_k, \cdot) \right\rangle_{\mathsf{k}} = \sum_{j=1}^{m} \sum_{k=1}^{m} g_j \left\langle \mathsf{k}(\mathbf{x}_j, \cdot), \mathsf{k}(\mathbf{x}_k, \cdot) \right\rangle g_k \\ &= \sum_{j=1}^{m} \sum_{k=1}^{m} g_j \mathsf{k}(\mathbf{x}_j, \mathbf{x}_k) g_k < \infty. \end{aligned} \tag{1.36}$$

So, $\mathbb{V}^* \subseteq \mathcal{H}_{\mathsf{k}}$. Secondly, for any $f(\cdot) \in \mathcal{H}_{\mathsf{k}}$,

$$\langle f(\cdot), g(\cdot) \rangle_{\mathsf{k}} = \sum_{j=1}^{m} g_j \left\langle f(\cdot), \mathsf{k}(\mathbf{x}_j, \cdot) \right\rangle_{\mathsf{k}} = \sum_{j=1}^{m} g_j f(\mathbf{x}_j). \tag{1.37}$$

Suppose there is a particular function of interest, $f(\cdot) \in \mathcal{H}_{\mathsf{k}}$, and we would like to construct the function $g(\cdot) \in \mathbb{V}^*$ that most closely resembles $f(\cdot)$ in shape. We could find the unit vector in $\mathbb{V}^*$ which has the largest component in the $f(\cdot)$ direction:

$$\widehat{g}(\cdot) = \underset{g(\cdot) \in \mathbb{V}^* : \|g(\cdot)\| = 1}{\arg\max} \langle f(\cdot), g(\cdot) \rangle_{\mathsf{k}}.$$

The size of the inner product, $\langle f(\cdot), \widehat{g}(\cdot) \rangle_{\mathsf{k}}$, is a measure of the ability of $\mathbb{V}^*$ to represent $f(\cdot)$.

Define $\mathbf{f} = [f(\mathbf{x}_1), \ldots, f(\mathbf{x}_m)]^\top$ and $\mathbf{g} = [g_1, \ldots g_m]^\top$ and let $\mathbf{K}$ be the matrix with elements $K_{i,j} = \mathsf{k}(\mathbf{x}_i, \mathbf{x}_j)$. Then (1.36) and (1.37) become

$$\langle f(\cdot), g(\cdot) \rangle_{\mathsf{k}} = \mathbf{g}^\top \mathbf{f} \quad \text{and} \quad \|g(\cdot)\|_{\mathsf{k}}^2 = \mathbf{g}^\top \mathbf{K} \mathbf{g}.$$

To find $\widehat{g}(\cdot)$ we must find the vector $\widehat{\mathbf{g}}$ that maximises $\mathbf{g}^\top \mathbf{f}$ subject to $\mathbf{g}^\top \mathbf{K} \mathbf{g} = 1$.

Let $\mathbf{A}$ be a square matrix such that $\mathbf{A}\mathbf{A}^\top = \mathbf{K}$ and set $\mathbf{h} = \mathbf{A}^\top \mathbf{g}$. Then, equivalently, we wish to maximise $\mathbf{h}^\top \mathbf{A}^{-1} \mathbf{f}$ such that $\|\mathbf{h}\| = 1$. We must find the unit m-vector with the largest component in the $\mathbf{A}^{-1}\mathbf{f}$ direction, which is

$$\widehat{\mathbf{h}} = \frac{\mathbf{A}^{-1}\mathbf{f}}{\sqrt{(\mathbf{A}^{-1}\mathbf{f})^\top \mathbf{A}^{-1}\mathbf{f}}} \implies \widehat{\mathbf{g}} = \frac{\mathbf{A}^{-\top}\mathbf{A}^{-1}\mathbf{f}}{\sqrt{\mathbf{f}^\top \mathbf{A}^{-\top}\mathbf{A}^{-1}\mathbf{f}}} = \frac{\mathbf{K}^{-1}\mathbf{f}}{\sqrt{\mathbf{f}^\top \mathbf{K}^{-1}\mathbf{f}}},$$

since $\widehat{\mathbf{g}} = \mathbf{A}^{-\top}\widehat{\mathbf{h}}$. The inner product $\widehat{\mathbf{g}}^\top \mathbf{f}$ is

$$\frac{\mathbf{f}^\top \mathbf{K}^{-1}\mathbf{f}}{\sqrt{\mathbf{f}^\top \mathbf{K}^{-1}\mathbf{f}}} = \sqrt{\mathbf{f}^\top \mathbf{K}^{-1}\mathbf{f}}.$$

This calculation *only* requires us to be able to evaluate $f(\mathbf{x}_j)$ and $\mathsf{k}(\mathbf{x}_j, \mathbf{x}_k)$ for $j, k = 1, \ldots, m$. We do not need to know the eigenfunctions $e_1(\cdot), \ldots$ nor eigenvalues $\lambda_1, \ldots$ of T_{k}. Indeed, we do not even need to know ν; only that (1.34) is satisfied.

Example 1.16 Let $\mathcal{X} = \mathbb{R}$ and let k be the one-dimensional case of the Gaussian kernel in Example 1.14. We find the approximations to the function

$$f(x) = \frac{1}{1 + x^2},$$

using gradually more and more kernel functions $\mathsf{k}(x_j, x)$. For points, $x_1, \ldots, x_J$, $\mathbf{K}$ is the matrix with elements $K_{i,j} = \exp[-(x_i - x_j)^2]$, and $\mathbf{f}$ is the vector with $f_j = f(x_j)$. We set $(x_1, x_2, x_3, x_4, x_5, x_6, x_7) = (-3, \ldots, 3)$ and approximate $f(x)$ using just x_1 then $x_1, \ldots, x_3$, then $x_1, \ldots, x_5$ and finally $x_1, \ldots, x_7$. Figure 1.6 compares the four approximations with the truth. Each time new points are added to the set, the approximation improves, but it matters where the points are added; some basis vectors are more helpful than others.

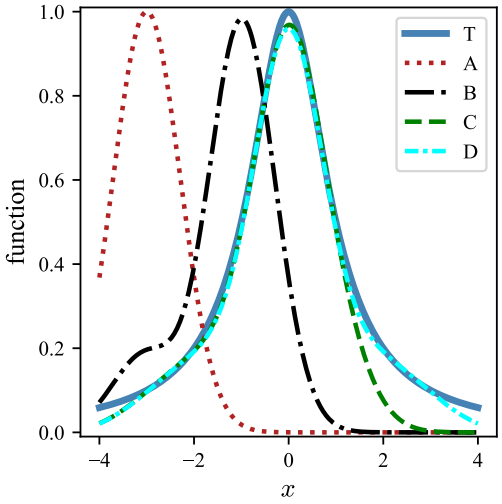

Figure 1.6 The function $f(x) = 1/(1+x^2)$ (T) and kernel-based approximations to $f(x)$ from Example 1.16. Curves use A: $x = -3$, B: $x = -3, -2, -1$, C: $x = -3, \ldots, 1$ and D: $x = -3, \ldots, 3$.

1.6 Chapter Notes

There are many texts which cover the introductory material from this chapter in more depth and rigour than we have allowed; we suggest a few on each topic.

Basic Monte Carlo and importance sampling is covered in Ripley (2009) and Rubinstein and Kroese (2008). For an introduction to Bayesian statistics and the use of Monte Carlo methods for Bayesian analysis, see Bernardo and Smith (2009), Robert (2007) and Robert and Casella (1999).

Norris (1998) provides a gentle introduction to Markov chains on discrete state spaces, while Meyn and Tweedie (2012) gives a thorough treatment on general state spaces; a less thorough but more readily accessible treatment for general state spaces is given in Roberts and Rosenthal (2004). Geyer (1992) describes methods for estimating the integrated auto-correlation time from a sample of the chain when the Markov chain is reversible; for the non-reversible chains of Chapter 4 the integrated auto-correlation can be estimated by fitting an auto-regressive process to the time series $\{h(X_k)\}_{k=1}^n$

or by estimating the spectral density of the series at a frequency of 0 (e.g. Heidelberger and Welch, 1981).

Stochastic differential equations and diffusions are the subject of Oksendal (2013), Rogers and Williams (2000a) and Rogers and Williams (2000b). An alternative to simple Monte Carlo, which attempts to obtain better convergence rates with the Monte Carlo sample size n, is quasi-Monte Carlo. See, for example, Caflisch (1998) for an introduction and L'Ecuyer and Lemieux (2002) for work on randomised quasi-Monte Carlo.

Chapter 1 of Conway (2010) introduces Hilbert spaces in general, and kernels and reproducing kernel Hilbert spaces are covered in Chapter 6 of Rasmussen and Williams (2005). Mercer's Theorem is usually stated for a compact $\mathcal{X}$; we have used the generalisation to non-compact spaces in Sun (2005).

2

Reversible MCMC and its Scaling

Building on the introductions to Bayesian statistics, Monte Carlo methods and Markov chains in Chapter 1, this section introduces Markov chain Monte Carlo algorithms as a generic computational solution to the challenge of using Monte Carlo methods to sample from the posterior distribution and, hence, estimate posterior expectations of quantities of interest.

As described in Chapter 1, if it is possible to sample directly from the posterior, $\pi(\boldsymbol{\theta}) := \pi(\boldsymbol{\theta}|\mathcal{D})$ (see equation (1.3)) then for any function h with $\mathbb{E}_\pi\left[h^2(\boldsymbol{\theta})\right] < \infty$, it is possible to estimate $\mathbb{E}_\pi\left[h(\boldsymbol{\theta})\right]$ via the Monte Carlo average (1.4), the typical error of which is of size $n^{-1/2}$, where n is the number of samples. Unfortunately, it is usually not possible, or is computationally infeasible, to generate independent and identically distributed samples from π. Importance sampling provides, perhaps, the most natural alternative to direct sampling; however, as exemplified in Section 1.1.5 the variance of importance sampling estimators typically degrades exponentially quickly with dimension.

Markov chain Monte Carlo (MCMC) is a generalisation of the Monte Carlo method that, as we will see, has several favourable properties when it comes to facilitating computation in Bayesian statistics problems. In this context, the aim of MCMC is to construct a Markov chain, $\{\boldsymbol{\theta}_k\}_{k=1}^{\infty}$ whose limiting distribution is the posterior distribution of interest, so that samples from a sufficiently long chain, except, perhaps, those near the beginning, arise approximately from the posterior and can be used to create Monte Carlo approximations to expectations as in (1.4), via the ergodic average defined in (1.10).

The workhorse of MCMC is the Metropolis–Hastings algorithm. We describe the general Metropolis–Hastings algorithm and show that the resulting chain satisfies detailed balance with respect to π. We then investigate particular special cases: the independence sampler, the random walk Metropolis algorithm, the Metropolis-adjusted Langevin algorithm, and Hamiltonian Monte Carlo. For each of these cases, we overview the be-

haviour as the dimension $d \to \infty$, motivating the need for further scalable methods.

Throughout this chapter, we denote the support of π by Θ; for example Θ might be $\mathbb{R}^d$ for some $d \in \mathbb{N}$.

2.1 The Metropolis–Hastings Algorithm

The idea of the *Metropolis–Hastings* algorithm is to define the dynamics of a Markov chain by specifying an arbitrary proposal distribution for the next state of the Markov chain, and then having a mechanism where this proposal is either accepted or rejected. If it is rejected, the state of the Markov chain is unchanged. As we will see, it is generally possible to choose the acceptance probability to depend on the target distribution, so that the resulting Markov chain will have the target distribution as its stationary distribution.

The Metropolis–Hastings algorithm is given in Algorithm 1. The posterior density, π, appears in both the numerator and denominator of the acceptance probability, $\alpha(\boldsymbol{\theta}_k, \boldsymbol{\theta}')$, so terms involving the typically intractable density $\pi(\boldsymbol{\theta})$ can be replaced with the tractable product of the prior and the likelihood, $\pi_0(\boldsymbol{\theta}) L(\boldsymbol{\theta}; \mathcal{D})$; we do not need to know the normalising constant, $p(\mathcal{D}) = \int_\Theta \pi_0(\boldsymbol{\theta}) L(\boldsymbol{\theta}; \mathcal{D}) \, \mathrm{d}\boldsymbol{\theta}$, as it will cancel in the ratio.

As well as $\pi(\boldsymbol{\theta})$, and an initial value for the parameter vector, the Metropolis–Hastings algorithm requires a proposal density, $q(\boldsymbol{\theta}'|\boldsymbol{\theta})$. Common choices of the density q include the following:

Metropolis–Hastings independence sampler (MHIS) $q(\boldsymbol{\theta}' \mid \boldsymbol{\theta}) := q'(\boldsymbol{\theta}')$ for some density q'. The proposal does not depend on the current state; for example, q' could be the same as a sensible importance sampling proposal distribution (see Section 1.1.5).

Random walk Metropolis (RWM) $q(\boldsymbol{\theta}' \mid \boldsymbol{\theta}) = q'(\boldsymbol{\theta}' - \boldsymbol{\theta})$, where q' is a density such that for any vector $\mathbf{v} \in \Theta$, $q'(\mathbf{v}) = q'(-\mathbf{v})$. For example $\boldsymbol{\theta}'|\boldsymbol{\theta} \sim \mathsf{N}(\boldsymbol{\theta}, \lambda^2 \mathbf{I}_d)$, where $\mathbf{I}_d$ is the $d \times d$ identity matrix and $\lambda > 0$.

Metropolis-adjusted Langevin algorithm (MALA) adds a specific form of offset to a Gaussian RWM proposal. For example, $\boldsymbol{\theta}'|\boldsymbol{\theta} \sim \mathsf{N}(\boldsymbol{\theta} + \frac{1}{2}\lambda^2 \nabla \log \pi(\boldsymbol{\theta}), \lambda^2 \mathbf{I}_d)$.

Hamiltonian Monte Carlo (HMC) starting from $\boldsymbol{\theta}$ and with a random momentum, such as $\mathbf{p} \sim \mathsf{N}(\mathbf{0}, M\mathbf{I}_d)$, Hamiltonian dynamics are approximately integrated forwards on a potential surface of $-\log \pi(\boldsymbol{\theta})$. The proposal, $\boldsymbol{\theta}'$, is the position after some fixed time T.

Later in this chapter we describe and investigate these classes of proposals in more detail and examine their relative efficiencies.

Algorithm 1: Metropolis–Hastings algorithm

Input: Density $\pi(\boldsymbol{\theta})$, initial value $\boldsymbol{\theta}_0$ and proposal density $q(\boldsymbol{\theta}'|\boldsymbol{\theta})$.
for $k \in 0, \ldots, n-1$ **do**
  Propose $\boldsymbol{\theta}'$ from $q(\boldsymbol{\theta}'|\boldsymbol{\theta}_k)$
  Calculate the acceptance probability:

$$\alpha(\boldsymbol{\theta}_k, \boldsymbol{\theta}') := \min\left(1, \frac{\pi(\boldsymbol{\theta}')q(\boldsymbol{\theta}_k|\boldsymbol{\theta}')}{\pi(\boldsymbol{\theta}_k)q(\boldsymbol{\theta}'|\boldsymbol{\theta}_k)}\right). \tag{2.1}$$

  With a probability of $\alpha(\boldsymbol{\theta}_k, \boldsymbol{\theta}')$ accept the proposal, $\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}'$; otherwise reject it, $\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k$.
end

That the Metropolis–Hastings algorithm has a stationary density of π, follows directly from the fact that it is reversible with respect to π (see Section 1.3). We now show that this is the case. First, notice that

$$\pi(\boldsymbol{\theta})q(\boldsymbol{\theta}'|\boldsymbol{\theta})\alpha(\boldsymbol{\theta}, \boldsymbol{\theta}') = \pi(\boldsymbol{\theta}')q(\boldsymbol{\theta}|\boldsymbol{\theta}')\alpha(\boldsymbol{\theta}', \boldsymbol{\theta}),$$

since both are $\min\left(\pi(\boldsymbol{\theta})q(\boldsymbol{\theta}'|\boldsymbol{\theta}), \pi(\boldsymbol{\theta}')q(\boldsymbol{\theta}|\boldsymbol{\theta}')\right)$. Now suppose that $\boldsymbol{\theta}_k \sim \pi$, let $\mathcal{B}, C \subseteq \Theta$ and let A be the event that the proposal is accepted. Then

$$\begin{aligned}
\mathbb{P}\left(\mathsf{A}, \boldsymbol{\theta}_k \in \mathcal{B}, \boldsymbol{\theta}_{k+1} \in C\right) &= \int_{\boldsymbol{\theta}_k \in \mathcal{B}} \int_{\boldsymbol{\theta}' \in C} \pi(\boldsymbol{\theta}_k) q(\boldsymbol{\theta}' \mid \boldsymbol{\theta}_k) \alpha(\boldsymbol{\theta}_k, \boldsymbol{\theta}') \, \mathrm{d}\boldsymbol{\theta}' \mathrm{d}\boldsymbol{\theta}_k \\
&= \int_{\boldsymbol{\theta}_k \in \mathcal{B}} \int_{\boldsymbol{\theta}' \in C} \pi(\boldsymbol{\theta}') q(\boldsymbol{\theta}_k \mid \boldsymbol{\theta}') \alpha(\boldsymbol{\theta}', \boldsymbol{\theta}_k) \, \mathrm{d}\boldsymbol{\theta}_k \mathrm{d}\boldsymbol{\theta}' \\
&= \int_{\boldsymbol{\theta}' \in \mathcal{B}} \int_{\boldsymbol{\theta}_k \in C} \pi(\boldsymbol{\theta}_k) q(\boldsymbol{\theta}' \mid \boldsymbol{\theta}_k) \alpha(\boldsymbol{\theta}_k, \boldsymbol{\theta}') \, \mathrm{d}\boldsymbol{\theta}_k \mathrm{d}\boldsymbol{\theta}' \\
&= \mathbb{P}\left(\mathsf{A}, \boldsymbol{\theta}_k \in C, \boldsymbol{\theta}_{k+1} \in \mathcal{B}\right)
\end{aligned}$$

where, on the penultimate line we have switched the labels. Since $\boldsymbol{\theta}_k = \boldsymbol{\theta}_{k+1}$ on a rejection, we also have that

$$\mathbb{P}\left(\mathsf{A}^{\complement}, \boldsymbol{\theta}_k \in \mathcal{B}, \boldsymbol{\theta}_{k+1} \in C\right) = \mathbb{P}\left(\mathsf{A}^{\complement}, \boldsymbol{\theta}_k \in C, \boldsymbol{\theta}_{k+1} \in \mathcal{B}\right).$$

Summing the two equalities above for A and $\mathsf{A}^{\complement}$ gives

$$\mathbb{P}\left(\boldsymbol{\theta}_k \in \mathcal{B}, \boldsymbol{\theta}_{k+1} \in C\right) = \mathbb{P}\left(\boldsymbol{\theta}_k \in C, \boldsymbol{\theta}_{k+1} \in \mathcal{B}\right),$$

as required.

Burn-In, Mixing, Estimators and their Variance

Typically, the initial value for the Markov chain is not sampled from π, since if it were possible to do this then there would be no need for MCMC. Hence, the marginal distributions of early points in the Markov chain might not be sufficiently close to π. In practice, we discard such early points from the sample; the terms *warm-up* or *burn-in* are applied to both this initial period and the samples that arise from it. Here, we imagine that there are b burn-in samples, $\boldsymbol{\theta}_{-b+1}, \ldots \boldsymbol{\theta}_0$ and that the remaining samples, which are deemed to be from a chain that has approximately converged, are $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_n$. The expectation of any function $h(\boldsymbol{\theta})$ with respect to the posterior is then estimated via:

$$\widehat{I}_n(h) := \frac{1}{n} \sum_{k=1}^{n} h(\boldsymbol{\theta}_k). \tag{2.2}$$

Following the exposition in Section 1.3.2, subject to conditions, including that $\mathbb{E}_\pi \left[h(\boldsymbol{\theta})^2 \right] < \infty$, the variance of this estimator may be approximated as in (1.15); this is an approximation both because the Markov chain has not fully converged after the b burn-in iterations, and because n is finite. As with standard Monte Carlo estimates, the standard error decreases as $n^{-1/2}$; however, the constant of proportionality is (typically) higher, reflecting the fact that the samples are (typically) positively correlated.

In most MCMC algorithms, the positive correlation arises from two separate sources: firstly, a proposal may be rejected, in which case the new position of the chain is the same as the old position; secondly most types of Metropolis-Hastings algorithm are *local*: the proposal is, in some sense, close to the current value when compared to the size of the main posterior mass. Consequently, the chain can take many iterations to move from one part of the posterior to another. The act of moving around the posterior is termed *mixing* and in this book we informally refer to the number of iterations taken to move substantially within the context of the posterior distribution as the *mixing time*.

Running Example

The following running example of an isotropic Gaussian target distribution serves to demonstrate some properties of the Metropolis–Hastings algorithm in practice. In the next section, it will be used to illustrate the different measures of efficiency of a Metropolis–Hastings Markov chain algorithm.

Example 2.1 Given $d \in \mathbb{N}$, and $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_d)^\top$, let

$$\pi_d(\boldsymbol{\theta}) = \mathsf{N}(\boldsymbol{\theta}; 0, \mathbf{I}_d) \equiv \frac{1}{(2\pi)^{d/2}} \exp\left(-\frac{1}{2}\sum_{i=1}^{d}\theta_i^2\right) \equiv \frac{1}{(2\pi)^{d/2}} \exp\left(-\frac{1}{2}\|\boldsymbol{\theta}\|^2\right),$$

where $\|\cdot\|$ denotes the Euclidean norm, and $\mathbf{I}_d$ the $d \times d$ identity matrix.

For now, we explore the above target using the RWM algorithm described above:

$$q(\boldsymbol{\theta}'|\boldsymbol{\theta}) = \mathsf{N}\left(\boldsymbol{\theta}'; \boldsymbol{\theta}, \lambda^2\mathbf{I}_d\right) \equiv \frac{1}{(2\pi)^{d/2}\lambda^d} \exp\left(-\frac{1}{2\lambda^2}\|\boldsymbol{\theta}' - \boldsymbol{\theta}\|^2\right).$$

Figure 2.1 shows plots from $n = 1000$ iterations of the algorithm in Example 2.1 with $d = 1$. The leftmost plot starts from $\boldsymbol{\theta}_0 = 20$ while the other two start from $\boldsymbol{\theta}_0 = 1$. More than 99.7% of the posterior mass lies between $\boldsymbol{\theta} = -3$ and $\boldsymbol{\theta} = 3$, and so the chain that was started outside of this region first heads towards the main mass. Once it has arrived, it then explores the region for the remainder of the time, n. The exploration is slow because the scale of the proposed jumps, $\lambda = 0.2$, is small compared with the size of the region. With larger proposed jumps, $\lambda = 2$, the exploration is much more rapid. However, with jumps of size $\lambda = 20$, most of the proposals are outside of the high-density region, so the acceptance ratio is small and the proposals are rejected. Thus, even though the proposed jumps are large, the algorithm does not explore the range of posterior values quickly.

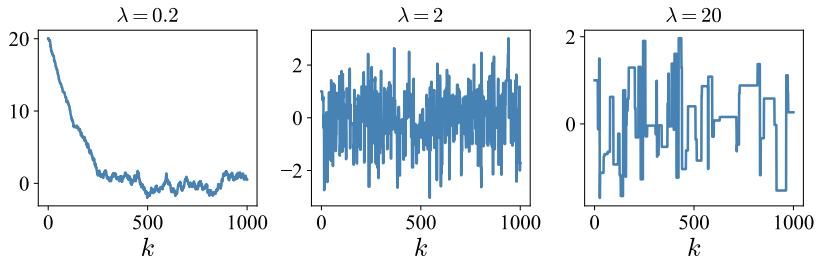

Figure 2.1 Trace plots from three RWM runs on π_d from Example 2.1 with $d = 1$, using initial values of 20, 1 and 1, and scalings of 0.2, 2 and 20 respectively.

Theory tells us that the distribution of $\boldsymbol{\theta}_k$ converges to π as $k \to \infty$. For a finite k, $\boldsymbol{\theta}_k$ will not be an exact draw from π but it might be close. In Figure 2.1 (left) we might deem the distribution sufficiently close after

approximately 300 iterations and so we might discard $\boldsymbol{\theta}_0, \ldots, \boldsymbol{\theta}_{299}$ as *burn-in* and take $\{\boldsymbol{\theta}_{300}, \ldots, \boldsymbol{\theta}_{1000}\}$ to be an approximate, correlated, sample from π for use in a Monte Carlo average, $\widehat{\mu}_h$, of the form (1.4). The runs illustrated in Figure 2.1 (middle and right) started from a sensible value in the posterior and so $\{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{1000}\}$ might reasonably be used.

Let us now ignore the need for burn-in when $\boldsymbol{\theta}_0 = 20$ and $\lambda = 0.2$, or consider a thought experiment where this algorithm was also started from $\boldsymbol{\theta}_0 = 1$. From Figure 2.2, the sample obtained when $\lambda = 2$ appears to represent π, which is symmetric about a single mode at 0 and has support beyond ± 2, much better than either of the other two samples. Thus, we might expect estimates, $\widehat{\mu}_h$, obtained from the algorithm when $\lambda = 2$ to be, in some sense, more accurate.

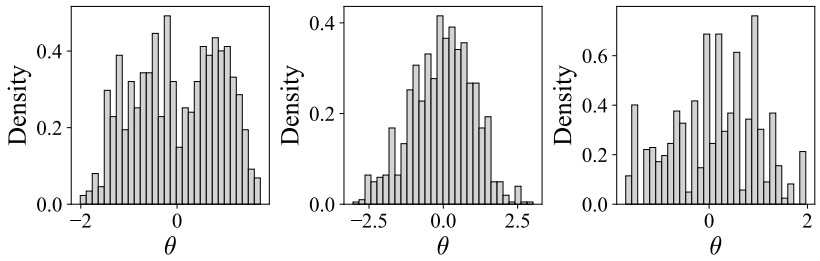

Figure 2.2 Histograms of the samples obtained from the algorithms in Figure 2.1. The left plot (corresponding to $\lambda = 0.2$, $\boldsymbol{\theta}_0 = 20$) was created after discarding $\{\boldsymbol{\theta}_0, \ldots, \boldsymbol{\theta}_{299}\}$ as burn-in; for the other two runs (centre: $\lambda = 2$; right: $\lambda = 20$) only $\boldsymbol{\theta}_0$ was discarded.

The *empirical acceptance rate* for a Metropolis–Hastings algorithm is the fraction of the n proposals that were accepted. For the three RWM algorithms, these were respectively 0.881, 0.485 and 0.059; the smaller the proposed jumps, the closer $\pi(\boldsymbol{\theta}')$ typically is to $\pi(\boldsymbol{\theta}_k)$ and so the higher the acceptance rate. For a stationary Metropolis–Hastings Markov chain, the empirical acceptance rate approximates the true acceptance rate at stationarity:

$$\alpha = \mathbb{E}_{\boldsymbol{\theta} \sim \pi, \boldsymbol{\theta}' \sim q(\cdot|\boldsymbol{\theta})} \left[\alpha(\boldsymbol{\theta}, \boldsymbol{\theta}')\right]$$

Aspects of the proposal for the RWM, MALA and HMC are often tuned by targeting an empirical acceptance rate that is neither too high nor too low. In later sections, the acceptance rate will provide us with an intuitive entrance into the behaviour of the canonical Metropolis–Hastings algorithms as the

dimension of the parameter vector increases. Here it will be helpful to define the *acceptance ratio*: $\rho(\boldsymbol{\theta}, \boldsymbol{\theta}') := \pi(\boldsymbol{\theta}')q(\boldsymbol{\theta}|\boldsymbol{\theta}')/\{\pi(\boldsymbol{\theta})q(\boldsymbol{\theta}'|\boldsymbol{\theta})\}$, so that $\alpha(\boldsymbol{\theta}, \boldsymbol{\theta}') = \min[1, \rho(\boldsymbol{\theta}, \boldsymbol{\theta}')]$.

2.1.1 Component-wise updates and Gibbs moves

Algorithm 1, the Metropolis–Hastings algorithm, and all of the special cases that we will examine in this chapter, are written so that a single iteration consists of a proposal to change the entire $\boldsymbol{\theta}$ vector and a decision on whether or not to accept this proposal. However, it is also possible, and sometimes helpful, to sequentially update subsets of the components of $\boldsymbol{\theta}$. Indeed, each iteration of the very first Metropolis–Hastings algorithm (Metropolis et al., 1953) cycled through pairs of components (x and y coordinates of each particle in a lattice of a large number of particles), applying a random walk Metropolis update one pair at a time.

Denote the set of components to be updated by $\boldsymbol{\theta}^{(i)}$ and the remaining components by $\boldsymbol{\theta}^{(-i)}$. By writing the target as $\pi(\boldsymbol{\theta}) = \pi(\boldsymbol{\theta}^{(-i)})\pi(\boldsymbol{\theta}^{(i)}|\boldsymbol{\theta}^{(-i)})$, and the proposal as $q_i(\boldsymbol{\theta}^{'(i)}|\boldsymbol{\theta}^{(i)}, \boldsymbol{\theta}^{(-i)})$, essentially the same argument as for a proposal that changes all components shows that the component-wise propose/accept-reject step with an acceptance probability of

$$\min\left(1, \frac{\pi(\boldsymbol{\theta}^{'(i)}|\boldsymbol{\theta}^{(-i)})q_i(\boldsymbol{\theta}^{(i)}|\boldsymbol{\theta}^{'(i)}, \boldsymbol{\theta}^{(-i)})}{\pi(\boldsymbol{\theta}^{(i)}|\boldsymbol{\theta}^{(-i)})q_i(\boldsymbol{\theta}^{'(i)}|\boldsymbol{\theta}^{(i)}, \boldsymbol{\theta}^{(-i)})}\right)$$

satisfies detailed balance with respect to the full posterior, π. Of course, if only that move were used, some components would never be updated, the algorithm would be reducible, and ergodic averages would, therefore, not converge to the corresponding true expectations. The composition of many such moves over different components, typically, does not satisfy this detailed balance condition, but, since each move preserves π, the composition does, too.

In the special case where the proposal for the ith block of components is

$$q_i(\boldsymbol{\theta}^{'(i)}|\boldsymbol{\theta}_k) := \pi(\boldsymbol{\theta}^{'(i)}|\boldsymbol{\theta}_k^{(-i)}),$$

the acceptance probability is 1 and the move is called a *Gibbs* move. Such a move is only feasible when it is possible to sample from $\pi(\boldsymbol{\theta}^{(i)}|\boldsymbol{\theta}_k^{(-i)})$ which most usually occurs when, conditional on $\boldsymbol{\theta}_k^{(-i)}$, the prior for $\boldsymbol{\theta}^{(i)}$ is conjugate with its likelihood. For example, when $y_1, \ldots, y_N$ are independent realisations from a $\mathsf{N}(\mu, 1/\tau)$ distribution and μ and τ have independent priors with μ following a Student-t distribution and $\tau \sim \mathsf{Gam}(a, b)$, then *a posteriori* $\tau|\mu \sim \mathsf{Gam}(a + n/2, b + \frac{1}{2}\sum_{j=1}^{n}(y_i - \mu)^2)$; this property is

sometimes called *conditional conjugacy*. Gibbs moves offer a further advantage when compared with many other Metropolis-Hastings moves, over and above the fact that the acceptance probability is 1: the updated parameter component is sampled from the full range of its conditional posterior. When components are close to independent, this contrasts with algorithms such as the random walk Metropolis and MALA, where, in moderate to high dimensions, the moves are local – the proposed value is close to the current value. However, when, as is typically the case, components are correlated, the conditional posterior for a component can have a much smaller range than its marginal posterior, and so the Gibbs moves, too, are, in effect, local. Whilst they are a useful tool in the MCMC armoury, Gibbs moves are not the focus of this book and for further information, we refer the interested reader to the general texts cited in Section 2.3.

2.1.2 The Metropolis–Hastings Independence Sampler

Consider the *Metropolis–Hastings independence sampler* (MHIS) in the case where $q(\boldsymbol{\theta}) = \pi(\boldsymbol{\theta})$. In this case $\alpha(\boldsymbol{\theta}_k, \boldsymbol{\theta}') = 1$ and every proposal is accepted. Since proposals are from π, the MHIS provides us with an i.i.d. sample from π. Of course, in practice, we are typically not able to sample from π, but this suggests a heuristic for the MHIS: choose a proposal so that the acceptance rate is as close as possible to 1.

For unimodal posteriors in low dimensions a reasonable approximation can often be obtained by first using a numerical method to find the posterior mode and then choosing a proposal that matches the mode and the curvature of the log-posterior at this point. This strategy does not scale favourably to high dimensions, however, as the following simple example shows.

Consider the isotropic unit Gaussian posterior from Example 2.1 and use the following MHIS proposal:

$$q(\boldsymbol{\theta}'|\boldsymbol{\theta}) = \mathsf{N}(\mathbf{0}, \sigma^2\mathbf{I}_d) \equiv \frac{1}{(2\pi)^{d/2}\sigma^d} \exp\left(-\frac{1}{2\sigma^2}\|\boldsymbol{\theta}'\|^2\right).$$

The acceptance ratio, $\rho(\boldsymbol{\theta}, \boldsymbol{\theta}')$, is

$$\frac{\exp(-\frac{1}{2}\|\boldsymbol{\theta}'\|^2)\exp(-\frac{1}{2\sigma^2}\|\boldsymbol{\theta}\|^2)}{\exp(-\frac{1}{2}\|\boldsymbol{\theta}\|^2)\exp(-\frac{1}{2\sigma^2}\|\boldsymbol{\theta}'\|^2)} = \exp\left\{\frac{1}{2}\left(1-\frac{1}{\sigma^2}\right)\left(\|\boldsymbol{\theta}\|^2 - \|\boldsymbol{\theta}'\|^2\right)\right\},$$

so

$$\frac{1}{d}\log\rho(\boldsymbol{\theta}, \boldsymbol{\theta}') = \frac{1}{2}\left(1-\frac{1}{\sigma^2}\right)\left(\frac{1}{d}\|\boldsymbol{\theta}\|^2 - \frac{1}{d}\|\boldsymbol{\theta}'\|^2\right).$$

If the chain is stationary, then $\|\boldsymbol{\theta}\|^2 = \sum_{i=1}^{d} \theta_i^2 \sim \chi_d^2$ since $\theta_i \overset{iid}{\sim} \mathsf{N}(0,1)$. Thus $\mathbb{E}\left[\|\boldsymbol{\theta}\|^2/d\right] = 1$ and $\mathrm{Var}\left[\|\boldsymbol{\theta}\|^2/d\right] = 2/d$, and the same properties hold for $\|\boldsymbol{\theta}'/\sigma\|^2/d$. Thus, in high dimensions, to a first-order approximation, $\|\boldsymbol{\theta}\|^2/d \approx 1$ and $\|\boldsymbol{\theta}'\|^2/d \approx \sigma^2$ and

$$\frac{1}{d}\log\rho(\boldsymbol{\theta},\boldsymbol{\theta}') \approx \frac{1}{2}\left(1-\frac{1}{\sigma^2}\right)\left(1-\sigma^2\right).$$

This gives a first-order approximation to the acceptance rate of

$$\min\left(1, \exp\left\{-\frac{d}{2\sigma^2}\left(\sigma^2-1\right)^2\right\}\right),$$

which grows exponentially small with dimension unless $\sigma = 1$. Alternatively, stabilising the acceptance rate above zero requires $\sigma^2 = 1+O(1/\sqrt{d})$; the approximation must become more and more accurate as $d \to \infty$.

The exponential decrease in acceptance rate with dimension is closely linked with the exponential increase in the variance of the weights with dimension in the importance sampling example at the end of Section 1.1.5. In high dimensions, a sufficiently accurate and tractable approximation, q, is rarely available; consequently importance sampling and MHIS are rarely used, except in relatively simple, low-dimensional scenarios.

2.1.3 The Random Walk Metropolis Algorithm

The *random walk Metropolis* (RWM) algorithm was the first MCMC algorithm to ever be used. Unlike the independence sampler, it does not require an accurate global approximation to the posterior and can be tuned so that it works even in very high dimensions. Furthermore, unlike the algorithms that we shall explore subsequently, it does not require the gradient of the log posterior.

The most frequently used RWM proposal, the so-called *preconditioned* RWM, has the form $\boldsymbol{\theta}'|\boldsymbol{\theta} = \boldsymbol{\theta} + \mathsf{N}(\mathbf{0}, \lambda^2\mathbf{V})$, where $\mathbf{V}$ is an estimate of the posterior variance matrix and λ is a tunable scaling parameter. This enhancement can increase the efficiency of the algorithm by many orders of magnitude when the components of $\boldsymbol{\theta}$ are highly correlated and/or vary on very different length scales. However, whatever the proposal, the RWM constraint, that $q(\boldsymbol{\theta}'|\boldsymbol{\theta}) = q(\boldsymbol{\theta}|\boldsymbol{\theta}')$, means that the acceptance probability simplifies to $\min\{1, \pi(\boldsymbol{\theta}')/\pi(\boldsymbol{\theta})\}$.

Scaling of RWM with Dimension

We again consider the isotropic Gaussian posterior of Example 2.1 and show how, when the chain is at equilibrium, the RWM algorithm using a $\mathsf{N}(\boldsymbol{\theta}, \lambda^2 \mathbf{I}_d)$ proposal can be made to work no matter what the dimension.

Write the proposal as $\boldsymbol{\theta}' = \boldsymbol{\theta} + \lambda \mathbf{Z}$, where $\mathbf{Z} \sim \mathsf{N}(\mathbf{0}, \mathbf{I}_d)$. The log acceptance ratio is then

$$\begin{aligned} \log \rho(\boldsymbol{\theta}, \boldsymbol{\theta}') &= -\frac{1}{2}\|\boldsymbol{\theta} + \lambda \mathbf{Z}\|^2 + \frac{1}{2}\|\boldsymbol{\theta}\|^2 = -\lambda \|\boldsymbol{\theta}\| \, \widehat{\boldsymbol{\theta}} \cdot \mathbf{Z} - \frac{1}{2}\lambda^2 \|\mathbf{Z}\|^2 \\ &\stackrel{\mathrm{D}}{=} -\lambda \|\boldsymbol{\theta}\| \, Z' - \frac{1}{2}\lambda^2 \|\mathbf{Z}\|^2, \end{aligned} \tag{2.3}$$

where $Z' \sim \mathsf{N}(0, 1)$ and $\widehat{\boldsymbol{\theta}} = \boldsymbol{\theta}/\|\boldsymbol{\theta}\|$. Now $\|\mathbf{Z}\|^2 \sim \chi^2_d$ and, at equilibrium, $\|\boldsymbol{\theta}\|^2 \sim \chi^2_d$. By the same argument as used for the MHIS, we might make a first approximation of $\|\mathbf{Z}\|^2 \approx d$ and $\|\boldsymbol{\theta}\| \approx \sqrt{d}$, from which it appears that the acceptance ratio must decay exponentially quickly with dimension. However, this need not be the case, since we can control the scaling, λ. The fact that $\|Z\|^2/d \approx 1$ and $\|\boldsymbol{\theta}\|/\sqrt{d} \approx 1$ suggests setting

$$\lambda = \frac{\ell}{\sqrt{d}} \tag{2.4}$$

for some fixed $\ell > 0$. In this case

$$\begin{aligned} \alpha(\boldsymbol{\theta}, \boldsymbol{\theta}') &\stackrel{\mathrm{D}}{=} \min\left[1, \exp\left\{-\ell Z' \, \frac{\|\boldsymbol{\theta}\|}{\sqrt{d}} - \frac{1}{2}\ell^2 \frac{1}{d}\|\mathbf{Z}\|^2\right\}\right] \\ &\approx \min\left[1, \exp\left\{-\ell Z' - \frac{1}{2}\ell^2\right\}\right]. \end{aligned}$$

This quantity is stable away from zero and does not depend on dimension. Taking expectations, elementary calculus gives

$$\mathbb{E}_{\boldsymbol{\theta} \sim \pi, \boldsymbol{\theta}' \sim q(\cdot|\boldsymbol{\theta})}\left[\alpha(\boldsymbol{\theta}, \boldsymbol{\theta}')\right] \approx 2\Phi\left(-\frac{1}{2}\ell\right),$$

where Φ is the cumulative distribution function of a standard normal random variable. This equation describes how, for a high-dimensional Gaussian target, the acceptance rate decreases as the (dimensionally-adjusted) scaling, ℓ, increases.

Indeed, much more is true. Figure 2.3 shows trace plots for θ_1, the first component of $\boldsymbol{\theta}$, when $d = 50$ and when $d = 500$ and using a scaling of $\lambda = \ell/\sqrt{d}$ with $\ell = 2$. The behaviours of the trace plots appear similar, except that when $d = 500$ the time scale over which the process explores the posterior is ten times that when $d = 50$.

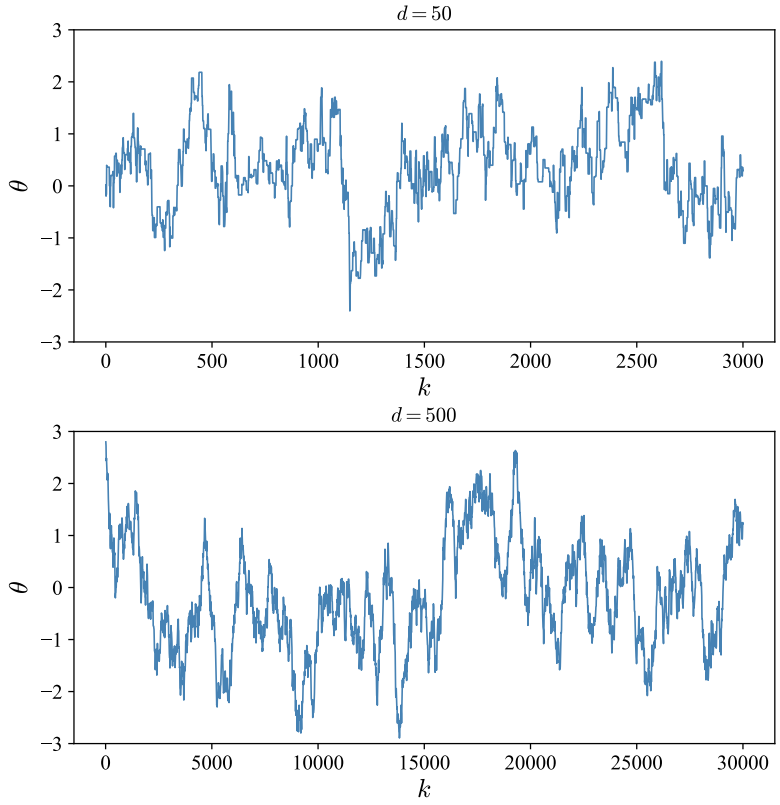

Figure 2.3 Trace plots for the first component, θ_1, of $\boldsymbol{\theta}$ for a RWMH on $\pi(\boldsymbol{\theta}) \propto \exp(-\frac{1}{2}\|\boldsymbol{\theta}\|^2)$ in $d = 50$ (top) and $d = 500$ (bottom). Both algorithms were initialised using a sample from π and each used a scaling of $\lambda = \ell/\sqrt{d}$ with $\ell = 2$.

As dimension goes to infinity, with a scaling of $\ell/\sqrt{d}$ and with time sped up by a factor of d (essentially running for nd iterations rather than n), the path of the first component approaches (in distribution) the path of the stochastic differential equation (2.5), below. For simplicity of notation, we denote the first component by θ and denote its marginal distribution by $f(\theta) \propto \exp(-\theta^2/2)$.

$$\mathrm{d}\theta_t = \frac{1}{2}\left[\log f(\theta_t)\right]' \; h(\ell)\mathrm{d}t + \sqrt{h(\ell)}\mathrm{d}W_t, \tag{2.5}$$

where $h(\ell) = \ell^2 \times 2\Phi(-\ell/2)$. This is the OU process defined in (1.17),

with $b = \sqrt{h(\ell)}$; it has a $\mathsf{N}(0, 1)$ stationary distribution. Here, $h(\ell)$ can be thought of as the *speed* of the diffusion, with a larger value corresponding to a diffusion that will converge to stationarity more quickly, and can be maximised with respect to ℓ, giving $\ell_{\text{opt}} \approx 2.38$. This corresponds to an acceptance rate of $2\Phi(-\ell_{\text{opt}}/2) \approx 0.234$, and leads to the well-known advice to choose the RWM scaling so that the acceptance rate is approximately $1/4$.

Of more importance for us is that the limiting process is approached by letting $\lambda = \ell/\sqrt{d}$ and speeding up time by a factor of d. Reversing this logic, in dimension d, the first component moves d times more slowly than the diffusion. In other words *the time or, equivalently, the number of iterations taken by the RWM to explore the posterior in dimension d is proportional to d*.

Figure 2.4 emphasises this linear dependence on dimension by continuing the example in Figure 2.3. In dimension $d = 50$, the algorithm is run for $n = 10000$ iterations, and for each component, the auto-correlations are calculated up to a lag of 300. The dotted blue line shows the component-wise average of each auto-correlation. For $d = 500$, $n = 100000$ iterations were used and auto-correlations up to a lag of 3000 were calculated. The dashed red line shows the component-wise averages plotted against lag/10. The curves are almost indistinguishable and the resulting estimated integrated auto-correlation times are, respectively, 70 and 718. The corresponding effective sample sizes are, therefore, almost identical, even though the experiment with $d = 500$ used ten times the number of iterations.

The above arguments have been made rigorous and applied to more complex targets such as $\pi(\boldsymbol{\theta}) = \prod_{i=1}^{d} C_i f(C_i\theta_i)$, for a large class of density functions f (see Roberts and Rosenthal, 2001, for example). The limiting process for the first coordinate becomes a Langevin diffusion (1.20) with a stationary density of $C_1 f(C_1\theta_1)$, and in all cases the time taken by the RWM to explore the target is proportional to d.

2.1.4 The Metropolis-Adjusted Langevin Algorithm

The *Metropolis-adjusted Langevin algorithm* (MALA) differs from the RWM proposal of $\boldsymbol{\theta}'|\boldsymbol{\theta} \sim \mathsf{N}(\boldsymbol{\theta}, \lambda^2\mathbf{I}_d)$ through an additional deterministic offset of $\frac{1}{2}\lambda^2\nabla \log \pi(\boldsymbol{\theta})$. We motivate this proposal and then generalise it to allow preconditioning via a positive definite variance matrix, $\mathbf{V}$; as with the RWM, this can bring dramatic efficiency improvements in practice.

The deterministic offset can be seen as an additional movement in the "uphill" direction, that is biasing the proposal to move to areas of higher

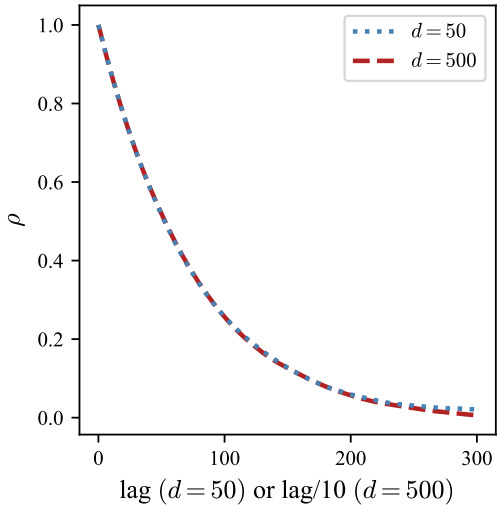

Figure 2.4 Component-wise average auto-correlation plots for a RWM on $\pi(\boldsymbol{\theta}) \propto \exp(-\frac{1}{2}\|\boldsymbol{\theta}\|^2)$ in $d = 50$ and $d = 500$. Both algorithms were initialised using a sample from π and each used a scaling of $\lambda = \ell/\sqrt{d}$ with $\ell = 2$.

posterior density; however, there is a deeper motivation. The proposal can be written as

$$\boldsymbol{\theta}'|\boldsymbol{\theta} = \boldsymbol{\theta} + \frac{1}{2}\lambda^2 \nabla \log \pi(\boldsymbol{\theta}) + \boldsymbol{\epsilon} \tag{2.6}$$

where $\boldsymbol{\epsilon} \sim \mathsf{N}(0, \lambda^2 \mathbf{I}_d)$.

Substituting $\lambda^2 = \delta t$, we see that the proposal is exactly the Euler–Maruyama discretisation of the Langevin diffusion that has a stationary distribution of π, (1.20) (with $b = 1$). In particular, in the hypothetical limit as $\lambda \downarrow 0$ the algorithm should require no accept-reject step to target π. In this sense, it is a natural form for the proposal.

We now derive the preconditioned MALA proposal. For a general positive-definite $\mathbf{V}$, let $\mathbf{A}$ be a square matrix such that $\mathbf{A}\mathbf{A}^\top = \mathbf{V}$, and consider $\boldsymbol{\psi} = \mathbf{A}\boldsymbol{\theta}$. Multiplying (2.6) by $\mathbf{A}$ gives

$$\boldsymbol{\psi}'|\boldsymbol{\psi} = \boldsymbol{\psi} + \frac{1}{2}\lambda^2 \mathbf{A} \nabla_\theta \log \pi(\boldsymbol{\theta}) + \mathbf{A}\boldsymbol{\epsilon},$$

where we have made explicit that the gradient is with respect to $\boldsymbol{\theta}$. The

density for $\boldsymbol{\psi}$ is $\widetilde{\pi}(\boldsymbol{\psi}) \propto \pi(\mathbf{A}^{-1}\boldsymbol{\psi}) = \pi(\boldsymbol{\theta})$. Further, $\nabla_\theta = \mathbf{A}^\top \nabla_\psi$, so

$$\boldsymbol{\psi}'|\boldsymbol{\psi} = \boldsymbol{\psi} + \frac{1}{2}\lambda^2 \mathbf{V}\nabla_\psi \log \widetilde{\pi}(\boldsymbol{\psi}) + \mathbf{A}\boldsymbol{\epsilon}.$$

Since $\mathbf{A}\boldsymbol{\epsilon} \sim \mathsf{N}(\mathbf{0}, \lambda^2\mathbf{V})$, this corresponds to the *preconditioned* MALA proposal:

$$\boldsymbol{\theta}'|\boldsymbol{\theta} \sim \mathsf{N}\left(\boldsymbol{\theta} + \frac{1}{2}\lambda^2 \mathbf{V}\nabla \log \pi(\boldsymbol{\theta}), \lambda^2\mathbf{V}\right).$$

When the posterior is unimodal, preconditioned MALA is often most efficient when $\mathbf{V}$ is an approximation to the posterior variance.

We now explore the scaling of MALA with dimension and the sensitivity to large gradients. In both of these analyses the following simplification of part of the log acceptance ratio will be helpful. For the MALA proposal in (2.6), and writing $\mathbf{g}(\boldsymbol{\theta})$ for the gradient at $\boldsymbol{\theta}$,

$$\begin{aligned}
\log \frac{q(\boldsymbol{\theta}|\boldsymbol{\theta}')}{q(\boldsymbol{\theta}'|\boldsymbol{\theta})} &= \frac{1}{2\lambda^2}\|\boldsymbol{\theta}' - \boldsymbol{\theta} - \frac{\lambda^2}{2}\mathbf{g}(\boldsymbol{\theta})\|^2 - \frac{1}{2\lambda^2}\|\boldsymbol{\theta} - \boldsymbol{\theta}' - \frac{\lambda^2}{2}\mathbf{g}(\boldsymbol{\theta}')\|^2 \\
&= \frac{1}{8}\left[\mathbf{g}(\boldsymbol{\theta}) + \mathbf{g}(\boldsymbol{\theta}')\right]^\top \left[\lambda^2\mathbf{g}(\boldsymbol{\theta}) - \lambda^2\mathbf{g}(\boldsymbol{\theta}') + 4(\boldsymbol{\theta} - \boldsymbol{\theta}')\right] \\
&= -\frac{1}{8}\left[\mathbf{g}(\boldsymbol{\theta}) + \mathbf{g}(\boldsymbol{\theta}')\right]^\top \left[\lambda^2\mathbf{g}(\boldsymbol{\theta}) + \lambda^2\mathbf{g}(\boldsymbol{\theta}') + 4\boldsymbol{\epsilon}\right]. \qquad (2.7)
\end{aligned}$$

Scaling of MALA with Dimension

Consider the isotropic Gaussian running example, Example 2.1, where $\mathbf{g}(\boldsymbol{\theta}) = -\boldsymbol{\theta}$. For the proposal noise, we write $\boldsymbol{\epsilon} = \lambda\mathbf{Z}$, where $\mathbf{Z} \sim \mathsf{N}(\mathbf{0}, \mathbf{I}_d)$. Using (2.7), the log acceptance ratio for MALA is

$$\log \rho(\boldsymbol{\theta}, \boldsymbol{\theta}') = \frac{1}{2}\|\boldsymbol{\theta}\|^2 - \frac{1}{2}\|\boldsymbol{\theta}'\|^2 - \frac{\lambda^2}{8}\|\boldsymbol{\theta} + \boldsymbol{\theta}'\|^2 + \frac{1}{2}\lambda(\boldsymbol{\theta} + \boldsymbol{\theta}')^\top\mathbf{Z}.$$

Substituting $\boldsymbol{\theta}' = (1 - \frac{1}{2}\lambda^2)\boldsymbol{\theta} + \lambda\mathbf{Z}$ from (2.6) and collecting terms, we obtain

$$\log \rho(\boldsymbol{\theta}, \boldsymbol{\theta}') = -\frac{\lambda^3}{8}\left[\lambda\left\{\|\mathbf{Z}\|^2 - \|\boldsymbol{\theta}\|^2\right\} + \frac{1}{4}\lambda^3\|\boldsymbol{\theta}\|^2 + (2 - \lambda^2)\boldsymbol{\theta}^\top\mathbf{Z}\right]. \quad (2.8)$$

Since $\|\mathbf{Z}\|^2$ and $\|\boldsymbol{\theta}\|^2$ are both χ^2_d, each has an expectation of d and their difference is $O_p(d^{1/2})$; further, $\boldsymbol{\theta}^\top\mathbf{Z} \sim \mathsf{N}(0, \|\boldsymbol{\theta}\|^2)$.

To understand the relative sizes of the terms we informally write:

$$\log \rho(\boldsymbol{\theta}, \boldsymbol{\theta}') = \lambda^3\left[\lambda O_p\left(d^{1/2}\right) + \lambda^3 O_p(d) + (2 - \lambda^2)O_p\left(d^{1/2}\right)\right].$$

Thus, with $\lambda = \ell/d^{1/6}$, the first term vanishes and the second and third are $O_p(1)$, leading to an acceptance ratio that is $O_p(1)$.

As for the RWM, it is possible to obtain a limiting diffusion of the form (2.5) for the first component of $\boldsymbol{\theta}$ as the dimension goes to infinity. For MALA, however, the required scaling is $\lambda = \ell/d^{1/6}$, time is sped up by a factor of $d^{1/3}$ (essentially running for $nd^{1/3}$ iterations rather than n) and, for the Gaussian target, the speed of the diffusion is $h(\ell) = 2\ell^2\Phi(-\ell^3/4)$. Optimising the speed with respect to the scaling leads to a recommended acceptance rate of approximately, 57.4%. As with the RWM, the more important point for us is that the limiting OU process mixes in a time of $O(1)$, so the original process, before it has been sped up, mixes in a time of $O(d^{1/3})$. This is considerably faster than the $O(d)$ mixing of the RWM.

As for the RWM, the above result, which is specific to a $\mathsf{N}(\mathbf{0}, \mathbf{I}_d)$ target, has been generalised to targets of the form $\pi(\boldsymbol{\theta}) = \prod_{i=1}^{d} C_i f(C_i\theta_i)$, again leading to a Langevin diffusion for the first component in the limit as $d \to \infty$, an optimal acceptance rate of 57.4%, and requiring time to be sped up by a factor of $d^{1/3}$.

The above product results for MALA rely on the existence and good behaviour of all derivatives of f up to the 7th order, and that the process was started from stationarity. Christensen et al. (2005) investigates the behaviour of MALA on a high-dimensional isotropic Gaussian target when the algorithm is started close to the mode. When a scaling of $\lambda = \ell/d^{1/6}$ is used, in the limit as $d \to \infty$ the process, sped up by a factor of $d^{1/3}$, does not move. Substituting $\boldsymbol{\theta} = 0$, for example, into (2.8), we see that $\log\rho = -\lambda^4\|\mathbf{Z}\|^2/8$. Since $\|\mathbf{Z}\|^2 = O_p(d)$, the acceptance probability is approximately $\exp[-\ell^4 d^{1/3}]$. If, instead, a scaling of $\lambda = \ell/d^{1/4}$ is used then then acceptance rate remains $O_p(1)$ as $d \to \infty$. This new process, sped up by a factor of $d^{1/2}$, moves deterministically towards the region of the main posterior mass. Reductions in efficiency can also occur if only lower derivatives of the target are well-behaved.

Sensitivity to gradients

Whilst the scaling properties of MALA are favourable compared with those of the RWM and MHIS, the performance of MALA is notoriously sensitive to large gradients. We illustrate this with a simple example in one dimension.

Example 2.2 Let $\theta \in \mathbb{R}$ and for some $a > 0$ let $\pi(\theta) \propto \exp\left(-\frac{1}{a}|\theta|^a\right)$, so $\nabla \log\pi(\theta) = -\theta|\theta|^{a-2}$ and $|\nabla \log\pi(\theta)| = |\theta|^{a-1}$.

When $a > 2$, whatever the (fixed) value of λ, for large enough θ, $\mathbb{E}[\theta'|\theta]$ is dominated by the term $\frac{1}{2}\lambda^2\nabla \log\pi(\theta)$, so that (with a very high proba-

bility) the proposal has the opposite sign to the current value and a much larger magnitude, $\lambda^2|\theta|^{a-1}/2$.

Writing $\epsilon = \lambda Z$, where $Z \sim \mathsf{N}(0,1)$, the log acceptance ratio for MALA is $\rho(\theta,\theta') = \log\pi(\theta') - \log\pi(\theta) + B(\theta,\theta')$, where, from (2.7),

$$B(\theta,\theta') = \frac{1}{8}\{\theta|\theta|^{a-2} + \theta'|\theta'|^{a-2}\}\{4\lambda Z - \lambda^2\theta|\theta|^{a-2} - \lambda^2\theta'|\theta'|^{a-2}\}. \quad (2.9)$$

The highest order term in (2.9) arises from the product of θ' terms, so it is negative and of order $|\theta|^{2(a-1)^2}$. The difference in log posteriors is dominated by $\log\pi(\theta') \sim -|\theta|^{a(a-1)}$, which is, again, large and negative. Hence, the acceptance probability is almost certainly very close to 0. Unsurprisingly, since the proposal is even further from the main mass than the current value is, the proposal is almost certainly rejected. As θ moves further and further into the tail of the target, the average (over the proposal distribution) acceptance probability for MALA becomes arbitrarily small and the algorithm converges increasingly slowly.

In Example 2.2, similar poor behaviour occurs even with $a = 2$ (a Gaussian posterior), provided $\lambda^2 > 2$. More generally, MALA can become "stuck" anywhere that $\|\nabla\log\pi\|$ is large. In particular, the basic MALA algorithm should be used with caution if the user suspects that the posterior has tails which are lighter than Gaussian. Mitigations against this behaviour are briefly discussed in the Chapter Notes.

2.2 Hamiltonian Monte Carlo

We have seen that when the dimension d is high, MALA can maintain a high acceptance rate with a scaling of $\ell/d^{1/6}$, whereas the RWM requires a scaling of $\ell/d^{1/2}$. In other words, MALA can propose much larger sensible jumps than the RWM. As we shall see, Hamiltonian Monte Carlo allows even larger jumps than MALA, whilst maintaining a high acceptance rate. *Hamiltonian Monte Carlo* (HMC) can be viewed as using a Metropolis–Hastings algorithm, but with a more intricate proposal mechanism than those seen so far.

One may consider $-\log\pi(\boldsymbol{\theta})$ as a *potential energy* surface. Intuitively one may think of this as a physical surface on which a "particle" with mass M currently sits at a "height" (strictly, potential energy) of $-\log\pi(\boldsymbol{\theta})$ above a current parameter value, $\boldsymbol{\theta} \in \mathbb{R}^d$. To obtain the proposal, the particle is given a random momentum, $\mathbf{p} \in \mathbb{R}^d$ drawn from a symmetric distribution. The true frictionless motion that the particle would undergo along the potential

surface according to Hamiltonian dynamics is approximated numerically. The proposal is the position $\boldsymbol{\theta}' \in \mathbb{R}^d$ after a time T, a tuning parameter.

As we shall see, the log-acceptance ratio for the algorithm can be written as

$$\log \rho(\boldsymbol{\theta}, \boldsymbol{\theta}') = -\log \pi(\boldsymbol{\theta}) + \frac{1}{2M}\mathbf{p}^\top \mathbf{p} - \left\{ -\log \pi(\boldsymbol{\theta}') + \frac{1}{2M}\mathbf{p}'^\top \mathbf{p}' \right\},$$

where $\mathbf{p}'$ is the momentum at time T. The term, $\mathbf{p}^\top \mathbf{p}/(2M)$ is the *kinetic energy* of the particle, and $-\log \pi(\boldsymbol{\theta})$ is the potential energy, so the acceptance rate is $\min[1, \exp(-\delta E)]$, where δE is the change in total energy over time T. Frictionless motion conserves the total energy so that under the exact dynamics the acceptance probability is 1. Numerical integration approximates the dynamics, using an integration step size, ϵ. A smaller ϵ gives a more accurate numerical scheme and a higher average acceptance rate, but for a given T it also requires more numerical steps and, hence, a larger computational cost.

We now provide a more rigorous description of a standard version of the algorithm, including an explanation of the acceptance probability that leads to a stationary distribution of π.

The first component of the algorithm is a positive-definite mass matrix, $\mathbf{M}$, the inverse of which plays a similar role to the preconditioning matrix $\mathbf{V}$ used in the RWM and MALA. The *mass* of an object, as used in the intuitive explanation above, is the ratio between the magnitude of a force that is applied and the magnitude of the acceleration that results and is a scalar. For additional generality, in HMC, we imagine that this ratio can be different along each of a set of d orthogonal principal axes leading to a mass matrix rather than a scalar.

The core component of the HMC algorithm is the numerical integration scheme, which repeatedly uses the leapfrog step to deterministically evolve the position and momentum from a time t to a time $t + \epsilon$: $(\boldsymbol{\theta}_{t+\epsilon}, \mathbf{p}_{t+\epsilon}) = \mathsf{Leap}(\boldsymbol{\theta}_t, \mathbf{p}_t; \epsilon, \mathbf{M})$, where

$$\mathbf{p}_* = \mathbf{p}_t + \frac{1}{2}\nabla \log \pi(\boldsymbol{\theta}_t), \quad \boldsymbol{\theta}_{t+\epsilon} = \boldsymbol{\theta}_t + \epsilon \mathbf{M}^{-1}\mathbf{p}_*, \quad \mathbf{p}_{t+\epsilon} = \mathbf{p}_* + \frac{1}{2}\nabla \log \pi(\boldsymbol{\theta}_{t+\epsilon}).$$

HMC uses the leapfrog scheme rather than, for example, the Euler or Runge–Kutta schemes because the leapfrog scheme possesses two key properties that will be discussed shortly.

HMC repeats the leapfrog step L times, where $L\epsilon = T$, to obtain the proposal, $\boldsymbol{\theta}' = \boldsymbol{\theta}_T$ as depicted in Figure 2.5. The proposed momentum is, in fact, $\mathbf{p}' = -\mathbf{p}_T$ and we denote the transformation: $(\boldsymbol{\theta}, \mathbf{p}) \to (\boldsymbol{\theta}', \mathbf{p}')$ by Leap_-^L.

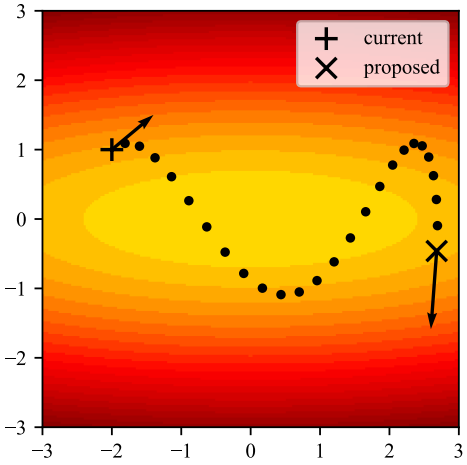

Figure 2.5 Initial point $\boldsymbol{\theta} = \boldsymbol{\theta}_0 = +$ and final point $\boldsymbol{\theta}' = \boldsymbol{\theta}_{2.5} = \times$ after $L = 25$ leapfrog steps using a time interval of $\epsilon = 0.1$ and a mass matrix of $\mathbf{M} = \mathbf{I}_2$. The momentum at the current and proposed point (before the moment flip) is proportional to the size of the corresponding arrow and intermediate points appear as small solid circles.

Standard HMC proposes $\mathbf{p}$ from a $\mathsf{N}(\mathbf{0}, \mathbf{M})$ distribution, and can be viewed as targeting a joint density of $(\boldsymbol{\theta}, \mathbf{p})$ that is the product of the density for $\mathbf{p}$ and the posterior:

$$\widetilde{\pi}(\boldsymbol{\theta}, \mathbf{p}) = \pi(\boldsymbol{\theta})(2\pi)^{-d/2}\mathsf{det}(\mathbf{M})^{-1/2} \exp\Big[-\frac{1}{2}\mathbf{p}^\top \mathbf{M}^{-1}\mathbf{p}\Big]$$

At the end of each iteration, we discard $\mathbf{p}$, and the marginal for $\boldsymbol{\theta}$ is π, as required. Algorithm 2 details the standard version of the Hamiltonian Monte Carlo algorithm.

HMC combines a momentum refresh with a Metropolis–Hastings step which uses a deterministic proposal $(\boldsymbol{\theta}, \mathbf{p}) \leftarrow (\boldsymbol{\theta}', \mathbf{p}') \equiv \mathsf{Leap}^L_-(\boldsymbol{\theta}, \mathbf{p}; \epsilon; \mathbf{M})$; finally the new momentum is discarded. The momentum refreshment preserves $\widetilde{\pi}$ as it samples directly from the correct conditional. We now explain why the accept-reject step with a deterministic proposal also preserves $\widetilde{\pi}$.

The leapfrog step possesses two key properties:

Algorithm 2: Hamiltonian Monte Carlo

Input: Density $\pi(\boldsymbol{\theta})$, initial value $\boldsymbol{\theta}_0$, mass matrix $\mathbf{M}$, time interval T, number of leapfrog steps L.

$\epsilon \leftarrow T/L$.

for $k \in 0, \ldots, n-1$ **do**

Sample $\mathbf{p} \sim \mathsf{N}(\mathbf{0}, \mathbf{M})$.

$(\boldsymbol{\theta}', \mathbf{p}') \leftarrow \mathsf{Leap}_{-}^{L}(\boldsymbol{\theta}_k, \mathbf{p})$.

Calculate the acceptance probability:

$$\alpha(\boldsymbol{\theta}_k, \mathbf{p}; \boldsymbol{\theta}', \mathbf{p}') := \min\left(1, \frac{\widetilde{\pi}(\boldsymbol{\theta}', \mathbf{p}')}{\widetilde{\pi}(\boldsymbol{\theta}_k, \mathbf{p})}\right).$$

With a probability of $\alpha(\boldsymbol{\theta}_k, \mathbf{p}; \boldsymbol{\theta}', \mathbf{p}')$ accept the proposal, $\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}'$; otherwise reject it, $\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k$.

end

Property 1 Leap has a Jacobian of 1.

Property 2 If $\mathsf{Leap}(\boldsymbol{\theta}, \mathbf{p}) = (\boldsymbol{\theta}', \mathbf{p}')$ then $\mathsf{Leap}(\boldsymbol{\theta}', -\mathbf{p}') = (\boldsymbol{\theta}, -\mathbf{p})$.

Property 1 arises because the Leapfrog is a composition of three transformations each of which has a Jacobian of 1. Property 2 is straightforward to verify, and emulates frictionless dynamics in that if after moving for some time the momentum of an object is suddenly reversed, after the same amount of time again the object will end up back where it started, moving with the same speed as when it started but in the opposite direction. The composition of L leapfrog steps possesses the same property: in Figure 2.5, starting at the $\times$ but with a momentum given by the reverse of the corresponding arrow, and proceeding for 25 leapfrog steps leads to the + position, but with a momentum of exactly the reverse of the true initial momentum that the corresponding arrow represents.

Naturally, Leap_{-}^{L}, the composition of L leapfrog steps, combined with a momentum flip, also has a Jacobian of 1. Moreover, Leap_{-}^{L} is self-inverse: $\mathsf{Leap}_{-}^{L}(\mathsf{Leap}_{-}^{L}(\boldsymbol{\theta}, \mathbf{p})) = (\boldsymbol{\theta}, \mathbf{p})$; equivalently, from $(\boldsymbol{\theta}', \mathbf{p}')$ we would propose $(\boldsymbol{\theta}, \mathbf{p})$.

As in Section 2.1, the detailed balance condition is trivial under rejection so we focus on acceptances. Let A be the event of an acceptance and write $(\boldsymbol{\theta}_k, \mathbf{p}_k)$ and $(\boldsymbol{\theta}_{k+1}, \mathbf{p}_{k+1})$ for the position and momentum before and after the acceptance step (and before the momentum is discarded). Then, for $\mathcal{B} \in \mathbb{R}^{2d}$ and $\mathcal{C} \in \mathbb{R}^{2d}$, and writing $\mathsf{Leap}_{-}^{L}(\mathcal{A})$ for the image of a set $\mathcal{A}$

under Leap_-^L,

$$\begin{aligned}
&\mathbb{P}\left(\mathsf{A}, (\boldsymbol{\theta}_k, \mathbf{p}_k) \in \mathcal{B}, (\boldsymbol{\theta}_{k+1}, \mathbf{p}_{k+1}) \in C\right) \\
&\quad = \iint_{\mathcal{B} \cap \mathsf{Leap}_-^L(C)} \widetilde{\pi}(\boldsymbol{\theta}, \mathbf{p}) \alpha(\boldsymbol{\theta}, \mathbf{p}; \boldsymbol{\theta}', \mathbf{p}') \, \mathrm{d}(\boldsymbol{\theta}, \mathbf{p}) \\
&\quad = \iint_{\mathsf{Leap}_-^L(\mathcal{B}) \cap C} \widetilde{\pi}(\boldsymbol{\theta}', \mathbf{p}') \alpha(\boldsymbol{\theta}', \mathbf{p}'; \boldsymbol{\theta}, \mathbf{p})) \, \mathrm{d}(\boldsymbol{\theta}', \mathbf{p}') \\
&\quad = \mathbb{P}\left(\mathsf{A}, (\boldsymbol{\theta}_k, \mathbf{p}_k) \in C, (\boldsymbol{\theta}_{k+1}, \mathbf{p}_{k+1}) \in \mathcal{B}\right),
\end{aligned}$$

where on both intermediate lines we have used that Leap_-^L is self inverse and the penultimate line uses that the Jacobian of $(\boldsymbol{\theta}, \mathbf{p}) \to (\boldsymbol{\theta}', \mathbf{p}')$ is 1.

Scaling of HMC with Dimension

Given a particular integration time, T, the smaller the step size, ϵ, the more accurate the leapfrog scheme, and the closer the acceptance rate is to 1. At the same time, the computational cost is proportional to the number of leapfrog steps, $L = \lceil T/\epsilon \rceil$. A large ϵ leads to many rejections and a small ϵ leads to a high computational cost, suggesting that there is an optimal choice of ϵ between these two extremes.

In this analysis, we consider a general product target, $\pi(\boldsymbol{\theta}) = \prod_{i=1}^d f(\theta_i)$, and assume an identity mass matrix, $\mathbf{M} = \mathbf{I}_d$. In this case, the evolution of each (θ_i, p_i) by Leap_-^L does not depend on any of the other components. The acceptance ratio for HMC is

$$\rho(\boldsymbol{\theta}, \mathbf{p}; \boldsymbol{\theta}', \mathbf{p}') = \frac{\widetilde{\pi}(\boldsymbol{\theta}', \mathbf{p}')}{\widetilde{\pi}(\boldsymbol{\theta}, \mathbf{p})} = \prod_{i=1}^d \rho_1^{(i)}$$

where $(\boldsymbol{\theta}', \mathbf{p}') = \mathsf{Leap}_-^L(\boldsymbol{\theta}, \mathbf{p})$, a deterministic function, and

$$\rho_1^{(i)} = \frac{f(\theta_i')\mathsf{N}(p_i'; 0, 1)}{f(\theta_i)\mathsf{N}(p_i; 0, 1)}.$$

At stationarity, after cancellations, and using the unit Jacobian of Leap_-^L, we have

$$\begin{aligned}
\mathbb{E}_{\theta_i \sim f, p_i \sim \mathsf{N}(0,1)}\left[\rho_1^{(i)}\right] &= \int f(\theta_i')\mathsf{N}(p_i'; 0, 1) \, \mathrm{d}\theta_i \mathrm{d}p_i \\
&= \int f(\theta_i')\mathsf{N}(p_i'; 0, 1) \, \mathrm{d}\theta_i' \mathrm{d}p_i' = 1. \qquad (2.10)
\end{aligned}$$

Moreover, the $\rho_1^{(i)}$ are i.i.d., so, by the central limit theorem, approximately,

$$\log \rho = \sum_{i=1}^d \log \rho_i \sim \mathsf{N}(d\mathbb{E}\left[\log \rho_1\right], d\mathrm{Var}\left[\log \rho_1\right]),$$

from which we see that ρ has approximately a lognormal distribution. From (2.10), and the component-wise independence, $\mathbb{E}[\rho] = 1$, so $\mathbb{E}[\log \rho] = -\frac{1}{2}\text{Var}[\log \rho]$ and, hence, $\mathbb{E}[\log \rho_1] \approx -\frac{1}{2}\text{Var}[\log \rho_1]$. This gives the same distribution for the log-acceptance ratio as we found for the RWM in (2.3) with the same scaling of the expectation and variance with dimension if λ (for the RWM) or ϵ (for HMC) is kept fixed. For the RWM, this necessitated taking $\lambda^2 \propto 1/d$; however, for Hamiltonian dynamics approximated by the leapfrog integrator with step size ϵ over a time period T, the error in the total energy is $O(\epsilon^2)$; *i.e.*, $\mathbb{E}[|\log \rho_1|] = O(\epsilon^2)$. Thus

$$\text{Var}[\log \rho_1] + \frac{1}{4}\text{Var}[\log \rho_1]^2 = \text{Var}[\log \rho_1] + \mathbb{E}[\log \rho_1]^2 = \mathbb{E}\left[(\log \rho_1)^2\right] = O(\epsilon^4).$$

Setting $\epsilon = O(d^{-1/4})$ gives $\text{Var}[\log \rho_1] = O(1/d)$, so both $\mathbb{E}[\log \rho]$ and $\text{Var}[\log \rho]$ are $O(1)$, as required for the acceptance ratio to be well-behaved. Taking $\epsilon \propto d^{-1/4}$, and T fixed as dimension increases, implies that for a given amount of movement in each component, the number of leapfrog steps, and hence the computational cost, increases in proportion to $d^{1/4}$. Contrasting this with a cost of $d^{1/3}$ for MALA and d for the RWM shows why HMC is often the algorithm of choice for high-dimensional targets.

Tuning HMC

After a more rigorous scaling analysis than our heuristic explanation, Beskos et al. (2013) concludes that given T, in the high-dimensional limit, ϵ should be chosen so that the acceptance rate is around 65%; this limit is approached slowly, however, and in practice, it is often found that a higher acceptance rate is optimal.

The main difficulty with tuning HMC is in choosing the integration time, T. For example, for a $\mathsf{N}(0, \sigma^2)$ target, π, using a momentum of $p \sim \mathsf{N}(0, 1)$, it is straightforward to show that if $\theta_0 \sim \pi$ then under the true Hamiltonian dynamics, $\mathsf{Cor}[\theta_0, \theta_T] = \cos(T/\sigma)$. If the target is a product of Gaussians, each with a different length scale, then the auto-correlations between the current values and the proposals for each coordinate have different periods. The periodicity means that increasing T does not monotonically decrease the auto-correlation and the different periods mean that the minimum correlation over all components, which upper bounds the minimum of the lag-1 auto-correlations, is an erratic function of T. Hence, the overall efficiency can, and often does, behave erratically as T is varied. This is illustrated in Figure 2.6 where the optimal choice of T is around 8–9, but slight deviations from this range lead to substantial decreases in efficiency.

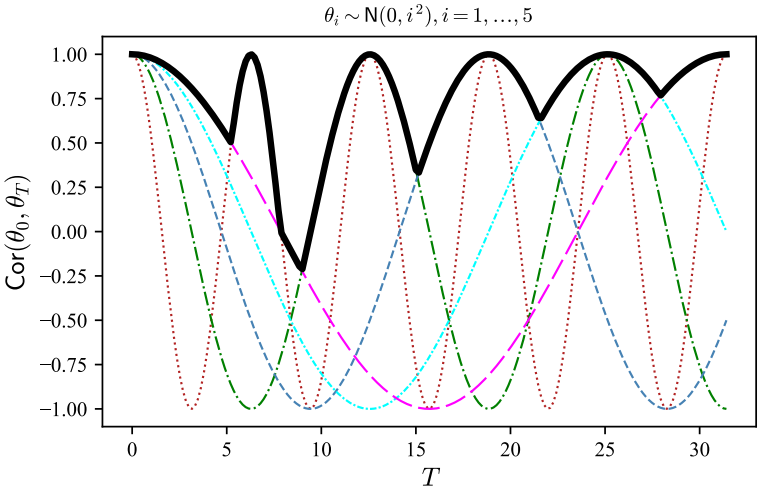

Figure 2.6 $\mathsf{Cor}(\theta_0, \theta_T)$ against T for all 5 components of $\boldsymbol{\theta}$ when $\pi(\boldsymbol{\theta}) \propto \exp[-\frac{1}{2}\sum_{i=1}^{5} \theta_i^2/i^2]$, $\boldsymbol{\theta}_0 \sim \pi$ and $\mathbf{p} \sim \mathsf{N}(\mathbf{0}, \mathbf{I}_5)$ (non-solid lines). The thick solid line is the pointwise maximum over components.

2.3 Chapter Notes

This chapter has only touched the surface on many established aspects of MCMC and variations on the Metropolis–Hastings algorithm of Hastings (1970). The first MCMC algorithm was the random-walk Metropolis-within-Gibbs algorithm of Metropolis et al. (1953), whilst the MALA was suggested and studied in Besag (1994) and Roberts and Tweedie (1996) and HMC was proposed in Duane et al. (1987). There are many texts and review articles devoted to Markov chain Monte Carlo, including Geyer (1992), Robert and Casella (1999), Gamerman and Lopes (2006) and Brooks et al. (2011).

The first high-dimensional RWM scaling result showing a limiting diffusion appears in Roberts et al. (1997) and applies to a product target, $\pi(\boldsymbol{\theta}) \propto \prod_{i=1}^{d} f(\theta_i)$; this is extended to MALA in Roberts and Rosenthal (1998) and, for both the RWM and MALA, to targets of the form $\prod_{i=1}^{d} C_i f(C_i \theta_i)$ in Roberts and Rosenthal (2001). Sherlock and Roberts (2009) tackles the RWM on spherical and elliptical targets, showing that in some situations the optimal acceptance rate can be less than 0.234, and

Sherlock et al. (2015) extends the analysis for product targets to the pseudo-marginal RWM. Of the many other scaling results for these two algorithms, we highlight the following: Christensen et al. (2005) examines the transient phases of the algorithms, Beskos et al. (2009) considers a change of measure from a product law and, finally, Kamatani (2020) considers the RWM on spherically symmetric scale-mixtures of Gaussians and shows that when the tails are heavier than exponential, although the optimal scaling is still ℓ/d, the norm of the process, $\|\boldsymbol{\theta}\|$, mixes in a time of $O(d^2)$ rather than $O(d)$, suggesting that the RWM may be too costly on some, more realistic heavy-tailed targets.

Example 2.2 illustrated the poor behaviour of MALA in the tails of targets with tails that are lighter than Gaussian. A simple solution is to truncate the gradient term (Roberts and Tweedie, 1996). Livingstone and Zanella (2022) provides an alternative use of gradients within the proposal that leads to the same limiting behaviour as MALA, but is automatically robust to issues with light-tails.

A single iteration of HMC approximates Hamiltonian dynamics over a finite time T, whatever the dimension. Thus, if, as in the earlier scaling analysis, T is kept fixed, there can be no limiting diffusion for a product target. A similar scaling analysis to ours appears in Neal (2011), which itself is based on Creutz (1988); a more rigorous analysis is given in Beskos et al. (2013).

Recent works have mitigated against the erratic dependence of the HMC efficiency on the integration time, T. Techniques include introducing randomness into the length of the path (Neal, 2011; Bou-Rabee and Sanz-Serna, 2017; Hoffman et al., 2021), randomising the choice of proposal point from those along the path (Hoffman and Gelman, 2014; Sherlock et al., 2023, the former also automatically choosing T at each iteration and the latter adjusting T according to a natural length scale of the target) or by jittering the momentum after each leapfrog step (Riou-Durand and Vogrinc, 2023).

Further variations on the HMC algorithm include the truly non-reversible Horowitz (1991), which is discussed in more detail in Section 4.3, and Sohl-Dickstein et al. (2014), which tries to mitigate one of the key issues with Horowitz (1991).

A separate strand of methodological developments for reversible MCMC starts with position-dependent preconditioning of MALA and extends to Riemann manifold Hamiltonian Monte Carlo (Girolami and Calderhead, 2011) and Riemann manifold MALA (Xifara et al., 2014) which itself feeds into the Stochastic Gradient Riemannian Langevin Dynamics described in Section 3.4.

3

Stochastic Gradient MCMC Algorithms

Chapter 2 introduced Markov chain Monte Carlo algorithms as a simulation-based approach to approximate distributions of interest. A drawback of the algorithms introduced in Chapter 2 is that their computational time scales poorly with large datasets. In this chapter, we will explore a class of algorithms that can be viewed as approximations of the algorithms introduced in Chapter 2. We introduce the stochastic gradient Langevin algorithm, and extensions of this algorithm, which are popular Bayesian inference methods in the field of machine learning. Compared to traditional MCMC algorithms, we will now replace the gradient of the log density of the target distribution with a stochastic approximation. This stochastic approximation is generated using a subsample of the full dataset to produce an approximate MCMC algorithm. This class of stochastic gradient MCMC algorithms is computationally faster than standard MCMC algorithms but at the expense of introducing a small asymptotic bias that can be corrected post-hoc. Through this chapter, we will discuss the motivation behind these algorithms, and their extensions, and provide empirical comparisons to traditional MCMC algorithms.

3.1 The Unadjusted Langevin Algorithm

Recall that we aim to sample from a posterior distribution with density $\pi(\boldsymbol{\theta})$, where for this chapter, $\boldsymbol{\theta}$ is a $d-$dimensional vector in $\mathbb{R}^d$. It is assumed for the methods we discuss in this chapter that $\log \pi(\boldsymbol{\theta})$ is continuous and differentiable almost everywhere. Simulating a stochastic process that has π as its stationary distribution is a well-established method for generating samples approximately from $\pi(\boldsymbol{\theta})$. By sampling from such a process for an extended period, and discarding the initial burn-in samples, we obtain a set of samples that approximate $\pi(\boldsymbol{\theta})$. The accuracy of the approximation depends on how quickly the stochastic process converges to its stationary distribution from the initial point, relative to the length of the burn-in period,

as well as on the time for the chain to mix within the stationary distribution. The Markov Chain Monte Carlo (MCMC; see Chapter 2) method is the most widely used technique for sampling in this manner

With $b = 1$, the *overdamped Langevin diffusion* first introduced in (1.20) of Section 1.4.3 is

$$\mathrm{d}\boldsymbol{\theta}_t = \frac{1}{2}\nabla \log \pi(\boldsymbol{\theta}_t)\mathrm{d}t + \mathrm{d}W_t, \tag{3.1}$$

where $\frac{1}{2}\nabla \log \pi(\boldsymbol{\theta}_t)$ is the drift term and W_t denotes d-dimensional Brownian motion. In this chapter we sometimes refer to the solution to this stochastic differential equation (SDE) simply as *the Langevin diffusion*. Under mild regularity conditions, the Langevin diffusion has π as its stationary distribution. As detailed in Section 1.4 and, in particular (1.16), this equation can be interpreted as defining the dynamics of a Markov process over infinitesimally small time intervals. That is, for a small time-interval $\delta > 0$, the Langevin diffusion has a discrete-time analogue given by the Euler–Maruyama approximation,

$$\boldsymbol{\theta}_{t+\delta} = \boldsymbol{\theta}_t + \frac{\delta}{2}\nabla \log \pi(\boldsymbol{\theta}_t) + \sqrt{\delta}\mathbf{Z}, \quad t \geq 0 \tag{3.2}$$

where $\mathbf{Z}$ is a vector of d independent standard Gaussian random variables. This discrete-time update equation is commonly known as the *unadjusted Langevin algorithm* (ULA) or the *Langevin Monte Carlo* algorithm. The discrete-time sequence $\{\boldsymbol{\theta}_t\}_{t\geq 0}$ generated by (3.2) differs from the sequence produced by the process in (3.1). The update equation given in (3.2) provides a straightforward and practically implementable method for generating approximate samples from the overdamped Langevin diffusion. To generate samples over a duration $T = n\delta$, where n is an integer, we begin by setting the initial state of the process to $\boldsymbol{\theta}_0$, and then repeatedly simulate the process using (3.2) to obtain values at times $\delta, 2\delta, \ldots, n\delta$. We will use the notation $\boldsymbol{\theta}_k$ to refer to the state of the process at time $k\delta$. As with the MCMC algorithms discussed in Chapter 2, an estimate of any expectation is obtained via a Monte Carlo average: $\mathbb{E}_\pi[h(\boldsymbol{\theta})] \approx \frac{1}{n}\sum_{k=1}^{n} h(\boldsymbol{\theta}_k)$.

To improve the accuracy of the Euler–Maruyama discretisation (3.2) when sampling from the Langevin diffusion at a fixed time T, we can decrease δ. As δ becomes smaller, the discretisation error decreases and the approximation becomes more accurate. In theory, we can achieve any desired degree of accuracy in approximating the SDE (3.1) by selecting δ small enough. However, for a fixed T, the computational cost increases in proportion to $1/\delta$. Alternatively, given a fixed computational budget, T decreases in proportion to δ. The longer T is, the more information

about the diffusion's stationary distribution we collect, and hence, for a fixed computation budget, the variance of any estimate from the samples increases as the bias decreases. In practice, therefore, the choice of δ requires a compromise between the bias and the variance of our estimators.

3.2 Approximate vs. Exact MCMC

The overdamped Langevin diffusion has π as its stationary distribution and therefore it is natural to consider this stochastic process as the basis for an MCMC algorithm. In fact, if it were possible to simulate exactly the dynamics of the Langevin diffusion, then we could use the resulting realisations at a set of discrete time points as our MCMC output. However, for general $\pi(\boldsymbol{\theta})$, the Langevin dynamics are intractable, and therefore it is necessary to resort to using samples generated by the Euler–Maruyama approximation (3.2).

This is most commonly seen with the Metropolis-adjusted Langevin Algorithm (MALA) (see Section 2.1.4). This algorithm uses the Euler–Maruyama approximation (3.2) over an appropriately chosen time-interval, δ, to define the proposal distribution of a standard Metropolis–Hastings sampler (see Algorithm 1). Simulated values are then either accepted or rejected based on the Metropolis–Hastings acceptance probability (2.1). Such an algorithm has good theoretical properties, and in particular, can scale better to high-dimensional problems than the simpler random walk MCMC algorithm. See Section 2.1.4 for a more detailed description of the MALA algorithm and its dimensional scaling.

A simpler algorithm is the just described *unadjusted Langevin algorithm* (3.2), which simulates from the Euler–Maruyama approximation of the Langevin diffusion but does not use a Metropolis–Hastings accept-reject step, and so the stationary distribution of the resulting Markov chain is not π. Hence, even once the Markov chain has essentially converged, the Monte Carlo samples are from an approximation to π rather than from π itself. Because of this, estimators of expectations are typically biased, even as the number of samples, n, grows to infinity. Computationally, such an algorithm is quicker per iteration, but often this saving is small as the cost of calculating $\nabla \log \pi(\boldsymbol{\theta})$, which is required for one step of the ULA algorithm, typically scales at least linearly with the dataset size. If the MALA algorithm is optimally tuned, then approximately 40% of the samples will be rejected, which leads to wasted computation compared to ULA where all samples, albeit biased, are accepted. However, this is counteracted by the larger step sizes that are possible with MALA.

Example: Sampling from a Gaussian Posterior Distribution

To illustrate the computational and statistical accuracy trade-offs between the ULA and MALA schemes, we consider a simple bivariate Gaussian posterior distribution, which we shall use as a running example throughout this chapter. We assume that data arise as realisations from a Gaussian location model with mean parameter $\boldsymbol{\theta}$ assumed to be unknown and the variance $\mathbf{V}$ is known. We select a conjugate Gaussian prior for the unknown $\boldsymbol{\theta}$ which leads to the generative model

$$\mathbf{y}_j|\boldsymbol{\theta} \sim \mathsf{N}(\boldsymbol{\theta}, \mathbf{V}), \qquad \boldsymbol{\theta} \sim \mathsf{N}(\mathbf{0}, \mathbf{I}_2), \quad \text{for } j = 1, \ldots, N, \tag{3.3}$$

where we set $\mathbf{V} = \begin{pmatrix} 1 & 0 \\ 0 & 10 \end{pmatrix}$ and $\mathbf{I}_2$ is a 2-dimensional identity matrix. For this simple model, it is possible to derive a tractable posterior distribution $\boldsymbol{\theta}|\mathbf{y} \sim \mathsf{N}(\boldsymbol{\mu}_N, \boldsymbol{\Sigma}_N)$, where $\boldsymbol{\Sigma}_N = (N\mathbf{V}^{-1} + \mathbf{I}_2)^{-1}$ and $\boldsymbol{\mu}_N = \boldsymbol{\Sigma}_N(\mathbf{V}^{-1}\sum_{j=1}^N \mathbf{y}_j)$. We can use both ULA and MALA schemes to sample from the posterior distribution and compare the Monte Carlo accuracy of both algorithms against the known ground-truth posterior distribution. We measure the distributional accuracy between the true posterior π and a Monte Carlo approximation $\tilde{\pi}$ using the Wasserstein-2 distance,

$$\mathsf{d}^2_{W_2}(\pi, \tilde{\pi}) = \inf_{\zeta \in \Gamma(\pi, \tilde{\pi})} \int_{\mathbb{R}^d \times \mathbb{R}^d} \|\boldsymbol{\theta} - \boldsymbol{\theta}'\|^2 \mathrm{d}\zeta(\boldsymbol{\theta}, \boldsymbol{\theta}'), \tag{3.4}$$

where the inf is taken with respect to all joint distributions ζ which have π and $\hat{\pi}$ as their marginal distributions. In the setting where both π and $\hat{\pi}$ are Gaussian, there is a tractable closed-form expression for the Wasserstein-2 distance,

$$\mathsf{d}^2_{W_2}(\mathsf{N}(\boldsymbol{\mu}_a, \boldsymbol{\Sigma}_a), \mathsf{N}(\boldsymbol{\mu}_b, \boldsymbol{\Sigma}_b)) = \|\boldsymbol{\mu}_a - \boldsymbol{\mu}_b\|_2^2 + \mathrm{trace}(\boldsymbol{\Sigma}_a + \boldsymbol{\Sigma}_b - 2(\boldsymbol{\Sigma}_a^{1/2}\boldsymbol{\Sigma}_b\boldsymbol{\Sigma}_a^{1/2})^{1/2}).$$

In Figure 3.1, we calculated the approximate Wasserstein-2 distance between the true posterior and a moment-matched Gaussian approximation to the Monte Carlo samples generated by the ULA/MALA algorithms. We simulated $N = 1000$ (left panel) and $N = 10000$ (right panel) data points from the model (3.3) and ran ULA/MALA for $n = 1000$ iterations. For each N, MALA and ULA used the same step size, $\delta = 1/N$. Since the true posterior is Gaussian, we expect the moment-matched Gaussian approximation for the MALA sampler to get more and more accurate as the number of iterations increases. Since the ULA algorithm update is a conditional Gaussian, the stationary distribution for ULA is also Gaussian, so the moment-matched Gaussian approximation to this will also get more and more accurate as the number of iterations increases.

Comparing the computational time for ULA and MALA, the per iteration cost of ULA is comparable to MALA initially, if not slightly better. However, with a larger computational budget, i.e. more Monte Carlo iterations, ULA is less accurate due to the asymptotic bias from discretising the Langevin diffusion. When taking into account the reduced computational cost of ULA, this means that ULA is better for small computational budgets, whereas for moderate to large computational budgets, MALA is better. Note that the computational budget required for MALA to display improved statistical efficiency over ULA is dependent on the dataset size. This is highlighted in Figure 3.1, where $N = 1000$ in the left panel and $N = 10000$ in the right panel.

The reason that ULA is competitive with MALA only for very small computational budgets is that the computational gain per iteration is only roughly two-fold (i.e. not calculating the accept-reject ratio roughly halves the cost as the gradient still needs to be calculated), and this is only a small gain relative to the bias that is introduced. If, on the other hand, there was a way of implementing ULA, or something like ULA, which was $O(N)$ faster, then the computational benefit compared to MALA would be more significant. To achieve such a speed-up, this would require an algorithm where the cost of calculating or approximating the gradient is only $O(1)$ – this is the key idea behind the *stochastic gradient Langevin dynamics* algorithm which will be explored in detail in the remainder of this chapter.

3.3 Stochastic Gradient Langevin Dynamics

We have seen how the ULA algorithm is computationally faster than MALA, at the expense of introducing a bias which produces samples not from the desired invariant distribution π, but a distribution close to π. Even without the Metropolis-Hastings acceptance probability, the ULA algorithm still incurs a cost in calculating the gradient of the log-posterior density and under common assumptions, this computational cost scales linearly with the data set size.

Recent interest in Bayesian analysis has considered the challenge of scalable inference in the presence of large datasets, where the log-posterior density is defined as a sum over data points. For instance, if we consider data $\mathbf{y}_1, \ldots, \mathbf{y}_N$ that are conditionally independent given $\boldsymbol{\theta}$, then $\pi(\boldsymbol{\theta}) \propto \pi_0(\boldsymbol{\theta}) \prod_{j=1}^{N} f(\mathbf{y}_j|\boldsymbol{\theta})$. Here, $\pi_0(\boldsymbol{\theta})$ is the prior density, and $f(\mathbf{y}_j|\boldsymbol{\theta})$ is the likelihood for the jth observation. In this context, we define the log-posterior density as

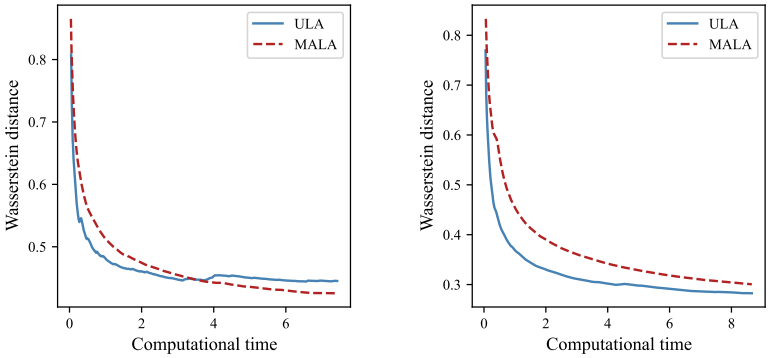

Figure 3.1 Wasserstein distance between the true π and approximate posterior distribution against computational time (in seconds), where the approximate posterior $\tilde{\pi}$ is generated using the ULA and MALA schemes. Left panel $N = 1000$ and right panel $N = 10000$.

$$\log \pi(\boldsymbol{\theta}) = \sum_{j=1}^{N} \log \pi_j(\boldsymbol{\theta}), \quad \text{where} \quad \log \pi_j(\boldsymbol{\theta}) = \log f(\mathbf{y}_j|\boldsymbol{\theta}) + \frac{1}{N} \log \pi_0(\boldsymbol{\theta}). \tag{3.5}$$

The computational bottleneck for ULA is in calculating $\nabla \log \pi(\boldsymbol{\theta})$, which can be expensive if we have a large dataset. For a dataset with N independent observations, where the log-posterior density is a sum over N independent terms (3.5), the computational cost of evaluating the log-posterior density, or its gradient, is $O(N)$ for each iteration of ULA.

A solution to this problem is to use *stochastic gradient Langevin dynamics* (SGLD, Welling and Teh, 2011), which avoids calculating $\nabla \log \pi(\boldsymbol{\theta})$, and instead uses an unbiased estimator of the gradient at each iteration. It is trivial to obtain an unbiased estimate using a random subsample of the terms in the sum. The simplest implementation is to choose $m \ll N$ and estimate $\nabla \log \pi(\boldsymbol{\theta})$ with

$$\nabla^{(m)} \log \pi(\boldsymbol{\theta}) = \frac{N}{m} \sum_{j \in \mathcal{S}_m} \nabla \log \pi_j(\boldsymbol{\theta}), \tag{3.6}$$

where $\mathcal{S}_m$ is a random sample of size m taken without replacement from $\{1, \ldots, N\}$. We call this the *simple* estimator of the gradient and use the

superscript (m) to denote the subsample size used in constructing our estimator. The resulting SGLD algorithm is given in Algorithm 3. Essentially, the SGLD algorithm is the same as ULA (3.2) and is simulating an Euler–Maruyama discretisation of the Langevin diffusion. The only difference is that the true gradient is replaced with the estimated gradient (3.6).

Using an estimator for the gradient adds additional noise, with a variance of $O(\delta^2)$, and therefore the stochastic dynamics no longer follow the ULA update equation (3.2); instead the SGLD algorithm targets a distribution that is close to a tempered version of π. However, for sufficiently small δ, this additional variance becomes negligible compared with the injected Gaussian noise of (3.2), which has a variance of δ. It is possible to generalise Algorithm 3 to the setting of adaptive step sizes δ_k which are dependent on the iteration number k. This is not commonly used in practice and for simplicity, we work with the constant step size version given in Algorithm 3. A justification for using the SGLD algorithm with a decaying step size can be given by an informal argument along the lines that if the step size is $\delta_k \downarrow 0$ for $k \to \infty$, then the process will converge to the true overdamped Langevin diffusion, and hence the Monte Carlo samples will be exact in the limit (see Section 3.3.3).

Algorithm 3: Stochastic Gradient Langevin Dynamics (SGLD)

Input: $\boldsymbol{\theta}_0$, δ.
for $k \in 1, \ldots, n$ **do**
  Draw a subset $\mathcal{S}_m \subset \{1, \ldots, N\}$
  Estimate $\nabla^{(m)} \log \pi(\boldsymbol{\theta})$ using (3.6)
  Draw $\mathbf{Z}_k \sim \mathsf{N}(0, \delta \mathbf{I})$
  Update $\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k + \frac{\delta}{2} \nabla^{(m)} \log \pi(\boldsymbol{\theta}) + \mathbf{Z}_k$
end

The advantage of SGLD is that, if the subsample size m is much smaller than the full dataset size N, the per-iteration cost of the algorithm can be much smaller than that of either MALA or ULA. For large data applications, SGLD has been empirically shown to perform better than standard MCMC when there is a fixed computational budget (Ahn et al., 2015; Li et al., 2016). In challenging examples, performance has been based on measures of predictive accuracy on a held-out test dataset, rather than based on how accurately the samples approximate the true posterior distribution. Furthermore, the conclusions from such studies will clearly depend on the computational budget, with larger budgets favouring exact methods such as

MALA; see the theoretical results in Section 3.3.3 and empirical results in Section 3.2.

The SGLD algorithm is closely related to *stochastic gradient descent* (SGD) (Robbins and Monro, 1951), an efficient algorithm for finding the local maxima of a function. The only difference is the inclusion of the additive Gaussian noise at each iteration of SGLD. Without the noise, but with a suitably decreasing step size, SGD would converge to a local maximum of the density $\pi(\boldsymbol{\theta})$. Again, SGLD has been shown empirically to out-perform stochastic gradient descent (Chen et al., 2014), at least in terms of predictive accuracy – intuitively this is because SGLD will produce samples from the region around the estimate obtained by SGD, and thus can average over the uncertainty in the parameters. This strong link between SGLD and SGD may also explain why the former performs well when compared to exact MCMC methods, at least in terms of predictive accuracy.

3.3.1 Controlling Stochasticity in the Gradient Estimator

The key ingredient of SGLD is found in replacing the true gradient with an unbiased estimator. The more accurate this estimator is, the lower the bias will be for the same computational cost, and thus it is natural to consider alternatives to the simple estimator (3.6). One way of reducing the variance of a Monte Carlo estimator is to use control variates (see Section 1.1.4 for a detailed explanation), which in our setting involves choosing a set of simple functions g_j, $j = 1, \ldots, N$, which we refer to as control variates, and whose sum $\sum_{j=1}^{N} g_j(\boldsymbol{\theta})$ can be evaluated for any $\boldsymbol{\theta}$.

We can rewrite the full-data gradient of the log-posterior density as

$$\sum_{j=1}^{N} \nabla \log \pi_j(\boldsymbol{\theta}) = \sum_{j=1}^{N} g_j(\boldsymbol{\theta}) + \sum_{j=1}^{N} \left(\nabla \log \pi_j(\boldsymbol{\theta}) - g_j(\boldsymbol{\theta}) \right),$$

and from this, we can obtain an unbiased estimator

$$\sum_{j=1}^{N} g_j(\boldsymbol{\theta}) + \frac{N}{m} \sum_{j \in \mathcal{S}_m} (\nabla \log \pi_j(\boldsymbol{\theta}) - g_j(\boldsymbol{\theta})), \tag{3.7}$$

where again $\mathcal{S}_m$ is a random sample of size m drawn from $\{1, \ldots, N\}$. The intuition behind this idea is that if each $g_j(\boldsymbol{\theta}) \approx \nabla \log \pi_j(\boldsymbol{\theta})$, then this estimator can have a much smaller variance than the simple subsampled gradient estimator (3.6).

One approach to choosing the control variate function $g_j(\boldsymbol{\theta})$ that is often used in practice, is to (i) use SGD to find an approximation, $\widehat{\boldsymbol{\theta}}$, to the mode

of the distribution π; and (ii) set $g_j(\boldsymbol{\theta}) = \nabla \log \pi_j(\widehat{\boldsymbol{\theta}})$. This leads to the following control variate estimator,

$$\nabla^{(m)} \log \pi_{\text{cv}}(\boldsymbol{\theta}) = \sum_{j=1}^{N} \nabla \log \pi_j(\widehat{\boldsymbol{\theta}}) + \frac{N}{m} \sum_{j \in \mathcal{S}_m} \left(\nabla \log \pi_j(\boldsymbol{\theta}) - \nabla \log \pi_j(\widehat{\boldsymbol{\theta}}) \right). \tag{3.8}$$

Implementing such an estimator involves an initial up-front cost for finding a suitable $\widehat{\boldsymbol{\theta}}$ and then calculating, storing, and summing $\nabla \log \pi_j(\widehat{\boldsymbol{\theta}})$ for $j = 1, \ldots, N$. For these types of control variate approaches, the main cost is from finding a suitable $\widehat{\boldsymbol{\theta}}$. Although, once found, we can then use $\widehat{\boldsymbol{\theta}}$ as a starting value for the SGLD algorithm, replacing $\boldsymbol{\theta}_0$ with $\widehat{\boldsymbol{\theta}}$ in Algorithm 3, which can significantly reduce the burn-in phase.

The advantage of using the control variate-based estimator can be seen if we compare the variance bounds of this estimator against the simple estimator. If we assume that each $\log \pi_j(\boldsymbol{\theta})$ is twice continuously differentiable on $\mathbb{R}^d$ and has Lipschitz-continuous gradients, then there exist positive constants $L_j > 0$ for all $j = 1, \ldots, N$, such that

$$\|\nabla \log \pi_j(\boldsymbol{\theta}) - \nabla \log \pi_j(\boldsymbol{\theta}')\| \leq L_j \|\boldsymbol{\theta} - \boldsymbol{\theta}'\|. \tag{3.9}$$

Lemma 3.1 and several subsequent results provide bounds on the trace of the variance matrix of each estimator of $\nabla \log \pi(\boldsymbol{\theta})$. Since variances are non-negative, this bounds the variances of each individual component. Also, since all eigenvalues of a variance matrix are non-negative, it bounds the largest of these; *i.e.* the variance of the worst-behaved linear combination of components. Finally, for any d-vector random variable $\boldsymbol{\xi}$ with $\mathbb{E}[\boldsymbol{\xi}] = 0$,

$$\text{tr}(\text{Var}[\boldsymbol{\xi}]) = \mathbb{E}\left[\sum_{i=1}^{d} \xi_i^2 \right] = \mathbb{E}\left[\|\boldsymbol{\xi}\|^2 \right] \geq \text{Var}\left[\|\boldsymbol{\xi}\| \right],$$

so the bounds also apply to the variance of the Euclidean norm of the gradient. For any random vector $\boldsymbol{\xi}$ with $\mathbb{E}[\boldsymbol{\xi}] = 0$, we refer to the important quantity of $\mathbb{E}\left[\|\boldsymbol{\xi}\|^2 \right] = \text{tr}(\text{Var}[\boldsymbol{\xi}])$ as its *pseudo-variance*.

Lemma 3.1 *Assume condition* (3.9)*, then there are constants* $C_1, C_2 > 0$ *where the pseudo variances of the simple gradient estimator* (3.6) *and control variate-based gradient estimator* (3.8) *have the following bounds:*

$$\text{tr}\left(\text{Var}\left[\nabla^{(m)} \log \pi(\boldsymbol{\theta}) \right] \right) \leq C_1 \frac{N^2}{m}, \tag{3.10}$$

$$\text{tr}\left(\text{Var}\left[\nabla^{(m)} \log \pi_{\text{cv}}(\boldsymbol{\theta}) \right] \right) \leq C_2 \|\boldsymbol{\theta} - \widehat{\boldsymbol{\theta}}\|^2 \frac{N^2}{m}, \tag{3.11}$$

Proof We prove this result for the control variate-based gradient estimators (3.8); the result for the simple SGLD estimator (3.6) follows analogously.

We first define $\boldsymbol{\xi} := \nabla^{(m)} \log \pi_{\text{cv}}(\boldsymbol{\theta}) - \nabla \log \pi(\boldsymbol{\theta})$, so that $\boldsymbol{\xi}$ measures the noise in the gradient estimate and has mean zero. The trace of the variance in the noise is then given by

$$\begin{aligned}
\mathbb{E}\left[\|\boldsymbol{\xi}\|^2\right] &= \mathbb{E}\left[\left\|\nabla^{(m)} \log \pi_{\text{cv}}(\boldsymbol{\theta}) - \nabla \log \pi(\boldsymbol{\theta})\right\|^2\right] \\
&= \mathbb{E}\left[\left\|\frac{N}{m}\sum_{j\in\mathcal{S}_m}\left(\nabla \log \pi_j(\boldsymbol{\theta}) - \nabla \log \pi_j(\widehat{\boldsymbol{\theta}})\right) - \left(\nabla \log \pi(\boldsymbol{\theta}) - \nabla \log \pi(\widehat{\boldsymbol{\theta}})\right)\right\|^2\right] \\
&= \mathbb{E}\left[\left\|\frac{N}{m}\sum_{j\in\mathcal{S}_m}\left\{\left(\nabla \log \pi_j(\boldsymbol{\theta}) - \nabla \log \pi_j(\widehat{\boldsymbol{\theta}})\right) - \frac{1}{N}\left(\nabla \log \pi(\boldsymbol{\theta}) - \nabla \log \pi(\widehat{\boldsymbol{\theta}})\right)\right\}\right\|^2\right] \\
&\leq \frac{N^2}{m^2}\mathbb{E}\left[\sum_{j\in\mathcal{S}_m}\left\|\left(\nabla \log \pi_j(\boldsymbol{\theta}) - \nabla \log \pi_j(\widehat{\boldsymbol{\theta}})\right) - \frac{1}{N}\left(\nabla \log \pi(\boldsymbol{\theta}) - \nabla \log \pi(\widehat{\boldsymbol{\theta}})\right)\right\|^2\right].
\end{aligned}$$

where the final line follows from the triangle inequality and due to independence between the $\nabla \log \pi_j(\boldsymbol{\theta})$ terms in the setting of subsampling with replacement. For subsampling without replacement, the sampled indices will be negatively correlated and thus will lead to lower variance. For any random variable $\mathbf{X}$, we have $\mathbb{E}\left[\|\mathbf{X} - \mathbb{E}[\mathbf{X}]\|^2\right] \leq \mathbb{E}\left[\|\mathbf{X}\|^2\right]$. Using this result, and the Lipschitz assumption (3.9), leads to

$$\begin{aligned}
\mathbb{E}\left[\|\boldsymbol{\xi}\|^2\right] &\leq \frac{N^2}{m^2}\sum_{j\in\mathcal{S}_m}\mathbb{E}\left[\left\|\nabla \log \pi_j(\boldsymbol{\theta}) - \nabla \log \pi_j(\widehat{\boldsymbol{\theta}})\right\|^2\right] \\
&\leq \frac{N^2}{m}\mathbb{E}\left[\left\|\nabla \log \pi_j(\boldsymbol{\theta}) - \nabla \log \pi_j(\widehat{\boldsymbol{\theta}})\right\|^2\right] \\
&\leq \frac{N^2}{m}\frac{1}{N}\sum_{j=1}^{N}\left(L_j\left\|\boldsymbol{\theta} - \widehat{\boldsymbol{\theta}}\right\|\right)^2 = \frac{N^2}{m}\left\|\boldsymbol{\theta} - \widehat{\boldsymbol{\theta}}\right\|^2\frac{1}{N}\left(\sum_{j=1}^{N}L_j^2\right),
\end{aligned}$$

where the second line follows from the exchangeability of the indices j and on that line, j, is a single index sampled uniformly at random from $1, \ldots, N$. This proves the stated result and gives $C_2 = \frac{1}{N}\sum_{j=1}^{N} L_j^2$. □

If we make the further assumption that for all $j = 1, \ldots, N$ there exists a positive constant $L > 0$ such that

$$\|\nabla \log \pi_j(\boldsymbol{\theta}) - \nabla \log \pi_j(\boldsymbol{\theta}')\| \leq L\|\boldsymbol{\theta} - \boldsymbol{\theta}'\| \tag{3.12}$$

holds. Then, it is straightforward to show that we have the following Lips-

chitz bound on the gradient of the log-posterior,

$$\|\nabla \log \pi(\boldsymbol{\theta}) - \nabla \log \pi(\boldsymbol{\theta}')\| \leq LN\|\boldsymbol{\theta} - \boldsymbol{\theta}'\|, \tag{3.13}$$

which now leads to an updated constant $C_2 = L^2$ in (3.11) of Lemma 3.1.

By comparing the upper bounds on the variance of the gradients in (3.10) and (3.11), we can see that when $\boldsymbol{\theta}$ is close to $\widehat{\boldsymbol{\theta}}$ we would expect the variance in the control variate estimator to be smaller than for the simple estimator. Furthermore, in many big data settings where N is large, we would expect by the Bernstein–von Mises theorem (e.g. LeCam, 1986) that a value of $\boldsymbol{\theta}$ drawn from the posterior distribution to be of distance $O(N^{-1/2})$ from the mode of the distribution $\widehat{\boldsymbol{\theta}}$, i.e. we expect $\|\boldsymbol{\theta} - \widehat{\boldsymbol{\theta}}\|^2$ to be $O(N^{-1})$. Therefore, compared to the $O(N^2/m)$ variance from the simple estimator (3.10), we would expect to see a reduced $O(N/m)$ variance (3.11) for the control variate gradient estimator. This simple argument suggests that, for the same level of accuracy, we can reduce the computational cost of SGLD by $O(N)$ if we use control variate-based gradient estimators. This is supported by a number of theoretical results which show that if we ignore the pre-processing cost of finding $\widehat{\boldsymbol{\theta}}$, the computational cost per effective sample of SGLD with control variates is $O(1)$, rather than the $O(N)$ cost for SGLD with the simple gradient estimator (3.6).

A further consequence of these bounds on the variance is that they suggest that if $\boldsymbol{\theta}$ is far from $\widehat{\boldsymbol{\theta}}$, then the variance when using control variates can be larger, potentially substantially larger than that of the simple estimator. This point is illustrated in the top-left panel of Figure 3.2, where the variance in the simple gradient estimator (3.6) is approximately constant for all $\boldsymbol{\theta}$. However, the variance in the control variate gradient estimator increases as $\boldsymbol{\theta}$ moves away from $\widehat{\boldsymbol{\theta}}$. Two natural ways of addressing this issue have been proposed in the literature. One option is to only use the control variate estimator when $\boldsymbol{\theta}$ is close enough to $\widehat{\boldsymbol{\theta}}$ (Fearnhead et al., 2018), though it is up to the user to define what is "close enough" in practice. The second approach, a Langevin interpretation of the optimisation algorithm SAGA, which was proposed in Dubey et al. (2016), is to update $\widehat{\boldsymbol{\theta}}$ whilst running the SGLD algorithm. This can be done efficiently by using $g_j(\boldsymbol{\theta}) = \nabla \log \pi_j(\boldsymbol{\theta}_{k_j})$, where $\boldsymbol{\theta}_{k_j}$ is the value of $\boldsymbol{\theta}$ at the most recent iteration of the SGLD algorithm where $\nabla \log \pi_j(\boldsymbol{\theta})$ was evaluated. This involves updating the storage of $g_j(\boldsymbol{\theta})$ and its sum at each iteration; importantly the latter can be done with an $O(m)$ cost.

An alternative approach to reducing the variance, for both the simple and control variate-based gradient estimators, is to use non-uniformly generated subsamples within the gradient estimator. This approach, introduced

in Putcha et al. (2023) and known as *preferential subsampling*, generates subsamples $\mathcal{S}_m \subset \{1, \ldots, N\}$ of size m with replacement, where the probability of drawing the jth data point is p_j and the expected number of times that the jth data point appears in the subsample is mp_j. This then leads to a new simple, as opposed to CV-based, unbiased gradient estimator

$$\nabla^{(m)} \log \pi_{\text{ps}}(\boldsymbol{\theta}) = \frac{1}{m} \sum_{j \in \mathcal{S}_m} \frac{\nabla \log \pi_j(\boldsymbol{\theta})}{p_j}, \tag{3.14}$$

where $p_j > 0$ for all $j = 1, \ldots, N$ and $\sum_{j=1}^{N} p_j = 1$. The simple gradient estimator (3.6) is given as a special case when $p_j = 1/N$ for all j.

If we define the error in the stochastic gradient as $\boldsymbol{\xi} = \nabla^{(m)} \log \pi_{\text{ps}}(\boldsymbol{\theta}) - \nabla \log \pi(\boldsymbol{\theta})$, then taking expectations over the random subsample, whose distribution depends on the weights $\mathbf{p} = (p_1, \ldots, p_N)$, leads to the pseudo-variance of $\nabla^{(m)} \log \pi_{\text{ps}}(\boldsymbol{\theta})$,

$$\mathbb{E}\left[\|\boldsymbol{\xi}\|^2\right] = \text{tr}\left(\text{Var}[\nabla^{(m)} \log \pi_{\text{ps}}(\boldsymbol{\theta})]\right).$$

Lemma 3.2 *The optimal weights* $\mathbf{p}^*$ *which minimise the pseudo-variance*

$$\min_{\mathbf{p}: p_j \in [0,1], \sum_j p_j = 1} \text{tr}\left(\text{Var}[\nabla^{(m)} \log \pi_{\text{ps}}(\boldsymbol{\theta})]\right)$$

are equivalently found by minimising

$$\min_{\mathbf{p}} \sum_{j=1}^{N} \frac{1}{p_j} \left\|\nabla \log \pi_j(\boldsymbol{\theta})\right\|^2,$$

resulting in optimal weights of the form

$$p_j^* = \frac{\|\nabla \log \pi_j(\boldsymbol{\theta})\|}{\sum_{i=1}^{N} \|\nabla \log \pi_i(\boldsymbol{\theta})\|} \text{ for } j = 1, \ldots, N. \tag{3.15}$$

Proof A full proof of this result is given in Putcha et al. (2023). In brief, this result follows from two steps. Firstly, a straightforward expansion of $\text{tr}\left(\text{Var}[\nabla^{(m)} \log \pi_{\text{ps}}(\boldsymbol{\theta})]\right)$ with the gradient estimator given in (3.14) leads to $\frac{1}{m} \sum_{j=1}^{N} \frac{1}{p_j} \left\|\nabla \log \pi_j(\boldsymbol{\theta})\right\|^2 + C$, where $C > 0$ is a constant that is independent of $\mathbf{p}$. Minimising this term with respect to the constraint $\{\mathbf{p} : p_j \in [0, 1], \sum_j p_j = 1\}$ follows by using Lagrange multipliers.

□

In practice, the optimal weights p_j^*, which would minimise the gradient variance, are dependent on the current iterate $\boldsymbol{\theta}_k$ of the SGLD algorithm. This means that for each iteration k in Algorithm 3, every p_j for all $j =$

$1, \ldots, N$ would need to be updated, which is an $O(N)$ calculation and would defeat the purpose of using subsamples of size $m \ll N$ in the SGLD algorithm. We could instead replace the optimal weights p_j^* (3.15) with approximate weights $\widehat{p}_j$, where in (3.15) we replace $\boldsymbol{\theta}$ with an estimate of the posterior mode $\widehat{\boldsymbol{\theta}}$. As discussed in the case of control variate gradient estimators, finding $\widehat{\boldsymbol{\theta}}$ requires a one-off pre-processing cost.

Weighted sampling can be combined with the control variate estimator (3.7) with a natural choice of weights that are increasing with the size of the derivative of $\nabla \log \pi_j(\boldsymbol{\theta})$ at $\widehat{\boldsymbol{\theta}}$. We can also use stratified sampling to try to ensure each subsample is representative of the full data. However, regardless of the choice of gradient estimator, an important question is: how large should the subsample size m be? Taking one iteration of SGLD, the variance of the noise from the gradient term is dominated by the variance of the injected noise. As the former is proportional to δ^2, and the latter to δ, m will be $O(1)$ as N increases if we choose $\delta = O(1/N)$ (the square of the target size). The choice of step size is discussed further in Section 3.5.

Subsample size could also be dynamically adjusted whilst running the SGLD algorithm. The idea, which is particularly relevant when using control variates, is that the accuracy of the estimator of the gradient can vary considerably with $\boldsymbol{\theta}$. To counteract this, it may be more efficient to have a larger subsample size when the variance would be larger. One simple approach is to specify an upper bound, say V, on the variance that we would like to achieve, and consider how to vary subsample size to achieve this.

An extension of the result in Lemma 3.1 can be derived for weighted gradients if the control variate gradient estimator (3.7) in Lemma 3.1 is replaced with the preferential sampling gradient estimator (3.14). This results in a new upper bound for the variance of the gradient estimator,

$$\operatorname{tr}\left(\operatorname{Var}\left[\nabla^{(m)} \log \pi_{\mathrm{cv}}(\boldsymbol{\theta})\right]\right) \leq \frac{1}{m}\|\boldsymbol{\theta} - \widehat{\boldsymbol{\theta}}\|^2 \sum_{i=1}^{N} \frac{L_j^2}{p_j}, \tag{3.16}$$

where p_j are subsample weights (3.15) and L_j are Lipschitz constants on the gradient components (3.9). A similar result can be derived for the simple gradient estimator (3.6). As with the control variate bound in Lemma 3.1, the bound in (3.16) is also $O(N)$. The $O(1/N)$ term $\|\boldsymbol{\theta} - \widehat{\boldsymbol{\theta}}\|^2$ cancels with the summation over N terms. However, each of the N terms is $O(N)$ since each p_j is $O(1/N)$. Choosing appropriate weights p_j for each $j = 1, \ldots, N$ reduces the multiplier of N.

Given our specified upper bound $V > 0$, we can plug this into (3.16). By rearranging the inequality and using the optimal weights p_j^* (3.15), we can

show that the subsample size should be at least

$$m > \frac{1}{V}\|\boldsymbol{\theta} - \widehat{\boldsymbol{\theta}}\|^2 \left(\sum_{i=1}^{N} \frac{L_j^2}{p_j^*} \right).$$

For a fixed bound V, fixed N and fixed weights p_j^*, the optimal subsample size is $m \propto \|\boldsymbol{\theta} - \widehat{\boldsymbol{\theta}}\|^2$, which suggests that for SGLD with control variates, the subsample size should increase at a rate which is quadratic in the distance between the current iterate of the SGLD chain $\boldsymbol{\theta}_k$ and the mode of the posterior distribution $\widehat{\boldsymbol{\theta}}$. Whilst the constant of proportionality may be hard to calculate, a user can choose a constant based on a reasonable average subsample size they want to achieve – and this would still enforce that we have similar accuracy for the estimate of the gradient for all iterations of SGLD.

3.3.2 Example: The Value of Control Variates

Recall the tractable Gaussian posterior example $\pi(\boldsymbol{\theta}) = \mathsf{N}(\boldsymbol{\mu}_N, \boldsymbol{\Sigma}_N)$ in (3.3), where $\boldsymbol{\Sigma}_N = (N\mathbf{V}^{-1} + \mathbf{I}_2)^{-1}$ and $\boldsymbol{\mu}_N = \boldsymbol{\Sigma}_N(\mathbf{V}^{-1}\sum_{j=1}^{N}\mathbf{y}_j)$. From this posterior distribution, we can calculate the full-data gradient, as well as the simple stochastic (3.6) and control variate-based gradients (3.8). The gradient for the jth component is $\nabla \log \pi_j(\boldsymbol{\theta}) = \nabla \log f(\mathbf{y}_j|\boldsymbol{\theta}) + (1/N)\nabla \log \pi_0(\boldsymbol{\theta})$ and for this example we can easily derive the posterior mode $\widehat{\boldsymbol{\theta}} = \boldsymbol{\Sigma}_N(\mathbf{V}^{-1}\sum_{j=1}^{N}\mathbf{y}_j)$ and use this within the control variate gradient estimator. For the full-data, simple stochastic, and control variate gradient estimators we have the following,

$$\nabla \log \pi(\boldsymbol{\theta}) = \sum_{j=1}^{N} \nabla \log \pi_j(\boldsymbol{\theta}) = -\sum_{j=1}^{N} \mathbf{V}^{-1}\mathbf{y}_j - (N\mathbf{V}^{-1} + \mathbf{I}_2)\boldsymbol{\theta},$$

$$\nabla^{(m)} \log \pi(\boldsymbol{\theta}) = \frac{N}{m} \sum_{j \in \mathcal{S}_m} \nabla \log \pi_j(\boldsymbol{\theta}) = -\frac{N}{m} \sum_{j \in \mathcal{S}_m} \mathbf{V}^{-1}\mathbf{y}_j - \left(N\mathbf{V}^{-1} + \mathbf{I}_2\right)\boldsymbol{\theta},$$

$$\begin{aligned}\nabla^{(m)} \log \pi_{\text{cv}}(\boldsymbol{\theta}) &= \sum_{j=1}^{N} \nabla \log \pi_j(\widehat{\boldsymbol{\theta}}) + \frac{N}{m} \sum_{j \in \mathcal{S}_m} \left(\nabla \log \pi_j(\boldsymbol{\theta}) - \nabla \log \pi_j(\widehat{\boldsymbol{\theta}}) \right) \\ &= -\sum_{j=1}^{N} \mathbf{V}^{-1}\mathbf{y}_j - (N\mathbf{V}^{-1} + \mathbf{I}_2)\widehat{\boldsymbol{\theta}} + \left(N\mathbf{V}^{-1} + \mathbf{I}_2\right) \left(\widehat{\boldsymbol{\theta}} - \boldsymbol{\theta} \right) \\ &= -\sum_{j=1}^{N} \mathbf{V}^{-1}\mathbf{y}_j - (N\mathbf{V}^{-1} + \mathbf{I}_2)\boldsymbol{\theta}.\end{aligned}$$

For this Gaussian example, with the control variate estimator set to the posterior mode, the control variate-based gradient results in the same gradient estimator as the full-data gradient. The simple gradient estimator (3.6) gives an unbiased estimate of the full-data gradient, however, for small subsample sizes, this estimator can lead to poor posterior approximations. For this particular model, it is possible to separate the data $\mathbf{y}_j$ from the parameters $\boldsymbol{\theta}$ in such a way that the gradient estimator can be updated at each Monte Carlo iteration without re-evaluating the data, i.e. the data component of the gradient could be pre-computed and stored. Therefore, for this particular model, the simple SGLD gradient estimator would not be recommended as the model structure easily leads to more efficient gradient estimators. However, deriving better gradient estimators, such as for this Gaussian posterior model, is usually only possible for simple models and therefore the simple unbiased gradient estimator (3.6) is still a popular choice within the SGLD scheme (Algorithm 3) for general models.

Figure 3.2 shows the posterior approximation for the Gaussian model using the three gradient estimators, where stochastic gradients are calculated using 1% of the full dataset. The ULA sampler (top-right) without a Metropolis–Hastings correction produces an accurate, albeit slightly biased, approximation similar to the SGLD with control variates (SGLD-CV) algorithm (bottom-right), which has the same gradient estimator for this model. The SGLD algorithm (bottom-left) has the correct posterior mean but produces an over-dispersed approximation to the posterior variance. Reducing the stochasticity in the gradient estimator, for example, by increasing the subsample size, will lead to improved empirical performance. This is a well-known feature of the SGLD algorithm and its theoretical properties are discussed in the next section.

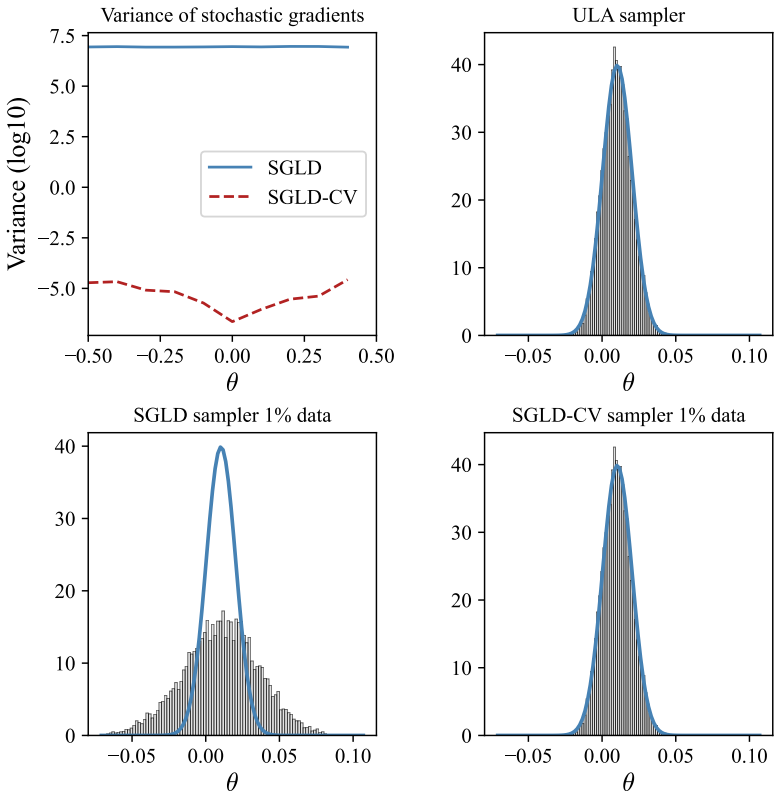

Figure 3.2 Top left: Variance of the estimated gradients taken over a range of $\theta^{(1)}$ for the first component of $\boldsymbol{\theta}$. The variance for the SGLD estimator is stable across $\theta^{(1)}$, whereas the variance in the SGLD-CV gradient estimator increases as $|\theta^{(1)} - \widetilde{\theta}^{(1)}|$ increases. The remaining plots give the posterior approximation to the first marginal component $\theta^{(1)}$ for ULA, SGLD and SGLD-CV, with the solid black line representing the true density.

3.3.3 Convergence Results for Stochastic Gradient Langevin Dynamics

The SGLD algorithm provides a simple and efficient approach for sampling from a posterior distribution π. However, a key question is whether errors can accumulate over many SGLD iterations, leading to poor approximate samples. Fortunately, under suitable regularity conditions on π, theoretical results indicate that SGLD can avoid persistent error accumulation. A key

assumption is that the drift in the underlying Langevin diffusion pushes the state $\boldsymbol{\theta}$ toward regions of high probability under π. This ensures that the diffusion is geometrically ergodic - i.e. it forgets its initial position at an exponential rate. As a result, the SGLD-generated samples tend to remain in the regions of high probability under π.

There are two main theoretical approaches for analyzing the accuracy of SGLD. (i) We can consider the accuracy of estimating expectations of the form $\mathbb{E}_\pi\left[h(\boldsymbol{\theta})\right]$, for some test function $h(\boldsymbol{\theta})$ under the posterior distribution π. This involves taking the average of $h(\boldsymbol{\theta}_k)$ over n SGLD iterates $\{\boldsymbol{\theta}_k\}_{k=1}^n$. The mean squared error (MSE) between this Monte Carlo average and the true expectation $\mathbb{E}_\pi\left[h(\boldsymbol{\theta})\right]$ provides one measure of SGLD accuracy. Teh et al. (2016) studied this error metric in the setting where the SGLD step size δ_k decays over iterations. (ii) Alternatively, we can bound the error between the distribution of the SGLD iterates and the posterior π after n steps. This involves analysing the total variation, or Wasserstein distance, between the marginal distribution of the SGLD chain at iteration n (i.e. $\tilde{\pi}_n$) and π. Since SGLD is based on the Langevin diffusion, its ergodicity properties can be leveraged to prove the marginal distributions converge to π.

Consider approach (i). The MSE of the SGLD estimator can be upper bounded by $C_N(\delta^2 + 1/n\delta)$, where C_N is a constant dependent on the dataset size N. The δ^2 term reflects the squared bias and $1/n\delta$ is the variance term. For a fixed computational budget n, the bias increases if the step size δ increases, while the variance decreases with increasing δ. Teh et al. (2016) showed that in the setting of a decreasing step size δ_k, in order to minimise the asymptotic MSE, the optimal δ decays at rate of $n^{-1/3}$. This yields an MSE rate of $n^{-2/3}$, which is slower than the n^{-1} rate for standard Monte Carlo methods. The slower convergence arises from controlling both bias and variance, which is similar to other asymptotically biased Monte Carlo methods. On the other hand, for larger computational budgets (i.e. larger n), exact MCMC will outperform SGLD, because unlike for exact MCMC methods, the bias from SGLD is non-vanishing.

For approach (ii), one quantifies the accuracy of SGLD via the accuracy of the marginal distribution of the SGLD samples, $\boldsymbol{\theta}_k$, at iteration k. We denote this marginal distribution by $\tilde{\pi}_k$. Accuracy is commonly measured by the Wasserstein distance (3.4) between $\tilde{\pi}_k$ and the posterior distribution π, as this makes the analysis more tractable. However, care is needed when interpreting the Wasserstein distance as it is not scale-invariant, i.e. changing the units scales the distance. Additionally, for a fixed level of accuracy in each marginal, the distance grows as $d^{1/2}$ with dimension d.

Much of the theory for ULA and SGLD assumes the log posterior density,

$\log \pi(\boldsymbol{\theta})$, is smooth and strongly concave. The key assumptions for analysis of these algorithms are that $\log \pi(\boldsymbol{\theta})$ is l-convex (3.18) and $\log \pi(\boldsymbol{\theta})$ is continuously differentiable with L-Lipschitz gradients (3.17). This means there exist constants $0 < l \leq L$ such that for all $\boldsymbol{\theta}$ and $\boldsymbol{\theta}'$:

$$\|\nabla \log \pi(\boldsymbol{\theta}') - \nabla \log \pi(\boldsymbol{\theta})\| \leq L\|\boldsymbol{\theta} - \boldsymbol{\theta}'\|, \quad \text{and} \tag{3.17}$$

$$-\langle \nabla \log \pi(\boldsymbol{\theta}) - \nabla \log \pi(\boldsymbol{\theta}'), \boldsymbol{\theta} - \boldsymbol{\theta}' \rangle \geq \frac{l}{2}\|\boldsymbol{\theta} - \boldsymbol{\theta}'\|^2, \tag{3.18}$$

where we define $\kappa = L/l$ as the condition number. The first condition (3.17) bounds how fast the Langevin drift can change, and thus controls the step size (which should be $\delta < 1/L$). The second assumption (3.18) ensures that the drift term pushes $\boldsymbol{\theta}$ towards high-density regions, making the Langevin diffusion geometrically ergodic. Together, these assumptions imply an upper and lower bound on the directional derivatives of the log posterior density. The bounds enable stable discretisation and prevent persistent error accumulation in the SGLD algorithm. They also imply $\log \pi(\boldsymbol{\theta})$ is unimodal. By leveraging strong concavity, the resulting theory provides step size conditions and rates of convergence for ULA/SGLD. When $\log \pi(\boldsymbol{\theta})$ is non-strongly concave, alternative assumptions are required (Raginsky et al., 2017; Majka et al., 2020), for instance that $\log \pi(\boldsymbol{\theta})$ is dissipative, i.e. $\langle \boldsymbol{\theta}, \nabla \log \pi(\boldsymbol{\theta}) \rangle \geq a\|\boldsymbol{\theta}\|^2 - b$, for some $a > 0$ and $b \geq 0$.

Under the above assumptions (3.17)-(3.18), Dalalyan and Karagulyan (2019) showed that running the SGLD algorithm with a step size parameter $\delta \leq 2/(l + L)$, the Wasserstein-2 distance $\mathsf{d}_{W_2}(\tilde{\pi}_n, \pi)$ between the SGLD marginal $\tilde{\pi}_n$ at iteration n and the posterior π can be bounded as:

$$\mathsf{d}_{W_2}(\tilde{\pi}_n, \pi) \leq (1 - l\delta)^n \mathsf{d}_{W_2}(\tilde{\pi}_0, \pi) + C_1(\delta d)^{1/2} + C_2\sigma(\delta d)^{1/2}, \tag{3.19}$$

where l, C_1, C_2 are constants, d is the dimension, and σ^2 is an upper bound on the variance of any individual component of the stochastic gradient (3.10). In the case of ULA, $\sigma^2 = 0$. The first term in the upper bound decays exponentially, controlling bias from the initial distribution of the algorithm $\tilde{\pi}_0$ where the Markov chain is initialised from. The second term represents the error that results from the Euler–Maruyama discretisation of the Langevin diffusion. The final term relates to the variance in the stochastic gradient. In the case of the simple gradient estimator (3.6) the variance is $O(N^2/m)$ and the final term in the bound is $O(N\sqrt{(\delta d/m)})$, which is then the dominating term in the upper bound.

Given that the main motivation of SGLD is to perform Bayesian inference over large-scale data, a natural question is to ask how does SGLD scale as data size N increases? One way of addressing this is to ask, as N increases,

what is the computational cost of running SGLD so that we have a fixed level of approximation? We need to define our measure of approximation appropriately to account for the fact that π will change as N increases: under certain assumptions, such as the Bernstein–von Mises (LeCam, 1986) assumption, the variance will decay as $1/N$. This has been investigated by Nagapetyan et al. (2017) and Baker et al. (2019) who consider using control variates as a pre-processing step, which has a computational cost that is linear in N. Ignoring the cost of this pre-processing step for SGLD, using control variates asymptotically has a computational cost per effective sample that is constant. By comparison, the computational cost per effective sample of SGLD with the simple estimator of the gradient (3.6) is linear in N.

Example: Theoretical Properties on a Gaussian Target Distribution

We can gain insight into the properties of the SGLD algorithm by returning to our running Gaussian example (3.3). Recall that the posterior under this model is a bivariate Gaussian distribution $\boldsymbol{\theta}|\mathbf{y} \sim \mathsf{N}(\boldsymbol{\mu}_N, \boldsymbol{\Sigma}_N)$. In (3.3), we assumed that the covariance matrix $\boldsymbol{\Sigma}_N$ was diagonal, however, we will now instead consider a general symmetric positive semi-definite matrix. We can express the variance matrix in terms of some rotation matrix $\mathbf{P}$ and a diagonal matrix $\mathbf{D}$, whose entries satisfy the condition $\sigma_1^2 \geq \sigma_2^2$, i.e. $\boldsymbol{\Sigma}_N = \mathbf{P}^\top \mathbf{D} \mathbf{P}$. Under this Gaussian posterior model, the drift term of the Langevin diffusion is

$$\nabla \log \pi(\boldsymbol{\theta}) = -\boldsymbol{\Sigma}_N^{-1}(\boldsymbol{\theta} - \boldsymbol{\mu}_N) = -\mathbf{P}^\top \mathbf{D}^{-1} \mathbf{P}(\boldsymbol{\theta} - \boldsymbol{\mu}_N).$$

The $k+1$th iteration of the SGLD algorithm is

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \frac{\delta}{2} \mathbf{P}^\top \mathbf{D}^{-1} \mathbf{P}(\boldsymbol{\theta}_k - \boldsymbol{\mu}_N) + \delta \boldsymbol{\nu}_k + \sqrt{\delta} \mathbf{Z}_k, \tag{3.20}$$

where $\mathbf{Z}_k$ is a vector of two independent standard Gaussian random variables and $\boldsymbol{\nu}_k$ is the error induced by our stochastic estimate of $\nabla \log \pi(\boldsymbol{\theta}_k)$. The entries of $\mathbf{D}^{-1}$ correspond to the constants that appear in assumptions (3.17)-(3.18), with $l = 1/\sigma_1^2$ and $L = 1/\sigma_2^2$, which are discussed further in Section 3.5 in terms of their impact on the step size parameter δ.

We can simplify the SGLD algorithm by considering updates on the transformed state $\tilde{\boldsymbol{\theta}} = \mathbf{P}(\boldsymbol{\theta} - \boldsymbol{\mu}_N)$, which gives SGLD updates,

$$\begin{aligned}
\tilde{\boldsymbol{\theta}}_{k+1} =& \tilde{\boldsymbol{\theta}}_k - \frac{\delta}{2} \mathbf{D}^{-1} \tilde{\boldsymbol{\theta}}_k + \delta \mathbf{P} \boldsymbol{\nu} + \sqrt{\delta} \mathbf{P} \mathbf{Z} \\
=& \begin{pmatrix} 1 - \delta/(2\sigma_1^2) & 0 \\ 0 & 1 - \delta/(2\sigma_2^2) \end{pmatrix} \tilde{\boldsymbol{\theta}}_k + \delta \mathbf{P} \boldsymbol{\nu}_k + \sqrt{\delta} \mathbf{P} \mathbf{Z}_k,
\end{aligned}$$

where the variance of $\mathbf{PZ}_k$ is still the identity as $\mathbf{P}$ is a rotation matrix.

The SGLD update is a vector auto-regressive process. This process will have a stationary distribution if $\delta < 4\sigma_2^2 = 4/L$, otherwise the process will produce trajectories which will tend to infinity in at least one of the two components. A similar assumption on the step size is required to establish the upper bound on $\mathsf{d}_{W_2}(\tilde{\pi}_n, \pi)$ in (3.19) for log-concave posterior distributions.

For simplicity in the manner of convergence, we choose $\delta < 2\sigma_2^2$ and then define $\lambda_j = \delta/(2\sigma_j^2) < 1$. This then leads to the following SGLD dynamics for each component, $j = 1, 2$

$$\tilde{\theta}_{k+1}^{(j)} = (1-\lambda_j)^k \tilde{\theta}_0^{(j)} + \sum_{i=1}^{k} (1-\lambda_j)^{k-i} \left(\delta \mathbf{P}\boldsymbol{\nu}_i + \sqrt{\delta}\mathbf{P}\mathbf{Z}_i\right)^{(j)}, \tag{3.21}$$

where $\tilde{\theta}_k^{(j)}$ is the jth component of $\tilde{\boldsymbol{\theta}}_k$. From this, we immediately see from the first term on the right-hand side of (3.21) that the SGLD algorithm forgets its initial condition exponentially quickly. However, the rate of exponential decay is slower for the component with the larger marginal variance, i.e. σ_1^2. Furthermore, as the step size δ is constrained by the smaller marginal variance σ_2^2, this rate will have to be slow if $\sigma_2^2 \ll \sigma_1^2$; this suggests that there are benefits from re-scaling the posterior so that the marginal variances of different components are approximately equal.

Taking the expectation of (3.21), with respect to $\boldsymbol{\nu}$ and $\mathbf{Z}$, and letting $k \to \infty$, results in SGLD dynamics that have the correct limiting mean, but with an inflated variance. This is most easily seen if we assume that the variance of $\mathbf{P}\boldsymbol{\nu}$ is independent of position. In this case, the stationary distribution of SGLD will have a variance of

$$\mathrm{Var}_{\tilde{\pi}}\left[\tilde{\boldsymbol{\theta}}\right] = \begin{pmatrix} (1-(1-\lambda_1)^2)^{-1} & 0 \\ 0 & (1-(1-\lambda_2)^2)^{-1} \end{pmatrix} (\delta^2 \boldsymbol{\Sigma}_N + \delta \mathbf{I}_2),$$

where $\mathbf{I}_2$ is a 2-dimensional identity matrix. The marginal variance for component j is thus

$$\sigma_j^2 \frac{1+\delta\Sigma_{jj}}{1-\delta/(4\sigma_j^2)} = \sigma_j^2\left(1+\delta\Sigma_{jj}\right) + \frac{\delta}{4} + O(\delta^2).$$

The inflation in variance comes both from the noise in the estimate of $\nabla \log \pi(\boldsymbol{\theta})$, which is the $\delta\Sigma_{jj}$ factor, and the Euler approximation, which supplies the additive constant, $\delta/4$. For more general posterior distributions, the mean of the stationary distribution of the SGLD algorithm will not necessarily be correct, but we would expect the mean to be more accurate than the variance, with the variance of SGLD being greater than that of the

true posterior. This analysis further suggests that, for posterior distributions which are close to Gaussian, it may be possible to perform a better correction to compensate for the inflation of the variance. For example, we could replace $\mathbf{Z}_k$ with Gaussian random variables where the covariance matrix is chosen such that the covariance matrix of $\delta \boldsymbol{\nu}_k + \sqrt{\delta}\mathbf{Z}$ becomes the identity matrix.

3.4 A General Framework for stochastic gradient MCMC

Stochastic gradient MCMC is not limited to approximating the Langevin diffusion. We can construct other diffusion processes that also have π as their stationary distribution. By approximating the dynamics of these alternative diffusions, we can develop stochastic gradient versions of many popular MCMC algorithms beyond the Langevin method.

Ma et al. (2015) proposed a general framework for stochastic gradient MCMC that extends the approach beyond the Langevin diffusion. This framework allows us to develop stochastic gradient analogues of algorithms like Hamiltonian Monte Carlo (HMC) (as introduced in Section 2.2), which leverages Hamiltonian dynamics. stochastic gradient HMC (SGHMC) has been shown to display improved mixing properties compared to stochastic gradient Langevin dynamics.

Beyond both SGLD and SGHMC, stochastic gradient MCMC approaches provide a general and flexible framework to adapt many MCMC algorithms to large datasets where full-data MCMC is infeasible. By approximating the dynamics of various diffusions with the correct stationary distribution, it is possible to develop fast, scalable MCMC algorithms tailored to different posterior geometries and datasets.

We introduce a general class of diffusion models, that may also include auxiliary variables, and we denote the full state by $\boldsymbol{\vartheta} \in \mathbb{R}^{d_{\boldsymbol{\vartheta}}}$. This state contains our variable of interest $\boldsymbol{\theta}$, as well as an auxiliary variable $\mathbf{p}$. For example, in the case of the overdamped Langevin diffusion, $\boldsymbol{\vartheta} = \boldsymbol{\theta}$, and extending the state space to include a velocity variable $\mathbf{p}$ allows for the underdamped Langevin diffusion of Section 1.4.3, which relates to Hamiltonian MCMC, giving $\boldsymbol{\vartheta} = (\boldsymbol{\theta}, \mathbf{p})$.

We define the general stochastic differential equation for $\boldsymbol{\vartheta}$ as

$$\mathrm{d}\boldsymbol{\vartheta} = \frac{1}{2}\mathbf{b}(\boldsymbol{\vartheta})\mathrm{d}t + \sqrt{\mathbf{D}(\boldsymbol{\vartheta})}\mathrm{d}W_t, \tag{3.22}$$

where $\mathbf{b}(\boldsymbol{\vartheta})$ is the *drift term*, $\mathbf{D}(\boldsymbol{\vartheta})$ is a positive semidefinite *diffusion matrix*

with matrix square root $\sqrt{\mathbf{D}(\boldsymbol{\vartheta})}$ and W_t denotes $d_{\boldsymbol{\vartheta}}$-dimensional Brownian motion. Ma et al. (2015) provide a general framework for how to choose $\mathbf{b}(\boldsymbol{\vartheta})$ and $\mathbf{D}(\boldsymbol{\vartheta})$ to achieve a desired stationary distribution. Given a general function $H(\boldsymbol{\vartheta})$, where $\exp\{-H(\boldsymbol{\vartheta})\}$ is integrable, simulating from an SDE with drift term,

$$\mathbf{b}(\boldsymbol{\vartheta}) = -\left[\mathbf{D}(\boldsymbol{\vartheta}) + \mathbf{Q}(\boldsymbol{\vartheta})\right] \nabla H(\boldsymbol{\vartheta}) + \Gamma(\boldsymbol{\vartheta}), \text{ where} \tag{3.23}$$
$$\Gamma_i(\boldsymbol{\vartheta}) = \sum_{j=1}^{d} \frac{\partial}{\partial \boldsymbol{\vartheta}_j}(\mathbf{D}_{ij}(\boldsymbol{\vartheta}) + \mathbf{Q}_{ij}(\boldsymbol{\vartheta})),$$

ensures that the stationary distribution of (3.22) is proportional to $\exp\{-H(\boldsymbol{\vartheta})\}$. The matrix $\mathbf{Q}(\boldsymbol{\vartheta})$ is a skew-symmetric curl matrix which controls the deterministic traversing effects of the SDE sampler, whereas $\mathbf{D}(\boldsymbol{\vartheta})$ controls the diffuse dynamics of the process. The general SDE (3.22) can be decomposed into (i) *Riemannian-Langevin dynamics* which are reversible and controlled by $\mathbf{D}(\boldsymbol{\vartheta})$ and (ii) *deterministic Hamiltonian dynamics* which are irreversible and controlled by $\mathbf{Q}(\boldsymbol{\vartheta})$. For the Langevin-type algorithms, where $\mathbf{Q}(\boldsymbol{\vartheta}) = 0$, it can be shown that they are able to quickly converge towards a mode of the distribution and diffusively explore the region around the mode. For the deterministic Hamiltonian Monte Carlo algorithm, where $\mathbf{D}(\boldsymbol{\vartheta}) = 0$, the algorithm excels at deterministically traversing along level sets of the Hamiltonian.

To approximately sample from the SDE we discretise the dynamics using the same Euler–Maruyama scheme used for the Langevin diffusion (3.1), i.e.,

$$\boldsymbol{\vartheta}_{k+\delta} = \boldsymbol{\vartheta}_k - \frac{\delta}{2}\left[(\mathbf{D}(\boldsymbol{\vartheta}_k) + \mathbf{Q}(\boldsymbol{\vartheta}_k))\nabla H(\boldsymbol{\vartheta}_k) + \Gamma(\boldsymbol{\vartheta}_k)\right] + \sqrt{\delta}\mathbf{Z}, \quad k \geq 0, \tag{3.24}$$

where $\mathbf{Z} \sim \mathsf{N}(0, \mathbf{D}(\boldsymbol{\vartheta}_k))$. If $\boldsymbol{\vartheta} = \boldsymbol{\theta}$, the posterior distribution π is the stationary distribution for the choice $H(\boldsymbol{\vartheta}) = -\log \pi(\boldsymbol{\theta})$. If we include an auxiliary variable, i.e. $\boldsymbol{\vartheta} = (\boldsymbol{\theta}, \mathbf{p})$, and we choose $H(\boldsymbol{\vartheta}) = -\log \pi(\boldsymbol{\theta}) + K(\mathbf{p})$ for some user-chosen function $K(\cdot)$, then π is the $\boldsymbol{\theta}$-marginal of the stationary distribution.

From the general SDE framework (3.22), we can obtain *stochastic gradient MCMC* (SGMCMC) algorithms by replacing $\nabla H(\boldsymbol{\vartheta}_k)$ in (3.24) with an unbiased estimate $\widehat{\nabla} H(\boldsymbol{\vartheta}_k)$ that is evaluated on a subsample of the dataset. As shown in Section 3.3.3, and Figure 3.2, the statistical efficiency of stochastic gradient algorithms is strongly tied to the variance of the gradient estimate. We can view that the variability of the gradient estimate inflates the noise input into (3.24) – which will lead to a stationary distribu-

tion whose variance is also inflated. Therefore, where feasible, we should correct for this. If we define $\mathbf{V}(\boldsymbol{\vartheta}_k) = \text{Var}\left[\widehat{\nabla} H(\boldsymbol{\vartheta}_k)\right]$ as the variance of the stochastic gradient, then the conditional variance of $\boldsymbol{\vartheta}_{k+\delta}$ given $\boldsymbol{\vartheta}_k$ will be inflated by $\delta^2\mathbf{B}(\boldsymbol{\vartheta}_k)$, where

$$\mathbf{B}(\boldsymbol{\vartheta}_k) = \frac{1}{4}(\mathbf{D}(\boldsymbol{\vartheta}_k) + \mathbf{Q}(\boldsymbol{\vartheta}_k))\mathbf{V}(\boldsymbol{\vartheta}_k)(\mathbf{D}(\boldsymbol{\vartheta}_k) + \mathbf{Q}(\boldsymbol{\vartheta}_k))^\top. \tag{3.25}$$

Correcting for this inflation in the SGMCMC setting leads to a modified discrete-time algorithm in (3.24), where $\nabla H(\boldsymbol{\vartheta}_k)$ is replaced by the stochastic approximation $\widehat{\nabla} H(\boldsymbol{\vartheta}_k)$ but we also reduce the variance of the noise term, so $\mathbf{Z} \sim \mathsf{N}(0, \mathbf{D}(\boldsymbol{\vartheta}_k) - \delta\mathbf{B}(\boldsymbol{\vartheta}_k))$. The challenge with this idea in practice is how do we estimate $\mathbf{B}(\boldsymbol{\vartheta}_k)$?

Through different choices of $H(\boldsymbol{\vartheta})$, $\mathbf{D}(\boldsymbol{\vartheta})$ and $\mathbf{Q}(\boldsymbol{\vartheta})$, we can derive several SGMCMC algorithms as special cases of the general discretised SDE (3.24). Some of these special cases will be introduced in the following sections.

Stochastic Gradient Langevin Dynamics

The SGLD algorithm (introduced in Section 3.3) follows from the dynamics of the general SDE (3.24) by setting

$$\boldsymbol{\vartheta} = \boldsymbol{\theta}, \quad H(\boldsymbol{\vartheta}) = -\log \pi(\boldsymbol{\theta}), \quad \mathbf{D}(\boldsymbol{\vartheta}) = \mathbf{I}, \quad \mathbf{Q}(\boldsymbol{\vartheta}) = \mathbf{0}$$

to give dynamics

$$\boldsymbol{\theta}_{k+\delta} = \boldsymbol{\theta}_k + \frac{\delta}{2}\left[\nabla \log \pi(\boldsymbol{\theta}_k)\right] + \sqrt{\delta}\mathbf{Z}, \quad k \geq 0,$$

which is the same Euler–Maruyama approximation of the Langevin diffusion introduced in (3.2), but where in practice $\nabla \log \pi(\boldsymbol{\theta}_k)$ would be replaced with a stochastic gradient estimator, using, for example, (3.6), (3.8) or (3.14).

Stochastic Gradient Hamiltonian Monte Carlo

The popular HMC algorithm introduced in Section 2.2 can also be derived as a special case of the general SDE dynamics (3.24). As discussed in Section 2.2, the HMC algorithm introduces a velocity component $\mathbf{p}$ to improve the mixing of the Markov chain and a mass matrix $\mathbf{M}$, where the Hamiltonian dynamics are used to update the position $\boldsymbol{\theta}$ and velocity components $\mathbf{p}$. In practice, the HMC algorithm uses the leapfrog numerical integration scheme to minimise numerical errors, however, for the purpose of illustration, we shall consider the simpler Euler integration scheme for creating a stochastic gradient HMC algorithm.

The Euler-discretised Hamiltonian dynamics for the state $\boldsymbol{\vartheta} = (\boldsymbol{\theta}, \mathbf{p})$ are

$$\begin{pmatrix} \boldsymbol{\theta}_{k+\delta} \\ \mathbf{p}_{k+\delta} \end{pmatrix} = \begin{pmatrix} \boldsymbol{\theta}_k \\ \mathbf{p}_k \end{pmatrix} + \frac{\delta}{2} \begin{bmatrix} \mathbf{M}^{-1}\mathbf{p}_k \\ \nabla \log \pi(\boldsymbol{\theta}_k) \end{bmatrix}, \quad k \geq 0, \tag{3.26}$$

which fits into the general SDE framework of (3.22) and (3.23) by setting

$$H(\boldsymbol{\vartheta}) = -\log \pi(\boldsymbol{\theta}) + \frac{1}{2}\mathbf{p}^\top \mathbf{M}^{-1}\mathbf{p}, \;\; \mathbf{D}(\boldsymbol{\vartheta}) = \mathbf{0} \;\text{ and }\; \mathbf{Q}(\boldsymbol{\vartheta}) = \begin{pmatrix} 0 & -\mathbf{I} \\ \mathbf{I} & 0 \end{pmatrix}.$$

If the gradient $\nabla \log \pi(\boldsymbol{\theta})$ in (3.26) is replaced by a stochastic gradient $\hat{\nabla} \log \pi(\boldsymbol{\theta}) = \nabla \log \pi(\boldsymbol{\theta}) + \mathsf{N}(0, \mathbf{V}(\boldsymbol{\theta}))$, where $\mathbf{V}(\boldsymbol{\theta})$ is the variance of the stochastic gradient, then under this stochastic setting the dynamics in (3.26) will become

$$\mathbf{p}_{k+\delta} = \mathbf{p}_k + \frac{\delta}{2}\nabla \log \pi(\boldsymbol{\theta}) + \mathsf{N}(0, \delta \mathbf{V}(\boldsymbol{\theta}_k)),$$

which is known as the *naive* stochastic gradient HMC (SGHMC) algorithm (Chen et al., 2014; Ma et al., 2015). It was proved in Chen et al. (2014) that the naive SGHMC algorithm does not work well as the error from estimating the gradient will accumulate over iterations and cannot be controlled. To overcome this, the authors suggest adding a *friction term* for the velocity, this is equivalent to using a stochastic gradient version of the *underdamped Langevin dynamics*, of Section 1.4.3. The intuition is that the introduction of friction means that errors from previous iterations will decay geometrically so that the overall error from using a stochastic gradient can be controlled. We can further improve accuracy by using the idea of correcting the variance of the injected noise that is introduced at each iteration.

Returning to the general SDE framework (3.22), the corrected stochastic gradient HMC algorithm follows by setting

$$\boldsymbol{\vartheta} = (\boldsymbol{\theta}, \mathbf{p}), \quad H(\boldsymbol{\vartheta}) = -\log \pi(\boldsymbol{\theta}) + \frac{1}{2}\mathbf{p}^\top \mathbf{M}^{-1}\mathbf{p},$$
$$\mathbf{D}(\boldsymbol{\vartheta}) = \begin{pmatrix} 0 & 0 \\ 0 & \mathbf{C} \end{pmatrix}, \quad \mathbf{Q}(\boldsymbol{\vartheta}) = \begin{pmatrix} 0 & -\mathbf{I} \\ \mathbf{I} & 0 \end{pmatrix},$$

where $\mathbf{C}$ is as a generic matrix, sometimes known as the *friction* term, and is chosen such that $\mathbf{C} \geq \delta \mathbf{V}(\boldsymbol{\theta})$ is a positive semi-definite matrix. Discretising this general form SDE with this particular $\mathbf{D}(\boldsymbol{\vartheta})$ and $\mathbf{Q}(\boldsymbol{\vartheta})$ leads to the dynamics,

$$\begin{pmatrix} \boldsymbol{\theta}_{k+\delta} \\ \mathbf{p}_{k+\delta} \end{pmatrix} = \begin{pmatrix} \boldsymbol{\theta}_k \\ \mathbf{p}_k \end{pmatrix} + \frac{\delta}{2} \begin{bmatrix} \mathbf{M}^{-1}\mathbf{p}_k \\ \hat{\nabla} \log \pi(\boldsymbol{\theta}_k) - \mathbf{C}\mathbf{M}^{-1}\mathbf{p}_k \end{bmatrix} + \begin{bmatrix} 0 \\ \sqrt{\delta}\mathbf{Z} \end{bmatrix}, \quad k \geq 0. \tag{3.27}$$

The gradient $\nabla \log \pi(\boldsymbol{\theta}_k)$ is replaced by a stochastic estimator $\hat{\nabla} \log \pi(\boldsymbol{\theta}_k)$ and $\mathbf{Z} \sim \mathsf{N}(0, \mathbf{C} - \delta\hat{\mathbf{B}})$, where $\hat{\mathbf{B}}$ is an estimate of $\mathbf{V}(\boldsymbol{\theta}_k)$.

The efficiency with which an MCMC algorithm can explore a posterior distribution is heavily tied to the geometry of the posterior. If the $\boldsymbol{\theta}$ components of the posterior distribution are strongly correlated, then the step size will need to be optimised for the component with the smallest variability in order to ensure that the algorithm does not diverge (as illustrated in Section 3.3.3 for the case of a Gaussian target distribution). This will significantly reduce the mixing time of the other components unless the posterior distribution is reparameterised so that the components of $\boldsymbol{\theta}$ are uncorrelated and have similar marginal distributions. Within the context of SGMCMC dynamics, it is possible to develop algorithms that incorporate reparameterisation by preconditioning the gradients with a positive-definite matrix $\mathbf{G}(\boldsymbol{\theta})$. If $\mathbf{G}(\boldsymbol{\theta})$ is the expected Fisher information of the posterior distribution, then the SGMCMC dynamics will be locally adapted to the posterior curvature by exploiting the Riemannian geometry of the posterior distribution.

Stochastic Gradient Riemannian Langevin Dynamics

Riemannian versions of SGLD (*stochastic gradient Riemannian Langevin dynamics*; SGRLD) and SGHMC (*stochastic gradient Riemannian Hamiltonian Monte Carlo*; SGRHMC) also follow as special cases of the general-form SDE (3.23). We can derive SGRLD by setting

$$\boldsymbol{\vartheta} = \boldsymbol{\theta}, \quad H(\boldsymbol{\vartheta}) = -\log \pi(\boldsymbol{\theta}), \quad \mathbf{D}(\boldsymbol{\vartheta}) = \mathbf{G}(\boldsymbol{\theta})^{-1}, \quad \mathbf{Q}(\boldsymbol{\vartheta}) = \mathbf{0}$$

which leads to the discrete-time dynamics

$$\boldsymbol{\theta}_{k+\delta} = \boldsymbol{\theta}_k + \frac{\delta}{2}\left[\mathbf{G}(\boldsymbol{\theta}_k)^{-1}\hat{\nabla} \log \pi(\boldsymbol{\theta}_k) + \Gamma(\boldsymbol{\theta}_k)\right] + \sqrt{h}\mathbf{Z}, \quad k \geq 0,$$

with $\mathbf{Z} \sim \mathsf{N}(0, \mathbf{G}(\boldsymbol{\theta}_k)^{-1})$ and $\hat{\nabla} \log \pi(\boldsymbol{\theta}_k)$ is a stochastic estimator for $\nabla \log \pi(\boldsymbol{\theta}_k)$. The term $\mathbf{G}(\boldsymbol{\theta}_k)^{-1}\nabla \log \pi(\boldsymbol{\theta}_k)$ is the *natural gradient* of the posterior distribution which gives the direction of steepest ascent by taking into account the geometry implied by $\mathbf{G}(\boldsymbol{\theta}_k)$. If $\mathbf{G}(\boldsymbol{\theta}) = \mathbf{I}$, then the direction of steepest ascent would be given in the Euclidean space. The term $\Gamma_i(\boldsymbol{\theta}) = \sum_j \frac{\partial(\mathbf{G}(\boldsymbol{\theta})^{-1})_{ij}}{\partial \boldsymbol{\theta}_j}$ accounts for the curvature of the manifold defined by $\mathbf{G}(\boldsymbol{\theta})$. A stochastic gradient HMC implementation that utilises the Riemannian geometry of the posterior distribution can also be derived within the general SDE framework. Taking the curl matrix $\mathbf{Q}(\boldsymbol{\theta})$ from SGHMC

and replacing the identity matrices with $\mathbf{G}(\boldsymbol{\theta})^{-1/2}$ leads to an SGHMC algorithm that is adaptive to the local posterior geometry.

Theoretical comparison of SGMCMC algorithms

When comparing the variety of SGMCMC algorithms it is natural to consider which algorithm is the most accurate and computationally efficient. It is possible to compare the theoretical convergence rates for some of these algorithms. In the case of SGHMC and SGLD, and in the context of smooth and strongly log-concave posteriors, it is possible to derive bounds on the Wasserstein-2 distance between the posterior and the SGMCMC sample distribution. These results are non-asymptotic and bound the Wasserstein-2 error for some k iterations of the SGMCMC algorithm using optimally tuned step sizes. These results show that if the stochastic gradients have low variance, then SGLD requires $O(d^2/\epsilon^2)$ iterations for a given accuracy ϵ, while SGHMC needs only $O(d/\epsilon)$. So SGHMC is generally preferred, with increasing benefits in higher dimensions (see Figure 3.4 for a numerical illustration). However, there is a phase transition in SGHMC as the gradient noise variance grows, above a threshold SGHMC behaves like SGLD, requiring $O(d^2/\epsilon^2)$ iterations.

3.5 Guidance for Efficient Scalable Bayesian Learning

Each of the SGMCMC algorithms introduced has tuning parameters that need to be chosen by the user when running these algorithms. Some SGMCMC algorithms, such as SGLD with control variates, require additional tuning parameters, e.g. choosing a control variate, but common to all SGMCMC algorithms is the step size parameter δ, the subsample size m, and the number of Monte Carlo iterations n.

For MCMC algorithms with Metropolis–Hastings corrections, such as MALA (Section 2.1.4) and HMC (Section 2.2), and which do not utilise data subsampling, the main tuning parameter is the step size δ. Existing theoretical results have established that the optimal δ should be chosen such that the Metropolis–Hastings acceptance rate is 57.4% (in the case of MALA) or at least 65% (in the case of HMC). Under certain assumptions on the posterior distribution, this choice minimises the integrated auto-correlation time of the Markov chain produced by these algorithms (see Section 1.3.2 for further details).

In the case of SGMCMC algorithms, which do not include a Metropolis–Hastings acceptance step, minimising the auto-correlation of the Markov

chain is not an appropriate objective for optimising δ. Intuitively, this is because the auto-correlation can be reduced by simply letting $\delta \to \infty$, but in the case of the ULA and SGMCMC algorithms, this will increase the bias in the discretised diffusion process and lead to poor posterior approximations. This is not an issue for MALA and HMC as the Metropolis–Hastings acceptance rate will tend to zero as $\delta \to \infty$, and so in this setting, minimising the auto-correlation time will balance between making large jumps in the posterior space (i.e large δ) against having a reasonable acceptance rate. Alternative metrics are required to minimise the variance in the Markov chain and account for the asymptotic bias present in SGMCMC algorithms. A popular choice is the kernel Stein discrepancy (KSD) metric, a kernelised version of the Stein discrepancy discussed in detail in Chapter 6. The KSD provides a measure of discrepancy between the true posterior distribution and the Monte Carlo approximation generated by an SGMCMC or other MCMC algorithm. Using the KSD metric, it is possible to optimise δ (and other SGMCMC tuning parameters) by assessing various tuning parameters over a grid of possible values and selecting the tuning parameters which minimise the KSD (Coullon et al., 2023). Related ideas, based on Stein's method and explored in Chapter 6, can be used to optimally thin the Markov chain to maximise the information about the posterior contained by a smaller set of Monte Carlo samples.

The step size parameter δ in SGLD can be chosen using the convergence results in Section 3.3.3. The upper bound on the Wasserstein-2 distance from Dalalyan and Karagulyan (2019), under assumptions 3.17 and 3.18, requires $\delta \leq 2/(l + L)$ in order to establish convergence for the SGLD algorithm. Setting δ to be too small leads to slow convergence of the Markov chain, but a step size that is too large can cause the SGLD algorithm to diverge. If l and L are known, then this information can be used to set δ. For example, returning to the running Gaussian example (3.3) where the posterior distribution is $\boldsymbol{\theta}|\mathbf{y} \sim \mathsf{N}(\boldsymbol{\mu}_N, \boldsymbol{\Sigma}_N)$, we know that $\nabla \log \pi(\boldsymbol{\theta}) = -\boldsymbol{\Sigma}_N^{-1}(\boldsymbol{\theta} - \boldsymbol{\mu}_N)$ and that the Lipschitz constant L measures the largest change in the gradient. Taking the Hessian *i.e.*, $\nabla\nabla \log \pi(\boldsymbol{\theta}) = -\boldsymbol{\Sigma}_N^{-1}$, the Lipschitz constant is equal to the spectral norm of the inverse covariance matrix $L = \|\nabla \log \pi(\boldsymbol{\theta})\| \leq \|\nabla\nabla \log \pi(\boldsymbol{\theta})\| \leq \|\boldsymbol{\Sigma}_N^{-1}\| = 1/\lambda_{\min}(\boldsymbol{\Sigma}_N)$, which is equal to the reciprocal of the smallest eigenvalue of the covariance matrix. As for the smoothness parameter l, this is the largest eigenvalue of the Hessian $l = \|\nabla\nabla \log \pi(\boldsymbol{\theta})\| \geq \lambda_{\max}(\boldsymbol{\Sigma}_N)$. Therefore, if $\delta \leq 2/(l + L)$, then $\delta \approx \lambda_{\min}(\boldsymbol{\Sigma}_N)/\lambda_{\max}(\boldsymbol{\Sigma}_N)$, which is equal to the condition number $\kappa = L/l$.

We can assess the relationship between the step size parameter and the properties of the Gaussian model by considering two covariance functions Σ

from the Gaussian model (3.3), where $\boldsymbol{\Sigma}^{(i)} = \begin{pmatrix} 1 & 0 \\ 0 & 10 \end{pmatrix}$ and $\boldsymbol{\Sigma}^{(ii)} = \begin{pmatrix} 1 & 3 \\ 3 & 10 \end{pmatrix}$. Under covariance $\boldsymbol{\Sigma}^{(i)}$, the variables $\boldsymbol{\theta}$ are uncorrelated whereas for $\boldsymbol{\Sigma}^{(ii)}$ there is imposed correlation between the components of $\boldsymbol{\theta}$. This leads to condition numbers $\kappa^{(i)} \approx 10^{-3}$ and $\kappa^{(ii)} \approx 10^{-4}$ for $\boldsymbol{\Sigma}^{(i)}$ and $\boldsymbol{\Sigma}^{(ii)}$, respectively. In Figure 3.3, we plot the Wasserstein-2 distance between the true Gaussian posterior distribution with $N = 1000$ data points with the approximation generated from $n = 10000$ iterations of the SGLD algorithm, where the step size parameter δ is varied over a grid of values $\delta \in \{10^{-5}, \ldots, 10^{-1}\}$. The left panel of Figure 3.3 is for the model with uncorrelated covariance matrix $\boldsymbol{\Sigma}^{(i)}$ and the right panel uses $\boldsymbol{\Sigma}^{(ii)}$. For both experiments, there is a value for δ which minimises the Wasserstein-2 distance and in the case of uncorrelated variables, i.e. $\boldsymbol{\Sigma}^{(i)}$, the optimal δ is larger than in the correlated case $\boldsymbol{\Sigma}^{(ii)}$. The dot indicates the step size recommended by the theoretical results, i.e. $\delta_{\text{opt}} = L/l$, which closely aligns with the optimal grid search result.

In general settings, however, it is not possible to calculate the optimal step size parameter as this depends on properties of the unknown posterior distribution. Under certain conditions on the posterior distribution, and assuming that N is sufficiently large, then by the Bernstein–von Mises theorem, the variance of the posterior distribution will be of order $O(d/N)$. Therefore, setting the step size parameter to be proportional to $1/N$, i.e. $\delta \propto 1/N$, gives a simple heuristic step size which is often used by practitioners. Note that, for the Gaussian posterior example given above, this heuristic would lead to a step size $\delta = 1/N = 10^{-3}$, which matches the optimal step size parameter for the setting with uncorrelated $\boldsymbol{\theta}$ components, i.e. $\boldsymbol{\Sigma}^{(i)}$ (see the left panel in Figure 3.3). However, this step size would be too large for the correlated setting with covariance $\Sigma^{(ii)}$ (see right panel in Figure 3.3). An alternative perspective discussed in Section 3.3.1 is that for the SGLD-type algorithms, the variance of the gradient component of the Langevin dynamics is $O(\delta^2)$ and therefore dominated by the $O(\delta)$ variance from the injected noise of the process. For the SGLD algorithm with a simple unbiased gradient estimator, the variance of the gradient is $O(N^2)$, and so a natural choice for δ to control the variance of the stochastic gradient is $\delta = 1/N$.

3.5.1 Experiments on a Logistic Regression Model

The logistic regression model, first introduced in Section 1.2.1, is used to predict the probability of binary outcomes $y_j \in \{0, 1\}$ given covariates

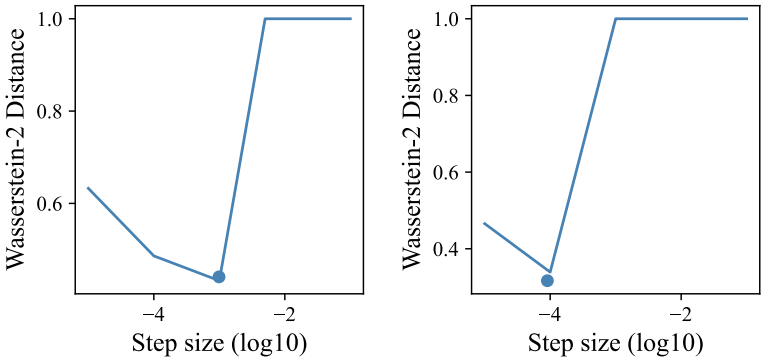

Figure 3.3 The Wasserstein-2 distance between the true and approximate posterior distributions when varying the step size parameter δ. Left panel is for the model with covariance $\Sigma^{(i)}$. Right panel is for the model with covariance $\Sigma^{(ii)}$.

$\mathbf{x}_j \in \mathbb{R}^{d_x}$. Assuming a Gaussian prior for the regression coefficients $\boldsymbol{\theta} \sim \mathsf{N}(\mathbf{0}, \Sigma_{\boldsymbol{\theta}})$, the posterior distribution, conditional on the data $\mathcal{D} = \{y_j, \mathbf{x}_j\}_{j=1}^N$, is given by the unnormalised density

$$\pi(\boldsymbol{\theta}) := \pi(\boldsymbol{\theta}|\mathcal{D}) \propto \exp\left\{-\frac{1}{2}\boldsymbol{\theta}^T\Sigma_{\boldsymbol{\theta}}^{-1}\boldsymbol{\theta}\right\}\prod_{j=1}^{N}\frac{\exp\{y_j\mathbf{x}_j^\top\boldsymbol{\theta}\}}{1+\exp\{\mathbf{x}_j^T\boldsymbol{\theta}\}}.$$

Metropolis–Hastings-based MCMC algorithms, such as MALA(Section 2.1.4) and HMC (Section 2.2), can be used to sample from the posterior distribution of the logistic regression model. However, in the large data setting, these algorithms will converge slowly due to the higher computational cost of evaluating the posterior density, and its gradient, on the full dataset. Whereas SGMCMC algorithms are faster, per Monte Carlo iteration, but introduce an asymptotic bias into the posterior approximation.

To assess the statistical accuracy of SGMCMC against exact MCMC approaches, consider a logistic regression model with $N = 10000$ observations, which is small enough to allow MALA and HMC approaches to be computationally feasible. The dataset is split into a training and test dataset with a 80/20 split. Data are simulated from the logistic regression model where the dimension of the parameter vector $\boldsymbol{\theta} \in \mathbb{R}^d$ is varied, $d \in \{100, 200, 300, 400, 500\}$. The statistical accuracy of the posterior approximation of the SGLD algorithm (Alg. 3) and SGHMC algorithm (3.27),

with and without control variates (3.7), is compared against a long Monte Carlo run of the NUTS algorithm (Hoffman and Gelman, 2014) using the Python package BlackJax (Cabezas et al., 2024), which provides a ground-truth Monte Carlo approximation to the posterior distribution. As noted in Section 3.5, standard MCMC diagnostics are not applicable for assessing the convergence of SGMCMC algorithms, and so as a proxy for posterior accuracy, the mean squared error (MSE) in the estimate of the posterior mean and variance given by the SGMCMC algorithms is compared against the posterior mean and variance taken from the NUTS samples. The NUTS and SGMCMC algorithms are each run for $n = 10000$ iterations and for the SGMCMC algorithms, a subsample size of 10% is used. Note that the step size parameter is fixed using the heuristic $\delta = 1/N$ for all experiments. Note that improved numerical results could be achieved by optimising the step size parameter for each dimension d.

Figure 3.4 shows the MSE in the estimate of $\mathbb{E}_\pi[\boldsymbol{\theta}]$ and $\mathrm{Var}_\pi[\boldsymbol{\theta}]$, where the true expectation and variance are given by the NUTS sampler. The plotted results are presented as the MSE averaged over the parameter dimension. The results show that the SGHMC algorithms are more robust to higher dimensions than the SGLD-type algorithms. This coincides with the established theory that compared to SGHMC, SGLD requires more iterations to achieve a similar level of accuracy. Note that for higher-dimensional problems, it may be necessary to run the SGD optimiser for longer to find the mode of the distribution that is used in control variate-based SGMCMC algorithms.

The posterior accuracy results in Figure 3.4 suggest that without increasing the number of Monte Carlo iterations, the SGMCMC algorithms will produce poorer posterior approximations with increasing parameter dimension. The results show that SGMCMC algorithms can produce highly accurate approximations for the first and second moment of the posterior distribution, and in the case of SGHMC, the first moment is very similar to the first moment given by the NUTS sampler. As illustrated previously in Figure 3.2, SGLD can produce good approximations to the first posterior moment, but for small data subsamples it tends to produce overestimates of the second posterior moment. The reason for the poorer posterior approximation of the SGLD-based samplers compared to the SGHMC-based samplers can be seen in the Monte Carlo trace plots in Figure 3.5. For the higher-dimensional setting ($d = 500$), we can see that for posterior components θ_1 and θ_2, the mixing is worse for SGLD and SGLD-CV compared to SGHMC and SGHMC-CV. This is then reflected in the Monte Carlo approximation for the posterior mean and variance (Figure 3.4), where SGLD

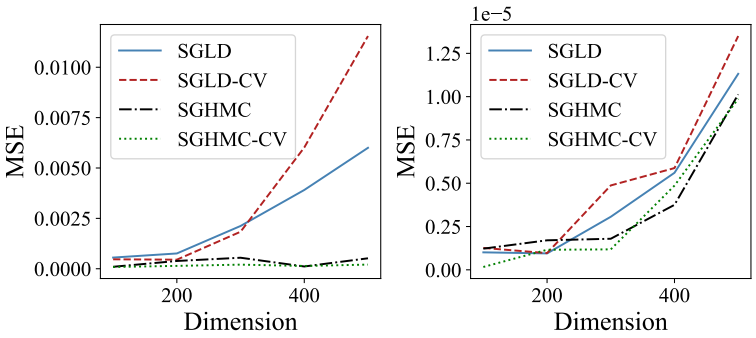

Figure 3.4 The mean squared error (MSE) of $\mathbb{E}_\pi[\boldsymbol{\theta}]$ (left panel) and $\text{Var}_\pi[\boldsymbol{\theta}]$ (right panel) compared to the same moments calculated from the NUTS posterior samples, which are treated as the ground-truth. The results plotted are for the average MSE taken over all dimensions of the mean and marginal variance of $\pi(\boldsymbol{\theta})$.

and SGLD-CV are not as accurate as SGHMC and SGHMC-CV when $d = 500$, but display similar levels of accuracy for $d = 100$ and $d = 200$. The mixing of the SGLD-based samplers could be improved by hand-tuning the step-size parameter δ, or preconditioning the gradients to account for the correlation structure of the posterior distribution.

Beyond posterior accuracy, we can also assess the predictive accuracy of SGMCMC algorithms against exact full-data MCMC algorithms, in this case using the NUTS sampler. Figure 3.6 illustrates that SGMCMC algorithms are competitive against slower Metropolis–Hastings-based algorithms when assessed against predictive accuracy. Figure 3.6 plots the percentage improvement in log-posterior predictive accuracy for each SGMCMC algorithm over the log-posterior predictive accuracy of the NUTS sampler, which is treated as the gold standard approach. The results are given for the logistic regression model on a test dataset using posterior samples over a range of parameter dimensions d. The results highlight that SGMCMC algorithms are competitive and potentially superior to slower, full-data, MCMC algorithms in terms of predictive accuracy, displaying only a small decrease in efficiency but with a significant computational advantage.

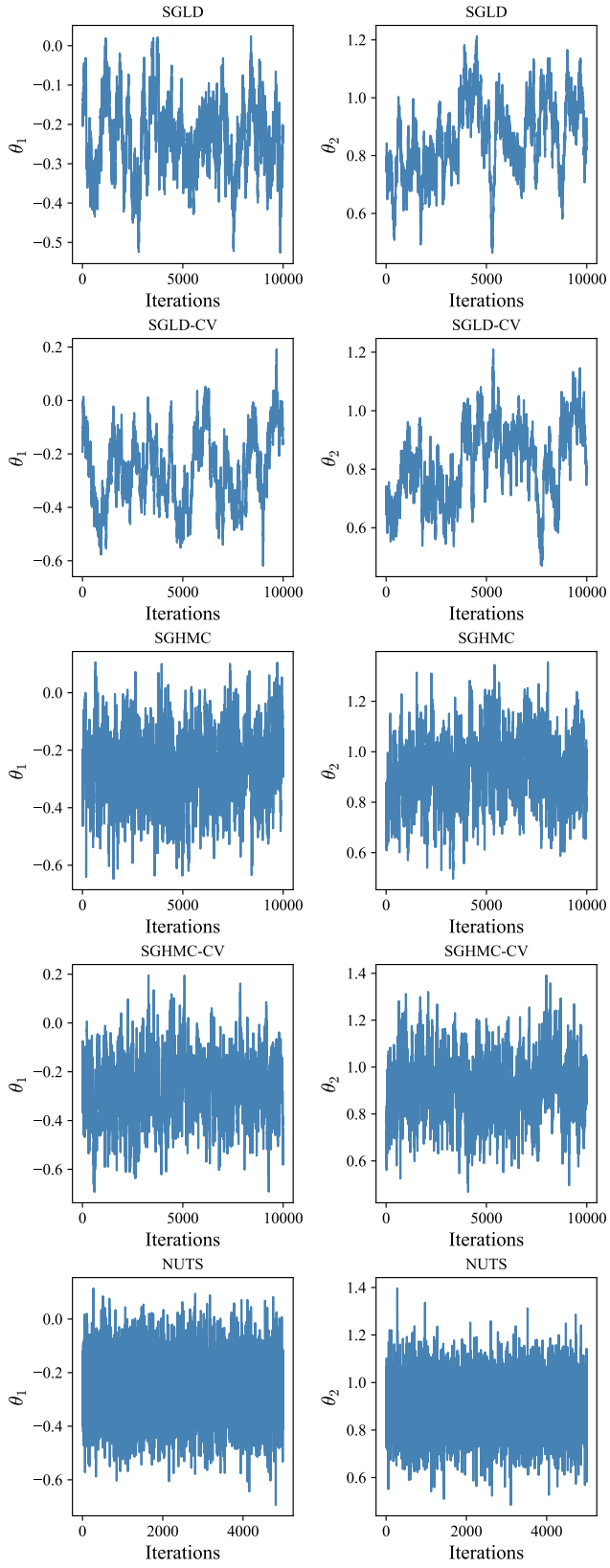

Figure 3.5 Trace plots for the first two components of $\boldsymbol{\theta}$ with $d = 500$.

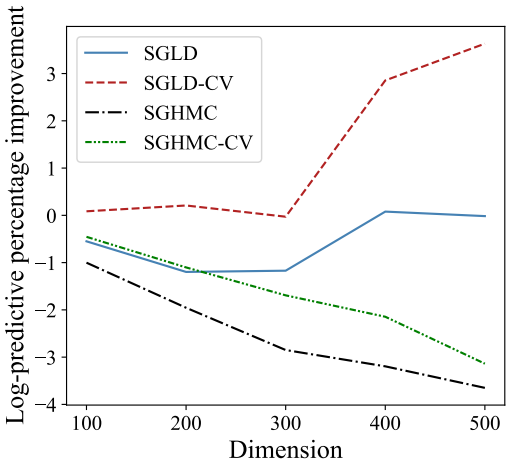

Figure 3.6 Percentage improvement in the log-predictive density of each SGMCMC algorithm relative to the log predictive density of the NUTS sampler on the logistic regression model calculated on a test data set. The dimension of the parameter of interest is $d \in \{100, 200, 300, 400, 500\}$.

3.5.2 Experiments on a Bayesian Neural Network Model

A Bayesian neural network model for multi-class classification was introduced in Section 1.2.3, where the dataset $\mathcal{D} = \{y_j, \mathbf{x}_j\}_{j=1}^N$is comprised of a collection of G classes $y_j \in \{1, \ldots, G\}$ and covariates $\mathbf{x}_j \in \mathbb{R}^{d_x}$. The unnormalised posterior density is

$$\pi(\boldsymbol{\theta}) := \pi(\boldsymbol{\theta}|\mathcal{D}) \propto \pi_0(\boldsymbol{\theta}) \prod_{j=1}^{N} \exp(\mathbf{A}_{y_j+1}^\top \sigma(\mathbf{B}^\top \mathbf{x}_j + \mathbf{b}) + a_{y_j+1}), \quad (3.28)$$

where $\sigma(\cdot)$ is a softmax activation function (see Section Section 1.2.3 for details). The model parameters $\boldsymbol{\theta} = \text{vec}(\mathbf{A}, \mathbf{B}, \mathbf{a}, \mathbf{b})$ are the weights $\mathbf{A}, \mathbf{B}$ and biases $\mathbf{a}, \mathbf{b}$ of the network model. We shall assume independent standard Gaussian priors for each parameter, i.e. $\boldsymbol{\theta} \sim \mathsf{N}(0, \mathbf{I})$.

Neural networks are commonly used for image classification tasks. One of the most fundamental and widely used datasets in image classification is the MNIST handwritten digit dataset. The MNIST dataset consists of images of handwritten digits, ranging from zero to nine, i.e.

Figure 3.7 A selection of digits from the MNIST dataset. Image source - Wikipedia.

$y_j \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ (see Figure 3.7 for a subsample of the dataset). Each image is represented by a small square of 28 pixels by 28 pixels which are treated as covariates. Each $\mathbf{x}_j \in \mathbb{R}^{784}$ is a vectorisation of a matrix made up of 28 rows and 28 columns, with each pixel containing grayscale information representing the darkness of that specific point in the image. A brighter pixel would have a higher value, while a darker one would have a lower value.

The Bayesian neural network for this example (3.28) has two layers: an input layer that receives the information from the 28×28 image, and a hidden layer containing 100 hidden variables that act as intermediate processing units. The parameters of the neural network $\boldsymbol{\theta}$ are of the form $\mathbf{A} \in \mathbb{R}^{10\times100}$, $\mathbf{B} \in \mathbb{R}^{784\times100}$, $\mathbf{a} \in \mathbb{R}^{1\times10}$, and $\mathbf{b} \in \mathbb{R}^{1\times100}$.

The MNIST dataset contains a large collection of 60000 images in the training set. Each image has a corresponding label, indicating which digit $(0-9)$ it represents. Using SGMCMC algorithms, we can approximate the posterior distribution of the Bayesian neural network using subsamples of the labelled images and pixel values to train the Bayesian neural network to recognise patterns and relationships between the pixels and the corresponding digits.

We use the SGLD and SGHMC algorithms from the Python package SGMCMCJax (Coullon and Nemeth, 2022), and their control-variate counterparts, to draw samples from the Bayesian neural network posterior (3.28). We run each algorithm for 2000 iterations and retain every 10th iteration of the Markov chain. A subsample size of 1% of the full dataset is used for all

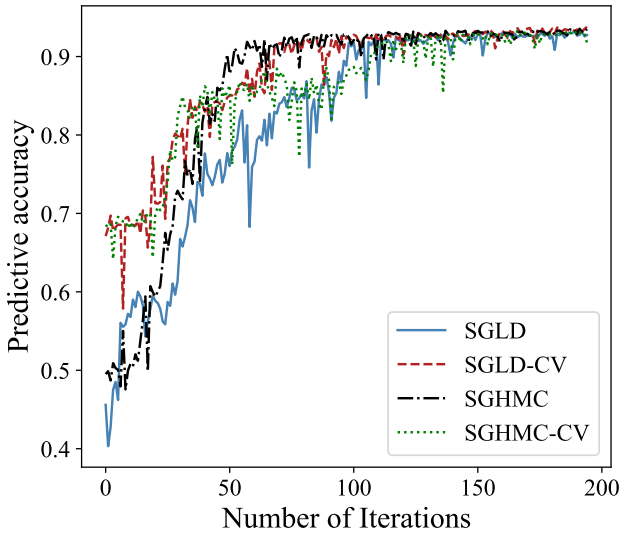

Figure 3.8 Average posterior predictive accuracy over all classes for each of the SGMCMC samplers.

SGMCMC samplers. For the control-variate-based algorithms, a stochastic gradient descent algorithm is used to find the posterior mode and the Markov chain is initialised at the posterior mode.

There is a separate set of 10000 unseen images, also with corresponding labels but hidden from the training process. This test set allows us to evaluate how well our neural network performs on new data. By feeding these new images into the network, we can see if the posterior network is able to accurately classify the images into one of the ten-digit categories (0-9). Figure 3.8 shows the posterior predictive accuracy for each SGMCMC sampler over the number of Monte Carlo iterations (storing every 10th posterior sample). The results show that all of the samplers converge to approximately 93% accuracy in classifying the MNIST test set digits. The SGLD and SGHMC samplers converge at a similar rate (in terms of predictive accuracy). The control variate-based SGLD and SGHMC samplers also converge at a similar rate to each other but achieve higher predictive accuracy with fewer Monte Carlo iterations as these samplers are initialised at the posterior mode and thus remove the burn-in phase of the Monte Carlo sampler.

3.6 Generalisations and Extensions

The SGMCMC framework outlined in Section 3.4 can be extended beyond the SGLD algorithm to improve Markov chain mixing. However, two key assumptions limit the applicability of current SGMCMC algorithms: (i) the parameters $\boldsymbol{\theta} \in \mathbb{R}^d$ exist in an unconstrained space, and (ii) the log-posterior density $\log \pi(\boldsymbol{\theta})$ is a sum over conditionally independent terms.

Assumption (i) precludes estimating $\boldsymbol{\theta}$ on a constrained space (e.g. $\boldsymbol{\theta} \in [0,1]^d$). Assumption (ii) requires data $\mathbf{y}_1, \ldots, \mathbf{y}_N$ to be independent or have limited dependence, restricting the applicability of SGMCMC for time series or spatial models.

Ongoing research aims to relax these assumptions and expand SGMCMC to broader model classes. Some promising directions include:

- Transformation techniques to enable sampling on constrained parameter spaces (Brosse et al., 2017; Bubeck et al., 2018; Hsieh et al., 2018).
- Exploiting short-range dependencies and other model structures to allow subsampling for time series and network data (Li et al., 2016; Ma et al., 2017; Aicher et al., 2023).
- Leveraging alternative stochastic processes with desired invariant distributions as samplers for models with complex data and parameter structures (Baker et al., 2018).

By developing specialised subsampling schemes and transformations, it is possible to make SGMCMC algorithms more applicable to a wider range of Bayesian models while retaining computational efficiency.

3.6.1 Scalable Inference for Models in Constrained Spaces

Many statistical models contain parameters with inherent constraints, such as the variance parameter τ^2 in a Gaussian distribution ($\tau \in \mathbb{R}^+$) or the success probability p in a Bernoulli model ($p \in [0,1]$). Simulating these constrained parameters using standard Langevin dynamics (3.2) will often produce samples violating the constraints. For instance, if at iteration k of the SGLD algorithm $\boldsymbol{\theta}_k = \tau_k^2$ is close to zero, then with high probability, the next iterate $\boldsymbol{\theta}_{k+1}$ is likely to be negative, breaking the positivity constraint. One solution is to shrink the step size $\delta \to 0$ as $\tau^2 \to 0$, but this leads to poor mixing near the boundary.

A natural approach is to transform the Langevin dynamics so sampling occurs in an unconstrained space, but the choice of transformation greatly

impacts mixing near the boundary. Alternatively, we can project the dynamics into the constrained space, however, this yields poorer convergence compared to the unconstrained setting. The *mirrored Langevin algorithm* (Hsieh et al., 2018) was proposed to address this issue. It builds on the mirrored descent algorithm (Beck and Teboulle, 2003) from the optimisation literature to transform constrained sampling into an unconstrained problem using a *mirror map*. Compared to a generic mapping function, mirror maps have additional properties, such as strict convexity, differentiability and diverging gradients at the boundary of the domain, which makes mirror map algorithms well-suited to constrained sampling and optimisation problems.

If we assume that $\pi(\boldsymbol{\theta})$ is the density of a constrained distribution, namely that $\log \pi(\boldsymbol{\theta})$ has a bounded convex domain, then assuming that there exists a mirror map $\psi(\cdot)$ which is closed and proper, we can map the variable $\boldsymbol{\theta} \sim \pi$ from the constrained space (primal space) to the unconstrained space (dual space), where $\boldsymbol{\vartheta} := \nabla\psi(\boldsymbol{\theta})$ and $\boldsymbol{\vartheta} \sim \nu$. Under the assumption that ψ is a convex function that is closed, proper, and twice continuously differentiable, with Fenchel dual noted as ψ^*, then Theorem 1 of Hsieh et al. (2018) shows that $\nabla\psi^*(\boldsymbol{\vartheta}) \sim \pi$. This result implies that it is possible to transform the problem of sampling from a constrained distribution π to the simpler problem of sampling from an unconstrained distribution ν. Using the algorithms highlighted in this chapter, we can simulate a Markov chain $\boldsymbol{\vartheta} \sim \nu$ and apply the mapping $\nabla\psi^*(\boldsymbol{\vartheta})$ to produce samples from the desired posterior distribution π. In the case of the Langevin sampler, this is achieved by modifying the original Langevin diffusion (3.1) to a *mirrored Langevin diffusion*,

$$\mathrm{d}\boldsymbol{\vartheta} = \frac{1}{2}(\nabla \log \nu \circ \nabla\psi)(\boldsymbol{\theta})\mathrm{d}t + \mathrm{d}W_t \tag{3.29}$$

$$\boldsymbol{\theta} = \nabla\psi^*(\boldsymbol{\vartheta}). \tag{3.30}$$

In practice, as noted earlier in this chapter, it is not possible to simulate exactly from the mirror Langevin diffusion. Using the same Euler–Maruyama discretisation scheme it is possible to create a practical discrete-time algorithm. Stochastic gradient implementations of the mirror Langevin dynamics are easily derived in the dual space and follow directly from the SGLD algorithm.

One popular model that requires sampling from a constrained domain is the latent Dirichlet allocation (LDA) model (Blei et al., 2003) which is used for topic modelling. Here, the model parameters are constrained to the probability simplex, i.e. $\theta_i \geq 0, i = 1, \ldots, d$ and $\sum_{i=1}^{d} \theta_i = 1$. Mirrored Langevin dynamics (3.29) can be used to simulate from the simplex

distribution by mapping the parameters to $\mathbb{R}^d$ and running the Langevin dynamics algorithm on the unconstrained space. The *entropic* mirror map (Beck and Teboulle, 2003) satisfies the required assumptions for a valid map function under the mirrored Langevin dynamics,

$$\psi(\boldsymbol{\theta}) = \sum_{i=1}^{d} \theta_i \log \theta_i + \left(1 - \sum_{i=1}^{d} \theta_i\right) \log\left(1 - \sum_{i=1}^{d} \theta_i\right), \tag{3.31}$$

where $0 \log 0 := 0$. The transformed log-posterior density is then given by

$$\log \nu(\boldsymbol{\vartheta}) = \log \pi(\boldsymbol{\theta}) \circ \nabla\psi^*(\boldsymbol{\vartheta}) - \sum_{i=1}^{d} \vartheta_i + (d+1)\psi^*(\boldsymbol{\vartheta}), \tag{3.32}$$

where $\psi^*(\boldsymbol{\vartheta}) = \log(1 + \sum_{i=1}^{d} \exp(\vartheta_i))$ is the Fenchal dual of $\psi(\cdot)$ and is strictly convex with Lipschitz gradients. Aside from transforming a constrained sampling problem into an unconstrained sampling problem, mirror maps can also lead to simpler posterior distributions in the dual space. For example, the Dirichlet posterior distribution introduced above, leads to a posterior distribution on the dual space (3.32) which is strictly log-concave.

Instead of using mirror maps to sample from the posterior distribution of the LDA model, we could instead use the stochastic gradient Riemannian Langevin dynamics algorithm (see Section 3.4). Under the SGRLD algorithm, the constrained parameters $\boldsymbol{\theta}$ can be transformed to $\mathbb{R}^d$ via several alternative reparameterisations (see Patterson and Teh (2013) for a list). However, this can induce asymptotic biases dominating the boundary regions. An alternative approach is to recognise that the LDA posterior can be expressed as a transformation of independent gamma random variables. Therefore, rather than simulating from the simplex distribution via the Langevin diffusion, one could instead utilise the Cox–Ingersoll–Ross (CIR) process (Cox et al., 1985), which is invariant with respect to the gamma distribution. This CIR-based approach (Baker et al., 2018) avoids boundary biases. More broadly, leveraging alternative stochastic processes with desired invariant distributions can enable specialised samplers for models with complex structures.

3.6.2 Scalable Inference with Time Series Data

A key requirement for developing stochastic gradient algorithms is the ability to generate unbiased estimates of $\nabla \log \pi(\boldsymbol{\theta})$ using data subsampling, as in (3.6). This is straightforward when the log-posterior density, and its gradient, are expressed as a sum of $\log \pi_i(\boldsymbol{\theta})$ and $\nabla \log \pi_i(\boldsymbol{\theta})$ terms,

respectively, i.e. $\log \pi(\boldsymbol{\theta}) = \sum_{i=1}^{N} \log \pi_i(\boldsymbol{\theta})$. Randomly subsampling these terms provides an unbiased log-density and gradient estimate.

However, for models where data are not conditionally independent, for example, network data, time series, or spatial data, the log-posterior density cannot be expressed as a simple sum. Naively subsampling will yield biased estimates of $\log \pi(\boldsymbol{\theta})$ and $\nabla \log \pi(\boldsymbol{\theta})$. Capturing both short- and long-range dependencies in spatial data with subsampling remains an open challenge for SGMCMC. For network data, Li et al. (2016) developed an SGMCMC algorithm for the mixed-membership stochastic block model using block structure and stratified subsampling to obtain unbiased gradient estimates.

Recent work in SGMCMC for temporally correlated data has focused on hidden Markov models (Ma et al., 2017) with finite states, linear dynamical systems (Aicher et al., 2019) and general nonlinear hidden Markov models (Aicher et al., 2023). Under this modelling framework, the hidden Markov model consists of two stochastic processes: i) a latent state process $\{\mathbf{X}_t\}_{t=0}^{T}$, which is a Markov chain that evolves over time $t = 1, \ldots, T$, with $\mathbf{X}_t$ depending only on $\mathbf{X}_{t-1}$ and $\boldsymbol{\theta}$, with transition density given by $p(\mathbf{x}_t|\mathbf{x}_{t-1}, \boldsymbol{\theta})$; and ii) an observed process $\{\mathbf{Y}_t\}_{t=0}^{T}$ that is conditionally independent given the latent states and $\boldsymbol{\theta}$, which are observed with probability density $p(\mathbf{y}_t|\mathbf{x}_t, \boldsymbol{\theta})$. Assuming model parameters $\boldsymbol{\theta}$, the full generative model (Figure 3.9) is

$$\mathbf{X}_t|(\mathbf{X}_{t-1} = \mathbf{x}_{t-1}, \boldsymbol{\theta}) \sim p(\mathbf{x}_t|\mathbf{x}_{t-1}, \boldsymbol{\theta}) \tag{3.33}$$

$$\mathbf{Y}_t|(\mathbf{X}_t = \mathbf{x}_t, \boldsymbol{\theta}) \sim p(\mathbf{y}_t|\mathbf{x}_t, \boldsymbol{\theta}). \tag{3.34}$$

The latent Markov chain $\mathbf{X}_t$ captures the dynamics and temporal dependence, while the observations $\mathbf{Y}_t$ depend only on the current state of the latent process. Hidden Markov models are useful for modelling complex time series data by augmenting the observables with latent states. It is often common to distinguish between hidden Markov models and state-space models, where in the case of the former the latent process $\mathbf{X}_t$ is discrete and is continuous in the case of the latter. However, for the sake of convenience, we shall use the term *hidden Markov model* to cover both model types.

Using SGMCMC methods, it is possible to estimate the model parameters $\boldsymbol{\theta}$ for general hidden Markov models which exhibit temporal dependency in the observations. As before, the goal is to sample from the posterior distribution $\pi(\boldsymbol{\theta}) := p(\boldsymbol{\theta}|\mathbf{y})$ where $\mathbf{y} = \{\mathbf{y}_1, \ldots, \mathbf{y}_T\}$ is the observed data sequence, which is proportional to the product of the likelihood $p(\mathbf{y}|\boldsymbol{\theta})$ and the prior $\pi_0(\boldsymbol{\theta})$. The likelihood $p(\mathbf{y}|\boldsymbol{\theta})$ typically cannot be evaluated exactly, as it requires summing (discrete setting) or integrating (continuous setting) over the latent states $\mathbf{X}$. Focusing on the continuous variable setting,

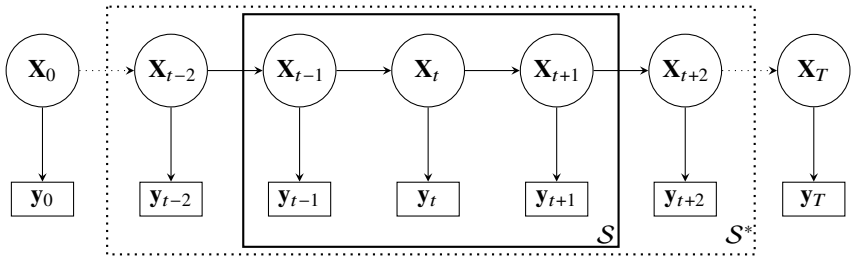

Figure 3.9 Graphical representation of the hidden Markov model with latent variables $\mathbf{X}_i$ and observations $\mathbf{y}_i$. The subsequence is captured in the solid box $\mathcal{S}$ and the buffer region is highlighted by the dotted box $\mathcal{S}^*$.

the latent states can be integrated out numerically using particle filtering techniques (Doucet et al., 2009). Using a particle approximation of Fisher's identity (Nemeth et al., 2016),

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \log p(\mathbf{y}_{1:T}|\boldsymbol{\theta}) &= \mathbb{E}_{\mathbf{X}|\mathbf{Y},\boldsymbol{\theta}} \left[\nabla_{\boldsymbol{\theta}} \log p(\mathbf{X}_{1:T}, \mathbf{y}_{1:T}|\boldsymbol{\theta})\right] \\ &= \sum_{t=1}^{T} \mathbb{E}_{\mathbf{X}|\mathbf{Y},\boldsymbol{\theta}} \left[\nabla_{\boldsymbol{\theta}} \log p(\mathbf{X}_t, \mathbf{y}_t|\mathbf{x}_{t-1}, \boldsymbol{\theta})\right]\end{aligned} \tag{3.35}$$

it is possible to unbiasedly approximate $\nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{\theta})$ by replacing the posterior distribution of the latent states $p(\mathbf{x}_{1:T}|\mathbf{y}_{1:T}, \boldsymbol{\theta})$ with a numerical approximation represented by a set of P particles $\{\mathbf{X}_{1:T}^{(p)}\}_{p=1}^{P}$.

Calculating Fisher's identity can be computationally expensive for large T and so an SGMCMC approximation to Fisher's identity can be used to replace the full data gradient (3.35) with a stochastic approximation estimated from a random subset of the data. This allows each gradient evaluation to be cheaper, enabling more MCMC iterations and better convergence for large T. However, naively subsampling the data randomly (3.6) induces bias since it breaks temporal dependencies. To address this issue, Aicher et al. (2019, 2023) propose to subsample contiguous subsequences. Figure 3.9 provides a graphical representation of the hidden Markov model which also highlights the subsampled data as a contiguous subsequence $\mathcal{S}$ of size m. The stochastic gradient following Fisher's identity (3.35) is then

$$\nabla_{\boldsymbol{\theta}}^{(m)} \log p(\mathbf{y}_{1:T}|\boldsymbol{\theta}) = \sum_{t \in \mathcal{S}} \Pr(t \in \mathcal{S})^{-1} \cdot \mathbb{E}_{\mathbf{X}|\mathbf{y}_{\mathcal{S}^*},\boldsymbol{\theta}} \left[\nabla_{\boldsymbol{\theta}} \log p(\mathbf{X}_t, \mathbf{y}_t|\mathbf{x}_{t-1}, \boldsymbol{\theta})\right]. \tag{3.36}$$

However, the stochastic gradient estimator (3.36) is biased because the expectation is taken over the latent state which is dependent only on $\mathbf{y}_{\mathcal{S}}$

and not $\mathbf{y}_{1:T}$. This bias can be reduced by extending the subsequence $\mathcal{S}$ to include a *buffered* region $\mathcal{S}^*$. The expectation in (3.35) is then taken over the wider buffered range $p(\mathbf{x}_t|\mathbf{y}_{\mathcal{S}^*}, \boldsymbol{\theta})$ rather than $p(\mathbf{x}_t|\mathbf{y}_{\mathcal{S}}, \boldsymbol{\theta})$. Results from Aicher et al. (2019) show that under Lipschitz assumptions on the model (3.33), and its gradients, the error in the stochastic gradients decays geometrically with the buffer size $|\mathcal{S}^*|$.

3.7 Chapter Notes

Markov chain Monte Carlo algorithms have been the cornerstone of Bayesian inference since the 1990s. However, as the size of datasets has grown, the requirement that each MCMC update uses all of the data to approximate the posterior distribution has created a computational challenge. There has been significant recent interest in the Statistics and Machine Learning communities to create new Monte Carlo-based algorithms for scalable Bayesian inference in the presence of large datasets (Bardenet et al., 2014). Broadly speaking, scalable MCMC algorithms tend to fall into two categories which either (i) subsample the data (Welling and Teh, 2011; Chen et al., 2014; Nemeth and Fearnhead, 2021), as covered in this chapter, or (ii) use parallel computing to distribute the computational cost across multiple CPUs (Scott et al., 2016; Nemeth and Sherlock, 2018; Vyner et al., 2023).

In the context of data subsampling, the per iteration cost of the Monte Carlo algorithm is reduced. However, this reduction in computational cost is only beneficial if the subsampling scheme leads to posterior approximations with high statistical accuracy. Chatterji et al. (2018) gives results on the number of iterations, and resulting computational cost, required for different stochastic gradient algorithms to produce samples from a distribution which is within a specified "distance" of π. Other works (Bierkens et al., 2019b; Huggins and Zou, 2017; Pollock et al., 2020) have studied the computational and statistical trade-offs that result from approximate and scalable MCMC schemes, which are often referred to as *exact-approximate algorithms*. It is often the case with scalable MCMC algorithms that *there is no free lunch* and simple naive subsampling alone does not lead to statistically efficient algorithms, see for example Johndrow et al. (2020). In the case of control-variates for subsampling, a number of theoretical results (e.g. Nagapetyan et al., 2017; Baker et al., 2019; Brosse et al., 2018) show that if we ignore the pre-processing cost of finding $\widehat{\boldsymbol{\theta}}$, the computational cost per-effective sample of SGLD with control variates is $O(1)$, rather than the $O(N)$ cost for SGLD with the simple gradient estimator (3.6).

4

Non-Reversible MCMC

A reversible Markov chain is any Markov chain that satisfies detailed balance; see Section 1.3. Remember, this condition states that, at stationarity, the probability of the chain starting in a set $\mathcal{B}$ and moving to set C is equal to the probability of it starting in the set C and moving to $\mathcal{B}$. This means that the dynamics of the process are the same backwards in time as forwards in time. One consequence of reversibility is that the Markov chain can exhibit random-walk behaviour, where it can return to states or regions of the state space where it has recently been.

A non-reversible Markov chain is any Markov chain that does not satisfy detailed balance. As we will see, the potential benefit of non-reversibility is that the Markov chain can more quickly explore the state space as it can suppress the random walk behaviour. However, designing non-reversible Markov chains with the required stationary distribution, or even determining the stationary distribution of a non-reversible Markov chain, is much more challenging than for a reversible chain. This chapter will describe one approach to designing non-reversible MCMC samplers based on the idea of *lifting* – which involves taking a reversible MCMC scheme and then lifting it to a higher-dimensional state-space to enable the use of non-reversible moves. These ideas naturally motivate the non-reversible continuous-time MCMC samplers of Chapter 5.

4.1 The Benefits of Non-Reversibility

To see the benefits of non-reversible Markov chains, we will consider the following simple example.

Example 4.1 Let $X_0 = 0$ and

$$X_k = (X_{k-1} + J_k) (\text{mod } S),$$

so that X_k follows a random walk on $\{0, \ldots, S-1\}$. Here J_k is the jump size,

which can have any distribution on $\{-h, \ldots, h\}$ that is symmetric about 0, and (mod S) means we take the remainder after dividing by S. We include (mod S), so a random walk that moves to negative values, or values equal to or above S, gets mapped back to $\{0, \ldots, S-1\}$, with S mapping to 0 and -1 to $S-1$ and so on. For simplicity, we consider J_k to have a uniform distribution on $\{-h, -h+1, \ldots, h\}$. It is straightforward to show that the resulting Markov chain is reversible and has the uniform distribution on $\{0, \ldots, S-1\}$ as its stationary distribution.

First, consider $h = 1$ and look at the behaviour of the chain as we increase S. In the top row of Figure 4.1, we show trace plots of the chain for $S = 100$, $S = 200$ and $S = 400$. In each case, we show the path of the chain over $40S$ iterations. What we observe is that as S increases the chain becomes much slower at exploring the state-space. We can see this more clearly if we look at the empirical marginal distribution of the chain after n time steps. Let b be a chosen burn-in period, then for $n > b$ the empirical marginal distribution, which is our natural estimator of the stationary distribution of the chain, is

$$\widehat{\pi}_n(i) = \frac{1}{n-b} \sum_{k=b+1}^{n} \mathbb{I}\{X_k = i\}\,. \tag{4.1}$$

This is just the proportion of time, after burn-in, that the chain was in state i. We can then compare this estimate with the true stationary distribution, by calculating the total variation distance between the two. This is just $\sum_{i=0}^{S-1} |\widehat{\pi}_n(i) - 1/S|$, and is shown in the top-row of Figure 4.2, where we show the total variation distance against n/S and against n/S^2 for $S = 100$, $S = 200$ and $S = 400$. We see evidence that, as we increase S, the time we need to run our Markov chain must increase proportionally with S^2 to have the same degree of accuracy.

It is interesting to compare this with the performance of the following non-reversible Markov chain, which we construct by making the random walk biased; *i.e.*, choosing a jump distribution J_k whose expectation is non-zero.

Example 4.2 In Example 4.1, keep J_k taking values in $\{-1, 0, 1\}$, but set the jump probabilities to be $\{2/9, 1/3, 4/9\}$ respectively so that a positive jump is twice as likely as a negative one.

We show the resulting trace plots of the Markov chain in the bottom row of Figure 4.1. Qualitatively the trace plots look very different to those in the top row, as the paths tend to move upwards at each iteration. This is linked

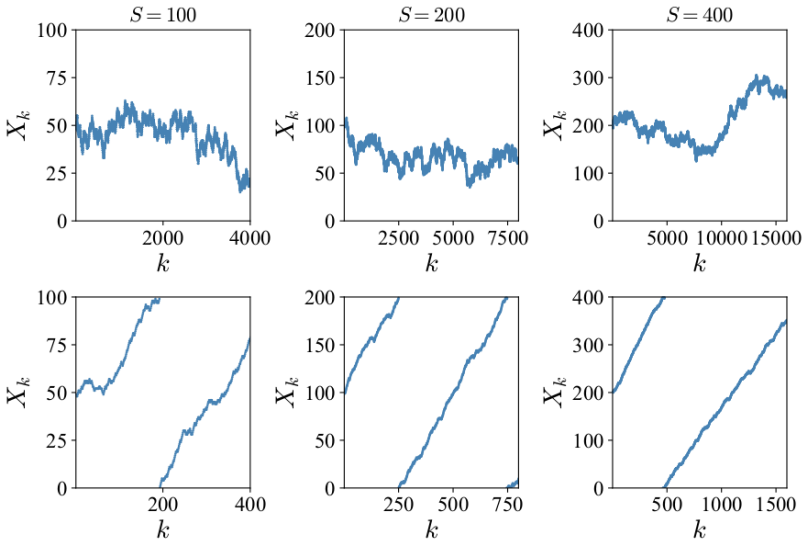

Figure 4.1 Trace plots of two MCMC algorithms for sampling from a uniform distribution on $\{0, \ldots, S-1\}$ for different values of S. A random walk of step size uniform on $\{-1, 0, 1\}$ (Example 4.1, top), and a non-reversible (biased) random walk which is twice as likely to have a step of 1 than a step of -1 (Example 4.2, bottom). Columns are for $S = 100$ (left), $S = 200$ (middle) and $S = 400$ (right). We show the state of the MCMC algorithm for $40S$ iterations (top row) or $4S$ iterations (bottom row). For this scaling of iterations, we see the reversible MCMC algorithm mixes more slowly as S increases, whereas qualitatively the mixing of the non-reversible algorithm remains similar.

to the non-reversibility of the chain, as a realisation of the chain forward in time will now look very different from a realisation backwards in time. Furthermore, the realisations for different S look similar. That is, once we scale the number of iterations by S, chains with different S mix similarly. This is shown quantitatively in the bottom left plot of Figure 4.2, where we plot the total variation distance of the empirical marginal distribution of our chain (4.1) from the uniform distribution, against n/S: this is almost identical for the three values of S.

Why does the non-reversible chain have better mixing properties? Intuitively, the poor performance of the reversible chain is because it has random-walk behaviour: it will often move up on one iteration and then move down on the next. The non-reversible chain suppresses this random-

walk behaviour as its bias means it will tend to move in the same direction. The qualitative difference between these reversible and non-reversible chains that we have demonstrated empirically, can be shown theoretically (Diaconis et al., 2000).

For a simple heuristic of this behaviour, consider $X_0 = \lfloor S/2 \rfloor$ and $n \leq \lfloor S/2 \rfloor$. For the symmetric random walk, $\mathbb{E}[X_n - X_0] = 0$ and, since the total movement by iteration n is a sum of n independent moves, $\text{Var}[X_n - X_0] \propto n$, so the typical amount of movement in the first n iterations is proportional to $\sqrt{n}$. However, for the biased random walk, $\mathbb{E}[X_n - X_0] \propto n$ and so the amount of movement is roughly proportional to n.

Importantly, when we measured the performance of the non-reversible Markov chain we looked at the accuracy of (4.1), which is the proportion of time it spends in each state averaged over time. If, instead, we look at the convergence of $\mathbb{P}(X_n = i)$ to $1/S$, we would obtain a very different result to that shown in Figure 4.2, as the bias of the random walk in Example 4.2 means that the centre of the distribution of X_n changes with n: the benefit of the non-reversible chain is only realised as we take the ergodic average over different time-points. This is most easily seen for the extreme case where $J_k = 1$ with probability 1. In that case, we find that the distribution of X_n is a point mass at a single value for each n, but by averaging over time we still have that (4.1) converges to the uniform distribution at a rate of $1/n$.

Finally, as an aside, we observe that the poor performance of the reversible chain is also linked to the fact that the size of the moves at each iteration is small – this can be quantified in terms of the variance of $X_k - X_{k-1}$ relative to the variance of the stationary distribution. And it is the fact that this ratio increased as we increased S that meant that the reversible chain performed relatively poorly for larger S. To see this we can implement the reversible random walk, but set the maximum step size $h = S/100$ so it is proportional to S. The total variation distance between (4.1) and the uniform distribution for such a chain is shown in the bottom right plot of Figure 4.2, and demonstrates better scaling with S: in fact, like the non-reversible chain, as S increases we now obtain the same accuracy providing we scale n to be proportional to S.

4.2 Hamiltonian Monte Carlo Revisited

The Hamiltonian Monte Carlo, or HMC, algorithm of Section 2.2 is often viewed as a non-reversible MCMC algorithm. However, strictly it is a reversible algorithm.

Remember that the HMC algorithm for sampling from a density $\pi(\boldsymbol{\theta})$,

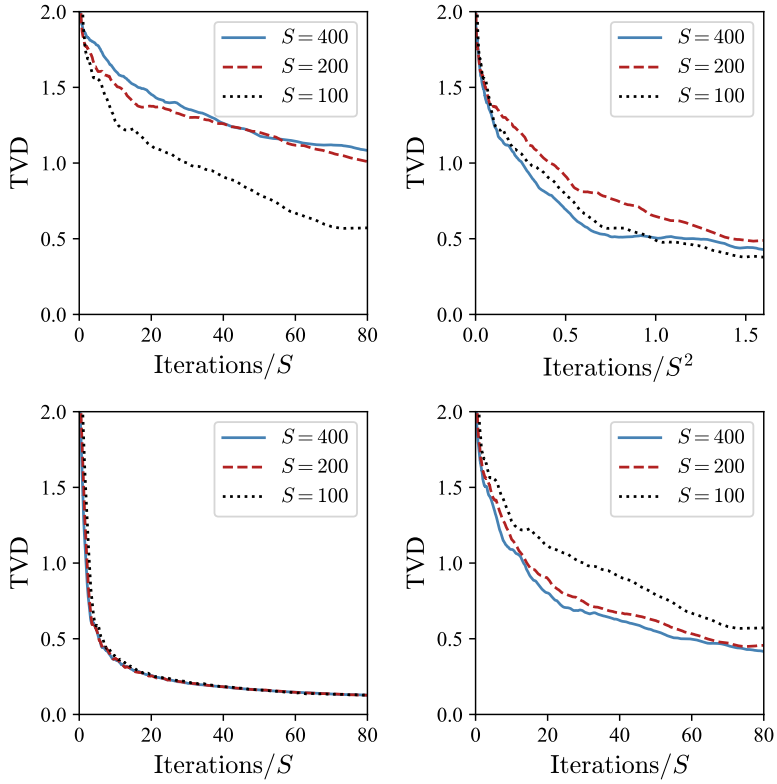

Figure 4.2 Empirical total variation distance (TVD) between (4.1) and the uniform distribution against the number of iterations and different values of S: $S = 100$ (blue), $S = 200$ (red) and $S = 400$ (black). TVD for the random walk of Example 4.1 with step size 1 (top row) with x-axis scaled by S (top left) or S^2 (top right). TVD for the non-reversible Example 4.2 (bottom left), and for the Example 4.1 with step size scaled by S (bottom right); in both case the x-axis is scaled by S. All results are averaged over 10 realisations of the chains.

involves augmenting the state of our MCMC algorithm with a momentum variable of the same dimension as $\boldsymbol{\theta}$. Denote the momentum variable by $\mathbf{p}$, and the state of our Markov chain by $(\boldsymbol{\theta}, \mathbf{p})$. In the following, for simplicity, we will assume that the mass matrix is the identity. We introduce a target

density for $(\boldsymbol{\theta}, \mathbf{p})$,

$$\tilde{\pi}(\boldsymbol{\theta}, \mathbf{p}) \propto \pi(\boldsymbol{\theta}) \exp\left\{-\frac{1}{2}\mathbf{p}^\top \mathbf{p}\right\},$$

which has independent components, with $\boldsymbol{\theta}$ from π and $\mathbf{p}$ having a standard Gaussian distribution. An HMC algorithm which has $\tilde{\pi}(\boldsymbol{\theta}, \mathbf{p})$ as its stationary density interleaves the following three moves at each iteration. Given current state $(\boldsymbol{\theta}_k, \mathbf{p}_k)$

(1) Simulate a new momentum $\mathbf{p}$ from a standard Gaussian distribution.
(2)(i) Obtain the proposed state $(\boldsymbol{\theta}', \mathbf{p}')$ by running the leapfrog dynamics, for some number, L, of steps with some step length, ϵ, starting from $(\boldsymbol{\theta}_k, \mathbf{p})$, and flip the final momentum.
(ii) Accept the proposed state, $(\boldsymbol{\theta}_{k+1}, \mathbf{p}'') = (\boldsymbol{\theta}', \mathbf{p}')$ with probability

$$\min\left(1, \frac{\pi(\boldsymbol{\theta}', \mathbf{p}')}{\pi(\boldsymbol{\theta}_k, \mathbf{p})}\right),$$

otherwise $(\boldsymbol{\theta}_{k+1}, \mathbf{p}'') = (\boldsymbol{\theta}_k, \mathbf{p})$.
(3) Flip the momentum, $(\boldsymbol{\theta}_{k+1}, \mathbf{p}_{k+1}) = (\boldsymbol{\theta}_k, -\mathbf{p}'')$.

In the notation of Section 2.2, the proposal in step (2i) is denoted by $\mathsf{Leap}_{-}^{L}(\boldsymbol{\theta}_k, \mathbf{p})$. Move (2) involves simulating non-reversible Hamiltonian dynamics to produce a potentially large proposed move, but the move itself is reversible. As, trivially, are moves (1) and (3). The move (3) has no net effect on the algorithm since the momentum is discarded at the next iteration, and it is not included in the description in Section 2.2. It does, however, ensure that, if the proposal is accepted, the final momentum is the same as that which was obtained through the leapfrog approximation to the Hamiltonian dynamics rather than its opposite. This will be helpful in the next section.

Interleaving three different reversible moves does not necessarily result in a reversible Markov chain (see the next section, for example). But in this case, it is straightforward to show that the marginal process for $\boldsymbol{\theta}$ is a reversible Markov chain. The benefit of HMC is in the use of the non-reversible deterministic leap-frog dynamics to produce large proposed moves that have a high chance of being accepted. As we saw in the previous section, the issues with reversible MCMC algorithms occur when the step size is small – and HMC gets around this by being able to propose large moves rather than being non-reversible. This fact is seen in the scaling results for HMC as the dimension of the state-space increases (see Section 2.2 and the literature in Section 2.3).

4.3 Lifting Schemes for MCMC

Whilst HMC is a reversible algorithm when viewed as a Markov chain on $\boldsymbol{\theta}$, some of the ideas behind the Hamiltonian dynamics are common to non-reversible MCMC algorithms. Furthermore, it is possible to adapt the HMC algorithm so that it is non-reversible.

Most non-reversible MCMC algorithms involve the idea of "lifting", that is, defining the Markov chain on a higher-dimensional state space than you wish to sample from. Moreover, they tend to do this in a way similar to HMC, in that if you wish to sample from some target distribution, $\pi(\boldsymbol{\theta})$, you work with a Markov chain with state $(\boldsymbol{\theta}, \mathbf{p})$, where $\mathbf{p}$ is of the same dimension, d, as $\boldsymbol{\theta}$ and can be interpreted as the momentum or velocity that is describing the direction and speed with which the Markov chain is currently moving through $\boldsymbol{\theta}$ space. The non-reversibility of the algorithm is based on trying to encourage Markov chain moves that continue in roughly the same direction over successive iterations.

4.3.1 Non-Reversible HMC

One of the first truly non-reversible algorithms is an extension of HMC due to Horowitz (1991). The idea is to adapt how the momentum is refreshed at each iteration so that the momentum at the current iteration will be similar to that at the previous iteration. This can be achieved by replacing step (1) of the HMC algorithm with the update

$$\mathbf{p}' = \gamma \mathbf{p} + (1 - \gamma^2)^{1/2} \boldsymbol{\zeta},$$

where $\boldsymbol{\zeta}$ is a realisation of a d-dimensional standard Gaussian random variable, and $0 < \gamma < 1$. If $\mathbf{p}$ has a standard Gaussian distribution, then so does $\mathbf{p}'$: it is Gaussian as it is a linear combination of two independent Gaussian random variables, the expectation of the right-hand side is 0, and the variance is $\gamma^2 \mathbf{I}_d + (1 - \gamma^2)\mathbf{I}_d = \mathbf{I}_d$. If γ is close to 1 then $\mathbf{p}'$ will be close to $\mathbf{p}$. The overall algorithm is given in Algorithm 4.

This algorithm has the required stationary distribution, as each step satisfies detailed balance with respect to the extended posterior, $\widetilde{\pi}$. However, whilst each step of the algorithm is a reversible Markov chain move, by interleaving the moves we obtain a non-reversible algorithm.

Whilst this algorithm generalises HMC, in practice it is rarely noticeably more efficient (see for example Neal, 2011). The reason is the momentum flip that occurs if we reject our proposal, as, if γ is large this sends the chain back in the direction from whence it came. Thus we need to either choose the

Algorithm 4: Non-reversible HMC of Horowitz (1991)

Input: Density $\pi(\boldsymbol{\theta})$, initial value $\boldsymbol{\theta}_0$ and momentum $\mathbf{p}_0$ sampled from $\mathsf{N}(\mathbf{0}, \mathbf{I}_d)$, and refresh rate $\gamma \in [0, 1)$.

for $k \in 0, \ldots, n-1$ **do**

Simulate $\boldsymbol{\zeta} \sim \mathsf{N}(\mathbf{0}, \mathbf{I}_d)$ and set $\mathbf{p}' = \gamma \mathbf{p}_k + (1-\gamma^2)^{1/2}\boldsymbol{\zeta}$.

$(\boldsymbol{\theta}', \mathbf{p}'') \leftarrow \mathsf{Leap}_{-}^{L}(\boldsymbol{\theta}_k, \mathbf{p}')$.

Calculate the acceptance probability:

$$\alpha(\boldsymbol{\theta}_k, \mathbf{p}'; \boldsymbol{\theta}', \mathbf{p}'') := \min\left(1, \frac{\widetilde{\pi}(\boldsymbol{\theta}', \mathbf{p}'')}{\widetilde{\pi}(\boldsymbol{\theta}_k, \mathbf{p}')}\right).$$

With a probability of $\alpha(\boldsymbol{\theta}_k, \mathbf{p}'; \boldsymbol{\theta}', \mathbf{p}'')$ accept the proposal, $(\boldsymbol{\theta}_{k+1}, \mathbf{p}''') \leftarrow (\boldsymbol{\theta}', \mathbf{p}'')$; otherwise reject it, $(\boldsymbol{\theta}_{k+1}, \mathbf{p}''') \leftarrow (\boldsymbol{\theta}_k, \mathbf{p}_k)$.

Flip the velocity: $(\boldsymbol{\theta}_{k+1}, \mathbf{p}_{k+1}) \leftarrow (\boldsymbol{\theta}_{k+1}, -\mathbf{p}''')$.

end

leapfrog step size, ϵ, to be small enough that the probability of acceptance after L steps is usually very close to 1, or we need to choose γ to be small so that the new momentum is relatively unrelated to the old momentum. The first comes at a high computational cost while the second leads to an algorithm that is very similar to standard HMC (which corresponds to $\gamma = 0$). Later, in Section 4.4, we will present some alternative ideas that can alleviate the problems of momentum flips in algorithms such as this.

4.3.2 Gustafson's Algorithm and Multidimensional Generalisations

It is possible to use similar ideas to obtain non-reversible versions of a random walk Metropolis–Hastings algorithm, known as *guided random walks*. This was first proposed by Gustafson (1998) for component-wise updates, though it has a natural extension to multivariate updates which we will also describe. First, consider the univariate case and a Gaussian random walk proposal with a variance of δ^2 and $\zeta \sim \mathsf{N}(0, \delta^2)$. Given a current value, θ, we can write this proposal as

$$\theta' = \theta + p|\zeta|,$$

where p is uniformly sampled from $\{-1, 1\}$. We can think of p as the direction of the move and $|\zeta|$ as the size of the move. The idea of Gustafson (1998) is to augment the state of the chain with p and to simulate a chain

such that the direction of the move is likely to be the same over successive time steps. This can be achieved by iterating the following two steps

(1) Simulate ζ, a realisation of a Gaussian random variable, and set $(\theta', p') = (\theta_k + p_k|\zeta|, -p_k)$ with probability

$$\min\left\{1, \frac{\pi(\theta')}{\pi(\theta_k)}\right\}$$

otherwise set $(\theta', p') = (\theta_k, p_k)$.

(2) Flip the direction, so $(\theta_{k+1}, p_{k+1}) = (\theta', -p')$.

The stationary distribution of this algorithm has independent distributions for θ and p, with θ from the distribution whose density is proportional to $\pi(\theta)$, and p having a uniform distribution on $\{-1, 1\}$. This follows as both steps (1) and (2) are reversible moves that keep such a distribution invariant.

To see the behaviour of this algorithm, and compare it to a standard random walk Metropolis–Hastings algorithm, we show results of both algorithms when sampling from a Gaussian target distribution in Figures 4.3 and 4.4. For Figure 4.3, we implement both algorithms using a small proposal variance, δ^2. Here we see the poor mixing of the random walk Metropolis due to its reversibility and the random walk behaviour of its output. This means that it takes nearly 1000 steps to reach the mode of the target distribution. By comparison, Gustafson's algorithm suppresses this random walk behaviour, with large periods of time spent moving in the same direction. As we are sampling from a uni-modal target, the qualitative dynamics of the algorithm are as follows: when it is moving towards the mode the acceptance probability is 1 and the algorithm keeps moving in that direction. It is only when it is moving away from the mode that it has any chance of rejecting a proposal and switching the direction of its move.

The behaviour of the two algorithms when we use a larger step size, as shown in Figure 4.4, is very different. In this case, both trace plots look qualitatively similar, and both samplers have similarly good performance as shown by the auto-correlation plots. The reason is that for a large step size, the chances of rejecting the proposal and switching the direction are now much higher, closer to 0.5 on average. This also ties in with the observation from Section 4.1 that reversibility is only an issue when the step sizes are small.

If we wish to sample from a multivariate target $\pi(\boldsymbol{\theta})$, we can do so by applying this update component-wise. That is, we augment the state to $(\boldsymbol{\theta}, \mathbf{p})$ where $\mathbf{p}$ is d-dimensional and each entry of $\mathbf{p}$ is either 1 or -1 and specifies the direction of the next update of the corresponding component

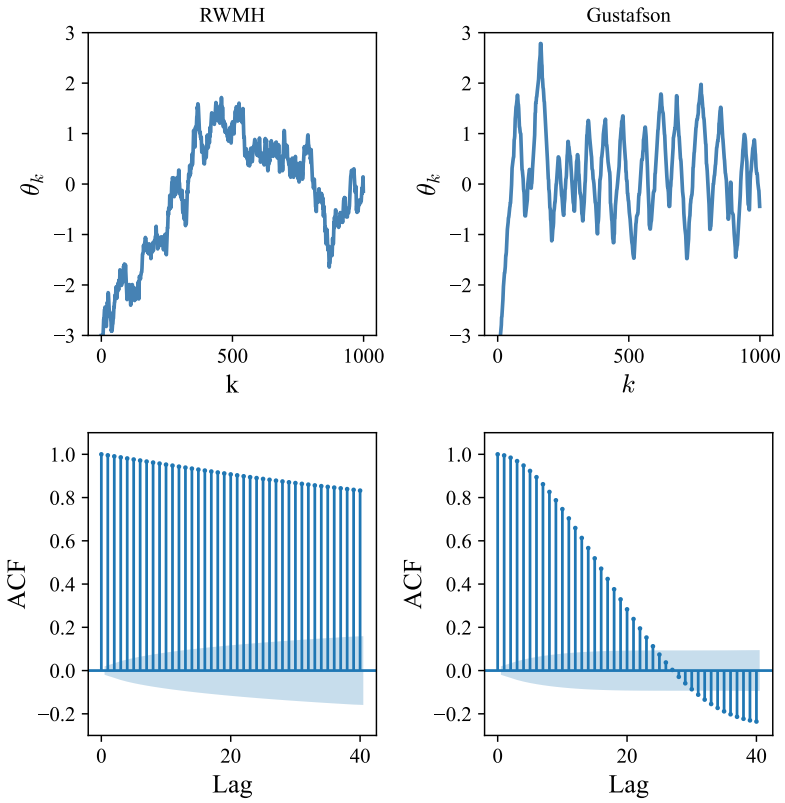

Figure 4.3 Comparison of random walk Metropolis–Hastings (left column) and the guided random walk (right column) when sampling from a 1-d Gaussian. The proposal is Gaussian with a standard deviation of 0.1 in both cases. Top row shows trace plots, and bottom row shows the estimated auto-correlation of the chains.

of $\boldsymbol{\theta}$. Then we have d parts to each iteration of the algorithm where each part uses the above algorithm to update a different component of $\boldsymbol{\theta}$.

One can see the similarity with the algorithm of Horowitz (1991). We have augmented the state to include a component, of the same dimension as $\boldsymbol{\theta}$, that governs the direction of the update of the Markov chain. Our Markov chain update is based on interleaving two reversible moves, but with the resulting Markov chain being non-reversible. Finally, each reversible move involves a flip of direction; providing the acceptance probability in step (1)

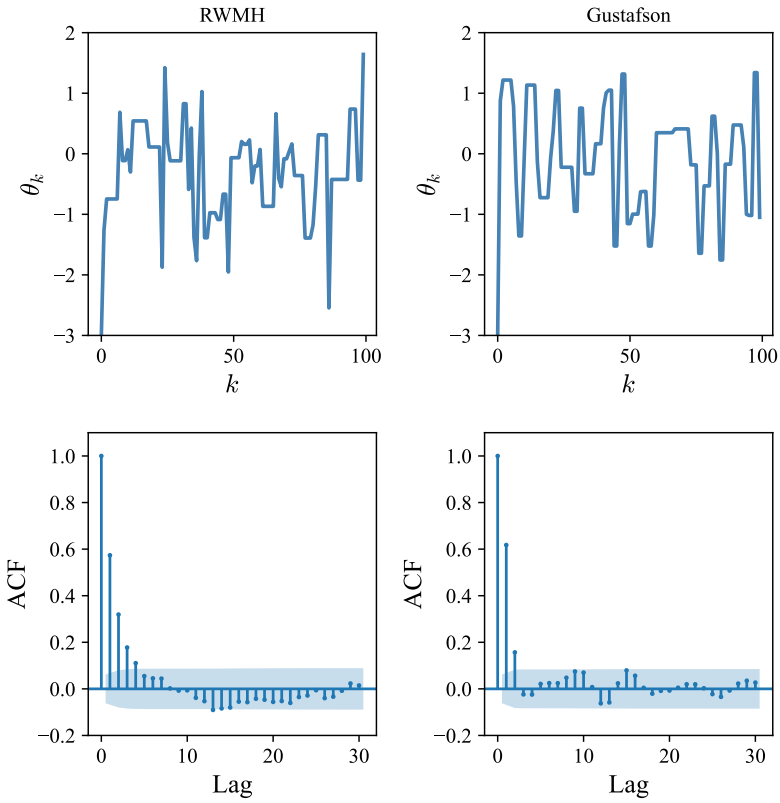

Figure 4.4 Comparison of random walk Metropolis–Hastings (left column) and the guided random walk algorithm (right column) when sampling from a 1-d Gaussian. The proposal is Gaussian with a standard deviation of 2.38 in both cases. Top row shows trace plots, and bottom row shows the estimated auto-correlation of the chains.

is high, then these cancel out and the chain is likely to move in the same direction over multiple time-steps.

Finally, one can implement the idea of Gustafson (1998) in a way that jointly updates the d-dimensional state. This can be achieved by letting $\mathbf{p}$ be a d-dimensional unit vector. Define a target distribution on $(\boldsymbol{\theta}, \mathbf{p})$ that is the product of $\pi(\boldsymbol{\theta})$ and the uniform density for $\mathbf{p}$ on the d-dimensional sphere; we denote the density by $\mathsf{U}_d(\mathbf{p})$ and the surface of the sphere as $\mathcal{S}_d$. This target will be kept invariant by the following algorithm, where to

aid the presentation of the algorithm in Section 4.4 we replace the random $|\zeta|$ with a fixed, user-prescribed $\delta > 0$. Here and for the remainder of this chapter the proposal is a deterministic, 1-1 map, rather than a density, and we use the symbol q rather than q.

Algorithm 5: Multi-dimensional guided random walk, with fixed direction.

Input: Density $\pi(\boldsymbol{\theta})$, initial value $\boldsymbol{\theta}_0$ and speed $\delta > 0$, unit vector $\mathbf{p}_0$ sampled from U_d.

for $k \in 0, \ldots, n-1$ **do**

Propose $(\boldsymbol{\theta}', \mathbf{p}') = \mathsf{q}_1(\boldsymbol{\theta}_k, \mathbf{p}_k) = (\boldsymbol{\theta}_k + \delta \mathbf{p}_k, -\mathbf{p}_k)$.

With probability

$$\alpha_1(\boldsymbol{\theta}_k, \mathbf{p}_k; \boldsymbol{\theta}', \mathbf{p}') := \min\left\{1, \frac{\pi(\boldsymbol{\theta}')}{\pi(\boldsymbol{\theta}_k)}\right\}$$

accept the proposal, $(\boldsymbol{\theta}_{k+1}, \mathbf{p}'') \leftarrow (\boldsymbol{\theta}', \mathbf{p}')$; otherwise reject it, $(\boldsymbol{\theta}_{k+1}, \mathbf{p}'') \leftarrow (\boldsymbol{\theta}_k, \mathbf{p}_k)$.

Flip the velocity: $(\boldsymbol{\theta}_{k+1}, \mathbf{p}_{k+1}) = (\boldsymbol{\theta}_{k+1}, -\mathbf{p}'')$.

end

The velocity flip does not update $\boldsymbol{\theta}$, and $\mathbf{p} \in \mathcal{S}_d \Leftrightarrow -\mathbf{p} \in \mathcal{S}_d$, so the flip is reversible with respect to $\pi(\boldsymbol{\theta})\mathsf{U}_d(\mathbf{p})$, which must, therefore, be the stationary density. It is helpful to explore exactly why the first step is also reversible with respect to this density. Suppose that $(\boldsymbol{\theta}, \mathbf{p})$ has a density of $\pi(\boldsymbol{\theta})\mathsf{U}_d(\mathbf{p})$ and let $\mathcal{B}$ and C be disjoint sets in $\mathbb{R}^d \times \mathcal{S}_d$. Then $\mathbb{P}\left((\boldsymbol{\theta}, \mathbf{p}) \in \mathcal{B}, (\boldsymbol{\theta}', \mathbf{p}') \in C\right)$ is

$$\begin{aligned}
&\int \pi(\boldsymbol{\theta})\mathsf{U}_d(\mathbf{p}) \min\left(1, \frac{\pi(\boldsymbol{\theta}')}{\pi(\boldsymbol{\theta})}\right) 1_{\mathcal{B}}(\boldsymbol{\theta}, \mathbf{p}) 1_C(\boldsymbol{\theta}', \mathbf{p}') \, \mathrm{d}\boldsymbol{\theta}\mathrm{d}\mathbf{p} \\
&= \int \pi(\boldsymbol{\theta}')\mathsf{U}_d(\mathbf{p}') \min\left(1, \frac{\pi(\boldsymbol{\theta})}{\pi(\boldsymbol{\theta}')}\right) 1_{\mathcal{B}}(\boldsymbol{\theta}, \mathbf{p}) 1_C(\boldsymbol{\theta}', \mathbf{p}') \, \mathrm{d}\boldsymbol{\theta}\mathrm{d}\mathbf{p} \\
&= \int \pi(\boldsymbol{\theta}')\mathsf{U}_d(\mathbf{p}') \min\left(1, \frac{\pi(\boldsymbol{\theta})}{\pi(\boldsymbol{\theta}')}\right) 1_{\mathcal{B}}(\boldsymbol{\theta}, \mathbf{p}) 1_C(\boldsymbol{\theta}', \mathbf{p}') \, \mathrm{d}\boldsymbol{\theta}'\mathrm{d}\mathbf{p}',
\end{aligned}$$

as required. Here, the second line follows by rearrangement and because the algorithm forces $\mathbf{p} \in \mathcal{S}_d \Leftrightarrow \mathbf{p}' \in \mathcal{S}_d$. The third line follows from the unit Jacobian of the transformation q_1.

While this algorithm keeps the target on $(\boldsymbol{\theta}, \mathbf{p})$ invariant, the algorithm is reducible; $\mathbf{p}$ can only take two values: $\mathbf{p}_0$ and $-\mathbf{p}_0$, whilst $\boldsymbol{\theta}$ is confined to a discrete grid on the vector $\boldsymbol{\theta}_0 + \delta\mathbf{p}_0$. It is straightforward to make the

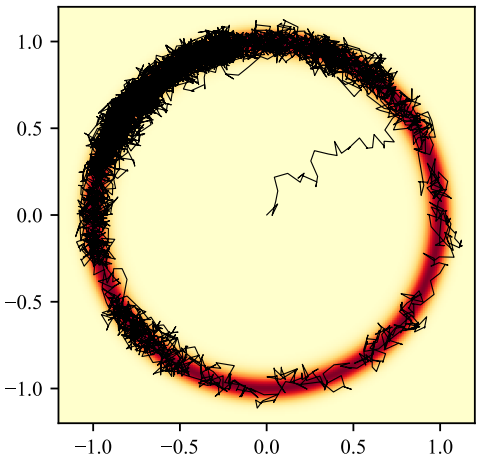

Figure 4.5 Trace plot for 50000 iterations of the algorithm of Gustafson (1998) sampling from a 2-d density concentrated on the unit circle. The heat map shows the target density (red is the region of high density), and the black line shows the trace of the algorithm.

algorithm irreducible in dimension 2 and above by adding occasional moves that update $\mathbf{p}$, for example, that sample a new value of $\mathbf{p}$ from U_d.

Comparison on a Ring-shaped Target

To see the benefits of these non-reversible algorithms, we compared the guided random walk algorithm with that of a standard random walk Metropolis algorithm for sampling from a density that concentrates on the perimeter of a circle in 2-d. This mimics a common challenge of sampling from a density that concentrates near a lower-dimensional manifold within a higher-dimensional space. We implemented Algorithm 5 with a refresh of the direction every 10 iterations.

Trace plots for the non-reversible and reversible algorithms are shown in Figures 4.5 and 4.6, respectively. Each algorithm uses the same average step size. For good mixing in the manifold scenario, the step size needs to be of the order of the width of the density orthogonal to the circle – and thus is small relative to the size of the circle itself. As we have seen

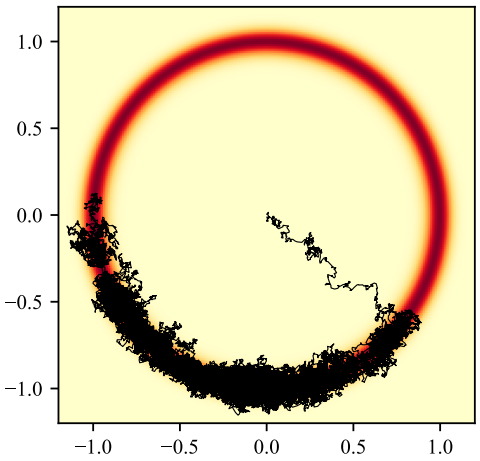

Figure 4.6 Trace plot for 50000 iterations of the random-walk Metropolis algorithm sampling from a 2-*d* density concentrated on the unit circle. The heat map shows the target density (red is the region of high density), and the black line is the trace of the algorithm.

previously, using such a small step size leads to random-walk behaviour for the reversible algorithm. By comparison, the non-reversible algorithm is able to suppress this. The overall effect is much better mixing for the non-reversible algorithm, which takes 50000 iterations to explore the full extent of the ring. By comparison, over the same number of iterations, the reversible algorithm has only explored the bottom half of the ring; in fact, it takes six times this number of iterations to explore the entire ring.

4.4 Improving Non-reversibility: Delayed Rejection

The major source of inefficiency in Algorithm 5 is the net reversal of momentum whenever the proposed new position and momentum $(\boldsymbol{\theta}', \mathbf{p}')$ are rejected. The subsequent momentum flip is designed to keep the process moving in the same direction as it was at the start of the iteration; however, it has the opposite effect when there is a rejection, causing the chain to retrace its steps. The *delayed-rejection* method of Green and Mira (2001)

can be applied to any reversible step and provides a natural mechanism for reducing the probability of rejection as follows: whenever the original step would have rejected the proposal, a new value is proposed; the acceptance probability for this new proposal is designed exactly so that the combination of the rejected existing step and the potential new step satisfies detailed balance with respect to the intended posterior.

The validity of the propose-accept/reject step of Algorithm 5 relies on the Jacobian of q_1 being 1 and the fact that $\mathsf{q}_1(\boldsymbol{\theta}', \mathbf{p}') \equiv \mathsf{q}_1(\boldsymbol{\theta}_k + \delta\mathbf{p}_k, -\mathbf{p}_k) = (\boldsymbol{\theta}_k, \mathbf{p}_k)$. Following analogous constraints, we incorporate a delayed-rejection move as follows: If $(\boldsymbol{\theta}', \mathbf{p}')$ is rejected, we can make a *new* proposal based not only on the current state, $(\boldsymbol{\theta}_k, \mathbf{p}_k)$ but also on the initial, rejected proposal, $(\boldsymbol{\theta}', \mathbf{p}')$. In this case, we now consider the current state to be the original current state and the initial, rejected proposal: $(\boldsymbol{\theta}_k, \mathbf{p}_k, \boldsymbol{\theta}', \mathbf{p}')$; we call this the *full state*. We then make a proposal for this full state; *i.e.*, we propose a new original state and a new initial, rejected proposal: $(\boldsymbol{\theta}'', \mathbf{p}'', \boldsymbol{\theta}''', \mathbf{p}''') = \mathsf{q}_2(\boldsymbol{\theta}_k, \mathbf{p}_k, \boldsymbol{\theta}', \mathbf{p}')$, where $(\boldsymbol{\theta}''', \mathbf{p}''') = (\boldsymbol{\theta}'' + \delta\mathbf{p}'', -\mathbf{p}'')$. To simplify the notation in the following, we define

$$\mathbf{z}_k = (\boldsymbol{\theta}_k, \mathbf{p}_k), \quad \mathbf{z}' = (\boldsymbol{\theta}', \mathbf{p}'), \quad \mathbf{z}'' = (\boldsymbol{\theta}'', \mathbf{p}'') \text{ and } \mathbf{z}''' = (\boldsymbol{\theta}''', \mathbf{p}''').$$

We will either accept or reject the proposal $(\mathbf{z}'', \mathbf{z}''') = \mathsf{q}_2(\mathbf{z}_k, \mathbf{z}')$ for the full state and we choose the probability of acceptance exactly so that the move satisfies detailed balance with respect to the density of the current state (which implies the initial proposal), including the fact that it was rejected:

$$\tilde{\tilde{\pi}}(\mathbf{z}_k, \mathbf{z}') := \pi(\boldsymbol{\theta}_k)\mathsf{U}_d(\mathbf{p}_k)\mathbb{I}\left[\mathbf{z}' = \mathsf{q}_1(\mathbf{z}_k)\right]\left\{1 - \alpha_1(\mathbf{z}_k; \mathbf{z}')\right\}.$$

By analogy with the Metropolis–Hastings algorithm we might expect this acceptance probability to be

$$\alpha_2(\mathbf{z}_k, \mathbf{z}'; \mathbf{z}'', \mathbf{z}''') := \min\left[1, \frac{\{1 - \alpha_1(\mathbf{z}''; \mathbf{z}''')\}\pi(\boldsymbol{\theta}'')}{\{1 - \alpha_1(\mathbf{z}_k; \mathbf{z}')\}\pi(\boldsymbol{\theta}_k)}\right],$$

where for simplicity of presentation we have not included the indicator functions $\mathbb{I}\left[\mathbf{z}''' = \mathsf{q}_1(\mathbf{z}'')\right]$ and $\mathbb{I}\left[\mathbf{z}' = \mathsf{q}_1(\mathbf{z}_k)\right]$ in the numerator and denominator of the fraction, respectively, since by design these are both 1. Indeed, subject to conditions on q_2, α_2 is correct. For the proposal and accept/reject step to be valid, q_2 must have a Jacobian of 1, and must satisfy $\mathsf{q}_2(\mathbf{z}'', \mathbf{z}''') = (\mathbf{z}_k, \mathbf{z}')$. The probability of being at $\mathbf{z}_k$ with a deterministic proposal of $\mathbf{z}'$ that has been rejected, and then moving to this new full state is, therefore,

$$\tilde{\tilde{\pi}}(\mathbf{z}_k, \mathbf{z}')\alpha_2(\mathbf{z}_k\mathbf{z}'; \mathbf{z}'', \mathbf{z}'''),$$

which, by design, is equal to

$$\tilde{\tilde{\pi}}(\mathbf{z}'', \mathbf{z}''')\alpha_2(\mathbf{z}'', \mathbf{z}'''; \mathbf{z}_k, \mathbf{z}').$$

Using the two constraints on q_2 and an analogous argument to that used for Algorithm 5, detailed balance is, therefore, satisfied.

4.4.1 The Discrete Bouncy Particle Sampler

The current state is $\mathbf{z}_k$, given this, $\mathbf{z}'$ is fixed via q_1. The 1-1 map q_1 similarly fixes the relationship between $\mathbf{z}''$ and $\mathbf{z}'''$ so the additional flexibility this delayed-rejection formulation allows is the freedom to choose $\mathbf{z}''$ or, equivalently, $\mathbf{z}'''$.

To make α_2 close to 1, a sensible aim is to choose $(\mathbf{z}'', \mathbf{z}''')$ such that $\pi(\boldsymbol{\theta}'') \approx \pi(\boldsymbol{\theta}_k)$ and $\alpha_1(\mathbf{z}''; \mathbf{z}''') \approx \alpha_1(\mathbf{z}_k; \mathbf{z}')$. If we set $\boldsymbol{\theta}''' = \boldsymbol{\theta}'$, both of these conditions will be satisfied if $\pi(\boldsymbol{\theta}')/\pi(\boldsymbol{\theta}) \approx \pi(\boldsymbol{\theta}')/\pi(\boldsymbol{\theta}'')$; *i.e.*

$$\log \pi(\boldsymbol{\theta}') - \log \pi(\boldsymbol{\theta}'') \approx \log \pi(\boldsymbol{\theta}') - \log \pi(\boldsymbol{\theta}_k).$$

Taylor expanding about $\boldsymbol{\theta}'$, we require $\delta\mathbf{g}\cdot\mathbf{p} \approx \delta\mathbf{g}\cdot\mathbf{p}''$, where $\mathbf{g} = \nabla \log \pi(\boldsymbol{\theta}')$; *i.e.* both $\mathbf{p}''$ and $\mathbf{p}$ should have approximately the same component in the gradient direction. Perhaps the most natural way to achieve this is by setting

$$\mathbf{p}'' = \Psi_{\mathbf{g}}(\mathbf{p}) := -\mathbf{p}_k + 2(\mathbf{p}_k \cdot \widehat{\mathbf{g}})\, \widehat{\mathbf{g}},$$

where $\widehat{\mathbf{g}} = \mathbf{g}/\|\mathbf{g}\|$, is the direction of the gradient of $\log \pi$ at $\boldsymbol{\theta}'$. In this case,

$$\Psi_{\mathbf{g}}(\mathbf{p}'') = \mathbf{p}_k - 2(\mathbf{p}_k \cdot \widehat{\mathbf{g}})\, \widehat{\mathbf{g}} + 2[\{-\mathbf{p}_k + 2(\mathbf{p}_k \cdot \widehat{\mathbf{g}})\, \widehat{\mathbf{g}}\} \cdot \widehat{\mathbf{g}}]\, \widehat{\mathbf{g}} = \mathbf{p}_k.$$

Further, since $\Psi_{\mathbf{g}}(\cdot)$ is self-inverse, the absolute value of its Jacobian must be 1. The full proposal is, therefore,

$$(\mathbf{z}'', \mathbf{z}''') = \mathsf{q}_2(\mathbf{z}_k, \mathbf{z}') = \left(\boldsymbol{\theta}' - \Psi_{\mathbf{g}}(\mathbf{p}_k), \Psi_{\mathbf{g}}(\mathbf{p}_k), \boldsymbol{\theta}', -\Psi_{\mathbf{g}}(\mathbf{p}_k)\right).$$

However, since $\mathbf{z}'''$ plays no part in any subsequent movement and since momenta do not figure in the acceptance probabilities, it can simplify notation to think of the proposal as

$$(\boldsymbol{\theta}'', \mathbf{p}'') = \mathsf{q}_2^*(\boldsymbol{\theta}_k, \mathbf{p}_k, \boldsymbol{\theta}') := \left(\boldsymbol{\theta}' - \Psi_{\mathbf{g}}(\mathbf{p}_k), \Psi_{\mathbf{g}}(\mathbf{p}_k)\right),$$

the initial acceptance probability as

$$\alpha_1^*(\boldsymbol{\theta}_k, \boldsymbol{\theta}') := \min\left\{1, \frac{\pi(\boldsymbol{\theta}')}{\pi(\boldsymbol{\theta}_k)}\right\},$$

and the second acceptance probability as

$$\alpha_2^*(\boldsymbol{\theta}, \boldsymbol{\theta}', \boldsymbol{\theta}'') = \min\left[1, \frac{\{1 - \alpha_1^*(\boldsymbol{\theta}'', \boldsymbol{\theta}')\}\pi(\boldsymbol{\theta}'')}{\{1 - \alpha_1^*(\boldsymbol{\theta}_k, \boldsymbol{\theta}')\}\pi(\boldsymbol{\theta}_k)}\right].$$

The full algorithm is given in Algorithm 6 and illustrated in Figure 4.7.

Algorithm 6: Discrete Bouncy Particle Sampler

Input: Density $\pi(\boldsymbol{\theta})$, initial value $\boldsymbol{\theta}_0$ and speed δ, unit vector $\mathbf{p}_0$ sampled from U_d.

for $t \in 0, \ldots, T-1$ **do**

Propose $(\boldsymbol{\theta}', \mathbf{p}') = \mathsf{q}_1(\boldsymbol{\theta}_k, \mathbf{p}_k)$.

With a probability of $\alpha_1^*(\boldsymbol{\theta}_k, \boldsymbol{\theta}')$ accept the proposal: $(\boldsymbol{\theta}_{k+1}, \mathbf{p}_{k+1}) \leftarrow (\boldsymbol{\theta}', \mathbf{p}')$.

If the proposal is not accepted then propose $(\boldsymbol{\theta}'', \mathbf{p}'') = \mathsf{q}_2^*(\boldsymbol{\theta}_k, \mathbf{p}_k, \boldsymbol{\theta}')$.

With a probability of $\alpha_2^*(\boldsymbol{\theta}_k, \boldsymbol{\theta}', \boldsymbol{\theta}'')$ accept this proposal: $(\boldsymbol{\theta}_{k+1}, \mathbf{p}_{k+1}) \leftarrow (\boldsymbol{\theta}'', \mathbf{p}'')$, otherwise $(\boldsymbol{\theta}_{k+1}, \mathbf{p}_{k+1}) = (\boldsymbol{\theta}_k, \mathbf{p}_k)$.

Flip the velocity: $(\boldsymbol{\theta}_{k+1}, \mathbf{p}_{k+1}) = (\boldsymbol{\theta}_{k+1}, -\mathbf{p}_{k+1})$.

end

As with Algorithm 5, we can ensure that Algorithm 6 is irreducible by refreshing the unit vector $\mathbf{p}$. This could involve sampling $\mathbf{p}_k \sim \mathsf{U}_d$ every m iterations for some integer m, or with probability p on any iteration; however, this still allows for rejections causing the sampler to exactly retrace the recent past. More usually, therefore, after every velocity flip step, the following update is made:

$$\mathbf{p}_{k+1} = \frac{\gamma \mathbf{p}_{k+1} + \sqrt{1-\gamma^2}\boldsymbol{\xi}}{\|\gamma \mathbf{p}_{k+1} + \sqrt{1-\gamma^2}\boldsymbol{\xi}\|},$$

where $\boldsymbol{\xi} \sim \mathsf{N}(\mathbf{0}, \frac{1}{d}\mathbf{I}_d)$ and $0 \leq \gamma < 1$.

Considering the combined effect of the initial proposal, the delayed-rejection step and the momentum flip on a hypothetical particle at $\boldsymbol{\theta}_k$ with a velocity of $\mathbf{p}_k$ moving along a level, frictionless surface provides a useful insight into the behaviour of Algorithm 6. In the following we define $\mathcal{R}_{\mathbf{g}}(\mathbf{p}) := -\Psi_{\mathbf{g}}(\mathbf{p}) = \mathbf{p} - 2(\mathbf{p} \cdot \widehat{\mathbf{g}})\,\widehat{\mathbf{g}}$, which is a reflection of $\mathbf{p}$ in the hyperplane perpendicular to $\mathbf{g}$.

- If the initial proposal is accepted then the net effect is $(\boldsymbol{\theta}_{k+1}, \mathbf{p}_{k+1}) = (\boldsymbol{\theta}_k + \delta\mathbf{p}_k, \mathbf{p}_k)$; the particle moves exactly as it should over a time δ.

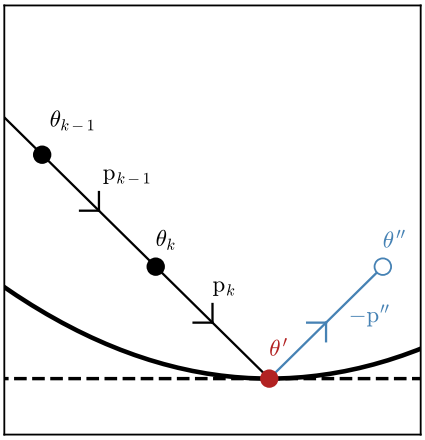

Figure 4.7 Heuristic of consecutive steps from the discrete bouncy particle sampler. From $(\boldsymbol{\theta}_{k-1}, \mathbf{p}_{k-1})$ the initial proposal is accepted and the momentum is flipped, leading to $(\boldsymbol{\theta}_k, \mathbf{p}_k)$. The initial proposal from this point ($\boldsymbol{\theta}'$ is shown but $\mathbf{p}' = -\mathbf{p}_k$ is not) is rejected. The thick solid line shows a contour of π at $\boldsymbol{\theta}'$ with the tangent line at $\boldsymbol{\theta}'$ shown dashed. The full proposal includes $(\boldsymbol{\theta}'', \mathbf{p}'')$; the figure shows $-\mathbf{p}''$ to emphasise how, if the proposal is accepted, with the subsequent momentum flipped the movement is analogous to a bounce off the tangent hyperplane.

- If the initial proposal is rejected, but the delayed-rejection proposal is accepted, then the net effect is $(\boldsymbol{\theta}_{k+1}, \mathbf{p}_{k+1}) = \big(\boldsymbol{\theta}_k + \delta\mathbf{p}_k + \delta\mathcal{R}_{\mathbf{g}}(\mathbf{p}_k), \mathcal{R}_{\mathbf{g}}(\mathbf{p}_k)\big)$; the particle moves forward for a time δ to $\boldsymbol{\theta}'$ then reflects off the hyperplane perpendicular to $\mathbf{g}$ and moves forward for another time δ.
- If both the initial step and the delayed-rejection step are rejected then $(\boldsymbol{\theta}_{k+1}, \mathbf{p}_{k+1}) = (\boldsymbol{\theta}_k, -\mathbf{p}_k)$; the particle reverses direction.

If it were not for the occasional full rejection, the path of the points outputted from the algorithm would resemble a time discretisation of the smooth path of a particle moving along a frictionless surface and occasionally bouncing off a hard barrier in the hyperplane perpendicular to the local gradient. For this reason, the algorithm is known as the *Discrete Bouncy Particle Sampler*. In the limit, as $\delta \downarrow 0$ and the number of steps is increased in proportion to $1/\delta$, this becomes a continuous-time algorithm known as the *Bouncy Particle Sampler*, which we shall meet in Section 5.3.1.

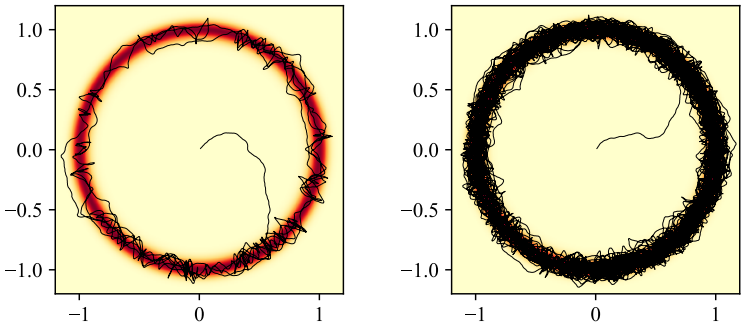

Figure 4.8 Trajectories after 5000 iterations (left) and 50000 iterations (right) of the discrete bouncy particle sampler on the two-dimensional ring target of Figures 4.5 and 4.6, and using the same step size.

Performance on the Ring-shaped Target

We exemplify the improved trajectories of the discrete bouncy particle sampler by implementing it on the same two-dimensional ring-shaped target for which the guided random walk with a directional update every 10 iterations took 50000 iterations to explore (Figure 4.5) and the random walk Metropolis with the same step size took 300000 iterations to explore (Figure 4.6 shows the first 50000 iterations).

Our discrete bouncy particle sampler uses the same step size as the earlier algorithms and sets γ so that the direction of travel is only partially forgotten from just after one bounce to just before the next (see Sherlock and Thiery, 2022). Figure 4.8 shows the path of the discrete bouncy particle sampler after 5000 (left) and 50000 (right) iterations, on the two-dimensional ring target. Exploration of the full circle takes only a tenth of the number of iterations that the guided random walk requires and a sixtieth of the number of iterations required by the RWM. The coverage after the full 50000 iterations is also more complete than when using the guided random walk.

4.5 Chapter Notes

Perhaps the earliest theoretical results showing that non-reversible chains have better mixing properties were given in Diaconis et al. (2000). The re-

sults from a pre-print of this work were extended in Chen et al. (1999). See also Neal (2004) and Sun et al. (2010) which show that non-reversible samplers reduce asymptotic variance. More recent results are given in Bierkens (2016).

The discrete bouncy particle sampler, described in Section 4.4, is given in Sherlock and Thiery (2022). Similar algorithms appear in Neal (2003) and Vanetti et al. (2017). The key difference is the use of "external bounces" rather than "internal bounces": reflection happens in the current point rather than the proposed point, so that: $(\boldsymbol{\theta}'', \mathbf{p}'', \boldsymbol{\theta}''', \mathbf{p}''') = (\boldsymbol{\theta}_k, \Psi_{\mathbf{g}}(\mathbf{p}_k), \boldsymbol{\theta}_k + \Psi_{\mathbf{g}}(\mathbf{p}_k), -\Psi_{\mathbf{g}}(\mathbf{p}_k))$, where $\mathbf{g} = \nabla \log \pi(\boldsymbol{\theta}_k)$. Use of the bounce move that is key to the success of the discrete bouncy particle sampler will be seen within the continuous-time bouncy particle sampler described in the next chapter. It has also been used in other discrete-time MCMC algorithms. For example, the Hug move in Ludkin and Sherlock (2022) uses repeated bounces in the same way that HMC uses repeated leapfrog steps, with the result that the path to the proposal stays close to the contour of π on which it started, and just as with HMC, for a given integration time, the acceptance probability can be made as close to 1 as desired by reducing the step size.

An alternative to the lifting schemes described in this chapter are methods that adapt a reversible Markov chain to a non-reversible one without adding additional states. The general approach is to find a loop of states, and then adapt the probability flow around this loop such that the net flow of probability at each state is preserved. For example, if we have states i, j and k then we can reduce each of the three probabilities of moving from i to i, j to j and k to k by the minimum of the three and increase each of the three probability transitions from i to j, j to k and k to i by the same amount, so that the invariant distribution of the chain is unchanged. Such changes have been described as adding *vortices*. These ideas are described in Sun et al. (2010) and Turitsyn et al. (2011). See Suwa and Todo (2010) for related ideas. These constructions are hard to adapt to general Markov chains, particularly chains without discrete states, though see Bierkens (2016) for an approach to adapt the Metropolis–Hastings acceptance probability to introduce non-reversibility.

5

Continuous-Time MCMC

The previous chapter introduced the idea of non-reversible MCMC, and demonstrated that non-reversibility may help improve the Markov chain's mixing by suppressing random-walk behaviour. In this chapter, we now present a class of non-reversible MCMC algorithms that can target a general distribution, $\pi(\boldsymbol{\theta})$. These algorithms are different from standard MCMC algorithms in that they are based on simulating a continuous-time Markov chain. Furthermore, specific examples of these continuous-time MCMC algorithms can be derived as continuous-time limits of the non-reversible algorithms we introduced in the previous chapter.

As a way of motivating these algorithms, we will first look at this continuous-time limit. The resulting continuous-time process is from a class of processes called *piecewise deterministic Markov processes*. We will introduce some background on this class of processes, including some details around how we can simulate their continuous-time trajectories, before introducing example MCMC algorithms and various extensions. These algorithms use gradient information, and unless stated otherwise, we will assume that the target distribution $\pi(\boldsymbol{\theta})$ is differentiable everywhere (in practice being continuous and differentiable almost everywhere is sufficient).

Within this chapter, we will need to refer to both components of a vector and possibly the state of the vector at different times. To distinguish between these, we will use the convention that when both time and component are needed, we subscript by time and superscript by the component. So, for example, $\theta_t^{(i)}$, will be the ith component of the state, $\boldsymbol{\theta}$, at time t.

5.1 Continuous-Time MCMC as the Limit of Non-Reversible MCMC

To help build intuition for continuous-time MCMC, and to see how it links to discrete-time MCMC algorithms, we will show we can derive the

continuous-time algorithm as the limit of a discrete-time algorithm as we let the step size in the discrete-time algorithm to tend to 0. Here, we will consider this limiting argument at an informal level, before presenting a more formal justification for continuous-time MCMC methods.

We will consider sampling from a 1-dimensional target distribution using a simplified version of the non-reversible guided random walk algorithm that was introduced in Section 4.3. Our simplification is to assume that the step size at each iteration is fixed. Our state will still be (θ, p), with p either 1 or -1 and specifying the direction of the next proposed move, and we will let the size of the move be equal to some fixed value δ. Such an MCMC algorithm would not be irreducible, as it could only explore values of the state that are integer multiples of δ away from its initial value. However, the algorithm will still have the correct invariant distribution; i.e. if we simulate the initial state from the target distribution for θ and a uniform distribution for p, then this will also be the distribution of the state at any future iteration. The issue of lack of irreducibility vanishes in the limit as $\delta \to 0$.

As a reminder, if the target distribution of interest is $\pi(\theta)$ then the MCMC algorithm will iterate the following moves

(1a) Propose a move from (θ, p) to $(\theta+\delta p, -p)$. Accept this with the standard Metropolis–Hastings acceptance probability, which simplifies to

$$\min\left\{1, \frac{\pi(\theta + \delta p)}{\pi(\theta)}\right\}$$

(1b) Move from (θ', p) to $(\theta', -p)$.

In step (1a), we propose a move of size δ in direction p. If we accept this move, then we will flip the direction p in both steps (1a) and (1b) – so the direction will be the same at the next iteration. If we reject the move, then we will only flip p in step (1b) and thus the direction of the move will be flipped for the next iteration.

Under this framework, we can then consider letting $\delta \to 0$ while increasing the number of iterations n. That is we fix a value s and for any number of iterations, n will set $\delta = s/n$. We will scale time so that the ith MCMC transition will occur at time $i\delta$, and define (θ_t, p_t) to be the value of the state after the ith MCMC transition for $i\delta \leq t < (i+1)\delta$.

Now, for each move in step (1a) the rejection probability for small δ is

$$\begin{aligned}
&\max\{0, 1 - \exp[\log \pi(\theta + \delta p) - \log \pi(\theta)]\} \\
&= \max\left\{0, 1 - \exp\left[\frac{\mathrm{d}\log \pi(\theta)}{\mathrm{d}\theta}\delta + o(\delta)\right]\right\} \\
&= \max\left\{0, -p\frac{\mathrm{d}\log \pi(\theta)}{\mathrm{d}\theta}\right\}\delta + o(\delta),
\end{aligned}$$

assuming that, for example, the derivative of $\pi(\theta)$ is continuous.

Thus in our limit as $\delta \to 0$, rejections in step (1a) will occur as events in a Poisson process of rate

$$\lambda(\theta_t, p_t) = \max\left\{0, -p_t\frac{\mathrm{d}\log \pi(\theta_t)}{\mathrm{d}\theta}\right\}.$$

The dynamics between these events will be deterministic, with p_t being constant and θ_t changing as per a constant velocity model with velocity p_t. At each event, the velocity will just flip. While the process is moving to areas of higher probability density, as defined by $\pi(\theta)$, the rate of the Poisson process will be 0. Thus events will only occur if the process is moving to areas of lower probability mass.

How does this limiting, continuous-time algorithm compare to the algorithm of Gustafson (1998)? We will compare with the standard, irreducible version of Gustafson (1998) where the step size at each iteration is random. We show trace plots for both algorithms for sampling from a mixture of two Gaussians in Figure 5.1. The target distribution is an equal mix of a Gaussian with mean 2 and variance 1 and with mean 0 and variance 0.1^2. This was chosen so that we have two modes where different step sizes would be optimal, whilst still allowing for sufficient overlap of the modes that the chains would mix between them.

We can see that qualitatively the two trace-plots are similar. Both methods produce zig-zag-like traces, as they will continue to move in the same direction when moving to areas of higher probability density. However, the continuous-time process has a number of potential advantages:

- If we could simulate the continuous-time trajectory directly, then it has the benefit of having fewer events where the velocity changes (and where the state needs to be updated) than iterations of Gustafson's algorithm – in our example, there are around 200 events in the continuous-time algorithm as compared to 1000 iterations of Gustafson's algorithm.
- It also has the benefit of less tuning, as no step size needs to be specified.

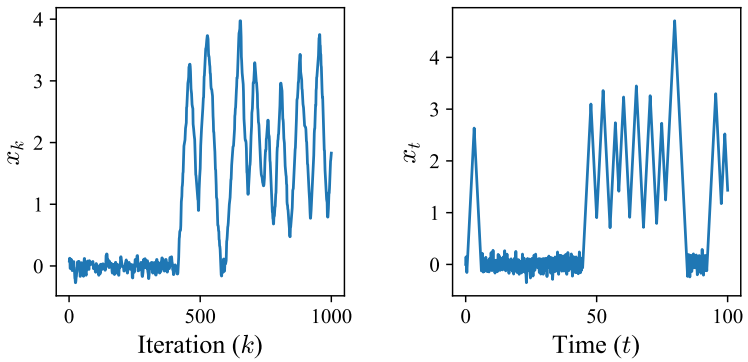

Figure 5.1 Trace plots of Gustafson's algorithm (left) and the continuous-time limit (right) for sampling from a mixture of Gaussians.

- Finally it has the advantage that the full continuous-time sample path can be used to calculate Monte Carlo averages.

However, directly simulating the continuous-time trajectory is not normally possible – and it is both the difficulty with simulating the continuous-time process and the additional computational overhead of doing so that are the main disadvantages. Furthermore, accounting for the very different computational costs, per iteration versus per unit of continuous time, makes the algorithms hard to compare theoretically. We will return to issues around simulating continuous-time Markov processes like this below.

5.2 Piecewise Deterministic Markov Processes

The continuous-time limiting process we derived in the previous section is an example of a *piecewise deterministic Markov process*, or PDMP. These are Markov processes that evolve deterministically between a set of random events. We will now introduce these processes before considering how they can be designed and used to sample from a target distribution of interest.

5.2.1 What is a PDMP?

We will denote the state of a PDMP at time t by $\mathbf{z}_t$. A PDMP is defined by the following properties:

(i) *The deterministic dynamics.* The PDMP evolves deterministically between a set of event times. We consider PDMPs where the deterministic dynamics are specified through an ordinary differential equation

$$\frac{\mathrm{d}\mathbf{z}_t}{\mathrm{d}t} = \boldsymbol{\phi}(\mathbf{z}_t), \tag{5.1}$$

for some gradient field $\boldsymbol{\phi}(\cdot)$. We will also define $\boldsymbol{\Phi}$ to be the transition function, or flow map, for these dynamics, so for $s > 0$ the solution to the differential equation satisfies $\mathbf{z}_{s+t} = \boldsymbol{\Phi}(\mathbf{z}_t, s)$.

(ii) *The event rate.* Random events occur at a rate $\lambda(\mathbf{z}_t)$ that depends on the current state.

(iii) *The transition kernel at events.* At each event, at time τ, the state changes according to some Markov transition kernel $\mathbb{Q}(\mathbf{Z}_\tau \in \cdot | \mathbf{z}_{\tau-})$, where

$$\mathbf{z}_{\tau-} = \lim_{\epsilon \downarrow 0} \mathbf{z}_{\tau-\epsilon}$$

is the value of the state immediately before the event. That is $\mathbb{Q}(\mathbf{Z}_\tau \in \cdot | \mathbf{z}_{\tau-})$ defines the conditional probability of moving to set $\cdot$, given the state immediately before the event.

To simplify exposition, and because this is consistent with the PDMPs we will use for MCMC, we will primarily focus on discrete transition kernels at events, and let $q(\cdot|\mathbf{z})$ denote the probability mass function associated with the transition kernel. Though the ideas below extend naturally to continuous transition kernels.

As the deterministic dynamics are Markov, and the event rate and transition density depend just on the current state, then the resulting process will be Markov.

5.2.2 Simulating PDMPs

To be able to use a PDMP as the basis of a sampling algorithm, we will need to be able to simulate and store realisations of the PDMP. First, we can store a realisation of the continuous-time path of a PDMP by storing just the initial state of the PDMP, and the time and event immediately after each event. We will call these points the *skeleton* of the realisation. If we simulate a PDMP up to time T then its skeleton will be of the form $\{t_k, \mathbf{z}_{t_k}\}_{k=0}^n$, where $t_0 = 0$, and $t_1, \ldots, t_n$ are the event times of the PDMP prior to T. Given this skeleton, we can fill in the continuous-time path using the transition function for the deterministic dynamics:

$$\mathbf{z}_t = \boldsymbol{\Phi}(\mathbf{z}_{t_k}, t - t_k), \text{where } t_k \text{ is the largest skeleton time less than } t.$$

Thus simulating a PDMP will just require the ability to simulate the skeleton points. If we assume that simulating from the transition density at events is straightforward, then the only challenge will be simulating the event times themselves. We can do this by using the following argument that shows that the time until the next event can be recast as the time until the first event in a time inhomogeneous Poisson process.

To simplify notation we will consider simulating the time of the first event, and denote the initial state as $\mathbf{Z}_0 = \mathbf{z}$. By the Markov property, the same approach can then be applied to simulating subsequent events: as if the current state is $\mathbf{Z}_t = \mathbf{z}$ then the subsequent time *until* the next event is the same as the time until the first event if we start the process at state $\mathbf{Z}_0 = \mathbf{z}$.

If there has not been an event by time t, then due to the deterministic dynamics between events we know that the state at time t will be $\mathbf{z}_t = \mathbf{\Phi}(\mathbf{z}, t)$. The instantaneous rate of an event at time t, if there has been no event before t, is thus $\lambda(\mathbf{z}_t) = \lambda(\mathbf{\Phi}(\mathbf{z}, t))$. Thus, the rate of the first event of the PDMP is equal to the rate of the first event in an inhomogeneous Poisson process with rate

$$\tilde{\lambda}_{\mathbf{z}}(t) = \lambda(\mathbf{\Phi}(\mathbf{z}, t)).$$

Here, we use the tilde symbol to distinguish this rate from the rate function, λ, that depends on the state. We also subscript the rate by $\mathbf{z}$, the initial state. As described above, by the Markov property, if we have simulated the PDMP until time s and the current state is $\mathbf{z}_s$, then the rate of the next event as a function of the further time, t, until the next event will be $\tilde{\lambda}_{\mathbf{z}_s}(t)$.

As a result, we have transformed the problem of simulating a PDMP to that of simulating the first event of a time inhomogeneous Poisson process. There are several methods for simulating such an event time (see e.g. Lewis and Shedler, 1979; Bouchard-Côté et al., 2018) and we will outline three general strategies for doing so. Whether these can be implemented in practice depends on the form of $\tilde{\lambda}_{\mathbf{z}}$, and we will return to this with some examples later.

Direct Simulation by Inversion

In theory, one can simulate directly the time to the next event. Let τ be the random variable that is the time until the first (or next) event. Standard properties of a Poisson process give that the probability of no event by time t is

$$\mathbb{P}(\tau > t) = \exp\left\{-\int_0^t \tilde{\lambda}_{\mathbf{z}}(s)\mathrm{d}s\right\}.$$

For a continuous random variable, X, with distribution function $F_X(\cdot)$, we have that the transformed random variable $F_X(X)$ has a standard uniform distribution. From this, we can simulate X by simulating u, a realisation of a standard uniform distribution and setting $x = F_X^{-1}(u)$. The distribution function for τ is $1 - \mathbb{P}(\tau > t)$, thus we can simulate τ from u by solving

$$u = 1 - \exp\left\{-\int_0^\tau \tilde{\lambda}_{\mathbf{z}}(s)\mathrm{d}s\right\}.$$

This can be rearranged to

$$-\int_0^\tau \tilde{\lambda}_{\mathbf{z}}(s)\mathrm{d}s = \log(1-u).$$

In practice, it is common to further simplify this expression using that if U is a standard uniform random variable then $-\log(1-U)$ has a standard exponential distribution. Thus if w is a realisation of a standard exponential random variable then τ can be simulated as the solution of

$$\int_0^\tau \tilde{\lambda}_{\mathbf{z}}(s)\mathrm{d}s = w. \tag{5.2}$$

A summary of this approach is given in Algorithm 7.

Algorithm 7: Direct simulation of event time

Input: Rate function $\tilde{\lambda}_{\mathbf{z}}$.
Simulate w, from a standard exponential distribution.
Find $\tau \geq 0$ the (smallest) solution to

$$\int_0^\tau \tilde{\lambda}_{\mathbf{z}}(s)\mathrm{d}s = w.$$

Output: τ.

Whether we can implement direct simulation depends on whether we can solve (5.2). This is possible if $\tilde{\lambda}_{\mathbf{z}}(s)$ is constant, linear, or piecewise linear in s. It is also possible if it is some other simple function, such as proportional to the exponential of a linear function of s.

Poisson Thinning

What if we cannot simulate the event directly by inversion? In this case, the most common approach to simulation is based on *Poisson thinning*. This approach is based on the following property of Poisson processes: If we have a Poisson process with rate $\lambda^+(s)$, and we simulate points from this

Poisson process and then accept each point with probability $\alpha(s)$, then the accepted points have the same distribution as points from a Poisson process with rate $\lambda^+(s)\alpha(s)$. As we are only keeping, i.e. accepting, some of the simulated points, this is called a *thinned* point process.

Poisson thinning inverts this property. For any rate function $\lambda^+(s)$ that upper bounds $\tilde{\lambda}_{\mathbf{z}}(s)$, i.e. where $\lambda^+(s) \geq \tilde{\lambda}_{\mathbf{z}}(s)$ for all $s \geq 0$, we can simulate the time until the next event as the first event of a thinned Poisson process. This leads to Algorithm 8

Algorithm 8: Simulation of event time via Poisson thinning

Input: Rate functions λ^+ and $\tilde{\lambda}_{\mathbf{z}}$, with $\lambda^+(s) \geq \tilde{\lambda}_{\mathbf{z}}(s)$ for all $s \geq 0$.
Set $s = 0$ and $\tau = 0$
while $\tau = 0$ **do**
  Simulate t, the time of the first event after s in a Poisson process with rate λ^+.
  Set $s = t$.
  With probability $\tilde{\lambda}_{\mathbf{z}}(t)/\lambda^+(t)$, set $\tau = t$.
end
Output: τ, the first event in a Poisson process with rate $\tilde{\lambda}_{\mathbf{z}}$.

For this to work we need to be able to upper bound $\tilde{\lambda}_{\mathbf{z}}$ by a simple rate function, for which we can directly simulate events. In practice, this would often be a linear or piecewise linear rate function. The efficiency of Poisson thinning depends on how close the upper bound rate is to $\tilde{\lambda}_{\mathbf{z}}$. In practice, we can improve on the simple Poisson thinning algorithm with adaptive thinning. That is, if we simulate an event and then reject it, we can use the information from evaluating $\tilde{\lambda}_{\mathbf{z}}$ to improve our upper bound. We will see some examples of such adaptive thinning below.

Superposition

The final property of Poisson processes that can help with simulating their events is that of *superposition*. This says that if we have two Poisson processes, one with rate $\lambda^{(1)}(s)$ and one with rate $\lambda^{(2)}(s)$, and we independently simulate events from each process and take the union of events, then these have the same distribution as events in a Poisson process with rate $\lambda^{(1)}(s) + \lambda^{(2)}(s)$.

In terms of simulating the first event in a Poisson process, this can be re-expressed as if $\tau^{(1)}$ is the first event time for a Poisson process with rate $\lambda^{(1)}(s)$, and $\tau^{(2)}$ is the first event time for a Poisson process with rate

$\lambda^{(2)}(s)$, then $\min\{\tau^{(1)}, \tau^{(2)}\}$ is distributed as the first event time in a Poisson process with rate $\lambda^{(1)}(s) + \lambda^{(2)}(s)$.

By induction, superposition trivially applies if we consider more than two independent Poisson processes. That is, the time of the first event in any of the processes is distributed as the time of the first event of a Poisson process whose rate is the sum of the rates. Superposition can be useful as it allows us to split a complex rate function into a sum of simpler rate functions. If we can simulate events from processes with each of the simpler rates, then it allows us to simulate events from the process with the complex rate function.

5.2.3 The Generator and Invariant Distribution of a PDMP

In order to use a PDMP to sample from a target distribution, we first need to be able to determine what the stationary distribution of a given PDMP is. Here, we give an informal derivation of how to calculate the invariant distribution of a PDMP. (Assuming the PDMP satisfies some regularity conditions, and in particular is irreducible, then this invariant distribution will be its stationary distribution.) In the next section, we will invert this characterisation of the invariant distribution to construct simple recipes for the dynamics of a PDMP to have a given target distribution as its stationary distribution. In terms of understanding the development of PDMP samplers in the rest of this chapter, the key result is (5.4) below – and those not interested in the intuition behind this result could skip the intervening material in this subsection.

First, we need to consider the generator of our PDMP. This is rigorously derived in Davis (1984). Remember from Section 1.4.2 that the generator of a continuous-time Markov process is an operator that gives the time derivative of the expected value of a function of the state, as a function of its current value. If $\mathcal{L}$ is the generator, and h a suitable test function from the domain of the generator, then

$$(\mathcal{L}h)(\mathbf{z}) = \lim_{t \downarrow 0} \frac{\mathbb{E}\left[h(\mathbf{Z}_t)|\mathbf{Z}_0 = \mathbf{z}\right] - h(\mathbf{z})}{t}.$$

Informally we can calculate the generator by considering the two types of dynamics of the PDMP. First, if we consider solely the deterministic dynamics (5.1), then the change in $h(\mathbf{z}_t)$ is deterministic and the contribution to the generator is just the time-derivative of $h(\mathbf{z}_t)$ at $t = 0$, which by the

product rule is

$$\left.\frac{\mathrm{d}h(\mathbf{z}_t)}{\mathrm{d}t}\right|_{t=0} = \boldsymbol{\phi}(\mathbf{z}) \cdot \nabla h(\mathbf{z}).$$

Second, we have the contribution from the random events. The probability of an event in $[0, t]$ is $\lambda(\mathbf{z})t + o(t)$. If an event occurs, the change in $h(\mathbf{z})$ is $\mathbb{E}_{\mathbb{Q}(\cdot|\mathbf{z})}[h(\mathbf{Z}')] - h(\mathbf{z}) + o(t)$, where the expectation is with respect to $\mathbf{Z}' \sim \mathbb{Q}(\cdot|\mathbf{z})$, the transition kernel at an event. This gives a contribution to the generator that is

$$\lambda(\mathbf{z}) \left[\mathbb{E}_{\mathbb{Q}(\cdot|\mathbf{z})}[h(\mathbf{Z}')] - h(\mathbf{z})\right].$$

Thus the generator is

$$(\mathcal{L}h)(\mathbf{z}) = \boldsymbol{\phi}(\mathbf{z}) \cdot \nabla h(\mathbf{z}) + \lambda(\mathbf{z}) \left[\mathbb{E}_{\mathbb{Q}(\cdot|\mathbf{z})}[h(\mathbf{Z}')] - h(\mathbf{z})\right].$$

The domain of the generator is given in Davis (1984). One extension of PDMPs that will be relevant later is that we can introduce boundaries, with additional specified, possibly random, behaviour if the PDMP hits a boundary. In such a case, the behaviour at the boundaries affects the generator solely through its domain. Essentially, the domain is reduced to include only those functions whose expectation is unaffected by the dynamics on the boundary.

If we start the PDMP with an initial distribution $\pi(\mathbf{z})$ for $\mathbf{Z}_0$, then the derivative of $\mathbb{E}[h(\mathbf{Z}_t)]$ at $t = 0$ is equal to the average of the generator applied to $h(\mathbf{z})$ with respect to $\pi(\mathbf{z})$. This is equal to

$$\int (\mathcal{L}h)(\mathbf{z})\pi(\mathbf{z})\mathrm{d}\mathbf{z}.$$

For any h in the domain of the generator, if $\pi(\mathbf{z})$ is sufficiently well-behaved, then this integral will be equal to

$$\int h(\mathbf{z})(\mathcal{L}^*\pi)(\mathbf{z})\mathrm{d}\mathbf{z}, \tag{5.3}$$

where $\mathcal{L}^*$ is a different operator, called the adjoint of the generator.

We can attempt to define the invariant distribution of the PDMP by the property that, if we draw $\mathbf{Z}_0$ from this invariant distribution, then the expectation of any function of the state will be constant over time – as if started from the invariant distribution, the marginal distribution of the PDMP will not change. Thus (5.3) must be equal to 0 if π is the invariant distribution. As this must happen for any function h of the state, for which the expectation exists, this suggests that π must satisfy $(\mathcal{L}^*\pi)(\mathbf{z}) = 0$.

It is possible to derive the adjoint $\mathcal{L}^*$ using integration by parts. From

this, we have that $(\mathcal{L}^*\pi)(\mathbf{z}) = 0$ implies that the invariant distribution must satisfy

$$-\sum_{i=1}^{d} \frac{\partial \phi_i(\mathbf{z})\pi(\mathbf{z})}{\partial z_i} + \sum_{\mathbf{z}'} \pi(\mathbf{z}')\lambda(\mathbf{z}')q(\mathbf{z}|\mathbf{z}') - \pi(\mathbf{z})\lambda(\mathbf{z}) = 0, \tag{5.4}$$

where $q(\mathbf{z}|\mathbf{z}')$ is the probability mass function associated with the transition kernel $\mathbb{Q}(\cdot|\mathbf{z}')$. This equation has a natural interpretation. The first term on the left-hand side quantifies the change in probability mass due to the deterministic dynamics, the second term is the change due to events moving into state $\mathbf{z}$ and the last is the change due to events that move the state out of $\mathbf{z}$. If π is an invariant distribution, then the net change in probability mass is zero.

In the following, we will use (5.4) to define the invariant distribution of our PDMP. Though this requires inverting the informal argument we have made – see Chevallier et al. (2021) for results that give relatively weak conditions under which you can invert this argument and where (5.4) implies that $\pi(\mathbf{z})$ is the invariant distribution of our PDMP.

5.2.4 The Limiting Process of Section 5.1 as a PDMP

We can now recognise the limiting process derived in Section 5.1 as a PDMP. Remember that we want to sample from a distribution $\pi(\theta)$ for some scalar θ. To do this we introduced a velocity or momentum, p, and defined a state $\mathbf{z} = (\theta, p)$ – strictly this is $\mathbf{z} = (\theta, p)^\top$, but we will use the shorthand (θ, p) in the following. Henceforth, we will use the notation $\mathbf{z}$, or $\mathbf{z}_t$, and (θ, p), or (θ_t, p_t) interchangeably – as viewed most helpful for the given context. For reasons that will become apparent, we will call θ the position component of the state, and p the velocity component.

The limiting process of Section 5.1 was a PDMP with state $\mathbf{z} = (\theta, p)$, with $\theta \in \mathbb{R}$ and $p \in \{-1, 1\}$, defined by the following properties:

(U1) *Deterministic dynamics.* The deterministic dyanamics are a constant velocity model:

$$\frac{\mathrm{d}\theta_t}{\mathrm{d}t} = p_t, \quad \frac{\mathrm{d}p_t}{\mathrm{d}t} = 0.$$

(U2) *Event rate.* The rate of events is

$$\lambda(\theta, p) = \max\left\{0, -p\frac{\mathrm{d}\log\pi(\theta)}{\mathrm{d}\theta}\right\}.$$

(U3) *Transition kernel at events.* At an event the velocity component of the state flips, i.e. $p_\tau = -p_{\tau-}$.

We will call this PDMP the *univariate* PDMP.

It is possible to show, using (5.4), that the invariant distribution of this PDMP is $\tilde{\pi}(\mathbf{z}) = \pi(\theta)\pi_p(p)$, where π_p is the probability mass function for a uniform distribution on $\{-1, 1\}$. That is the θ-marginal is $\pi(\theta)$, the distribution we wish to sample from. Furthermore, under the invariant distribution, p is independent of θ and has a uniform distribution. For this simple example, the invariant distribution will also be the stationary distribution, unless we have reducibility caused by a region where $\pi(\theta) = 0$ that separates two regions with positive probability under π.

As we will be considering more general PDMP samplers, it is helpful to consider a slightly different question. For PDMPs with the given deterministic dynamics and transitions at events, how would we calculate event rates that would give an invariant distribution whose θ marginal is $\pi(\theta)$? In answering this question, we will cover the steps for showing that the event rate given above leads to the stated invariant distribution.

To do this we will use (5.4). If we substitute in a distribution $\tilde{\pi}(\theta, p)$, and the deterministic dynamics and transition kernel at events, this illustrates that for $\tilde{\pi}(\theta, p)$ to be an invariant distribution, it must satisfy

$$-p\frac{\mathrm{d}\tilde{\pi}(\theta, p)}{\mathrm{d}\theta} + \lambda(\theta, -p)\tilde{\pi}(\theta, -p) - \lambda(\theta, p)\tilde{\pi}(\theta, p) = 0. \tag{5.5}$$

Here, the first term comes from the constant velocity deterministic dynamics, and the second comes from there only being one possible transition to state (θ, p) at an event, and this is from a state $(\theta, -p)$.

So what event rate would lead to an invariant distribution with the correction θ-marginal? In answering this question, we first see that there may be a range of different event rates that would lead to a valid invariant distribution. Not least because many possible invariant distributions would have a θ-marginal as $\pi(\theta)$. So, our first step is to attempt to find a rate such that the invariant distribution has θ independent of p. Denote such a distribution by $\tilde{\pi}(\mathbf{z}) = \pi(\theta)\pi_p(p)$, where π_p can be any distribution on $\{-1, 1\}$. Then substituting this into (5.5), and using

$$\frac{\mathrm{d}\pi(\theta)}{\mathrm{d}\theta} = \pi(\theta)\frac{\mathrm{d}\log\pi(\theta)}{\mathrm{d}\theta}$$

gives

$$-p\frac{\mathrm{d}\log\pi(\theta)}{\mathrm{d}\theta}\pi(\theta)\pi_p(p) + \lambda(\theta, -p)\pi(\theta)\pi_p(-p) - \lambda(\theta, p)\pi(\theta)\pi_p(p) = 0.$$

If we consider θ for which $\pi(\theta) > 0$ then this states

$$\lambda(\theta, -p)\pi_p(-p) - \lambda(\theta, p)\pi_p(p) = p\frac{\mathrm{d}\log\pi(\theta)}{\mathrm{d}\theta}\pi_p(p). \tag{5.6}$$

If we consider the same argument, but for the state $(\theta, -p)$, we get

$$\lambda(\theta, p)\pi_p(p) - \lambda(\theta, -p)\pi_p(-p) = -p\frac{\mathrm{d}\log\pi(\theta)}{\mathrm{d}\theta}\pi_p(-p). \tag{5.7}$$

Adding (5.6) to (5.7) gives

$$0 = \frac{\mathrm{d}\log\pi(\theta)}{\mathrm{d}\theta}\left(p\pi_p(p) - p\pi_p(-p)\right).$$

From this, we can conclude that the distribution π_p must satisfy $\pi_p(p) = \pi_p(-p)$, i.e. be the uniform distribution on $\{-1, 1\}$. This makes sense intuitively. The transition at the events only changes the velocity. Thus invariance for the θ-component comes from averaging out the dynamics for different p, and this requires that the invariant distribution for p has a mean of zero. As p can only take two values, this means it must be the uniform distribution. For the PDMPs that we consider later, that only change the velocity at events and have constant velocity dynamics, a similar argument holds that the invariant distribution for the velocity component must have zero as the mean.

If we now return to our question of what rates will lead to an invariant distribution with θ-marginal equal to $\pi(\theta)$, and substitute in (5.7) that $\pi_p(p) = \pi_p(-p) = 1/2$ for $p \in \{-1, 1\}$ then, by removing this common factor, we get

$$\lambda(\theta, p) - \lambda(\theta, -p) = -p\frac{\mathrm{d}\log\pi(\theta)}{\mathrm{d}\theta}.$$

A solution to this equation is the rate we specified above,

$$\lambda(\theta, p) = \max\left\{0, -p\frac{\mathrm{d}\log\pi(\theta)}{\mathrm{d}\theta}\right\}. \tag{5.8}$$

However, this is not the only solution. In fact, for any positive function $\gamma(\theta) \geq 0$, the rate

$$\max\left\{0, -p\frac{\mathrm{d}\log\pi(\theta)}{\mathrm{d}\theta}\right\} + \gamma(\theta),$$

will also lead to the same invariant distribution.

The rate in (5.8) is the smallest rate that will lead to the required invariant distribution. This is often called the *canonical rate*. Intuitively, there are two advantages of using the canonical rate, as opposed to a larger rate. The

first is that a larger rate will lead to more events, and thus is likely to have a larger computational cost for simulating the resulting PDMP. The second is that a larger rate will lead to more changes in velocity and will re-introduce the random walk behaviour that we were trying to avoid with non-reversible MCMC. Thus, we would expect that using the canonical rate will lead to better mixing.

A final comment on the rates we are required to use, such as the canonical rate, is that they depend on the target distribution $\pi(\theta)$ only through the derivative of $\nabla \log \pi(\theta)$. This is important as it means that $\pi(\theta)$ only needs to be known up to a constant of proportionality, as is commonly the case for sampling from the posterior distribution in Bayesian statistics, see Section 1.1.5.

5.3 Continuous-time MCMC via PDMPs

In practice, we will want to use MCMC to sample a target density in $\mathbb{R}^d$. Various PDMPs generalise the process introduced in the previous section to $d > 1$. We will describe three such families of PDMPs, all of which reduce to the univariate PDMP of Section 5.2.4 if $d = 1$ but differ for $d > 1$. They each share some common features, which we will describe first.

Assume we wish to sample from $\pi(\boldsymbol{\theta})$ where $\boldsymbol{\theta}$ is d-dimensional. The state of our PDMP will be $\mathbf{z}_t = (\boldsymbol{\theta}_t^\top, \mathbf{p}_t^\top)^\top$, where $\mathbf{p}_t$ is also d-dimensional. As we use the convention that vectors are column vectors, when defining $\mathbf{z}_t$ we have had to transpose these vectors to concatenate $\boldsymbol{\theta}_t$ and $\mathbf{p}_t$. In the following, to simplify notation, we will abuse this and just write $\mathbf{z}_t = (\boldsymbol{\theta}_t, \mathbf{p}_t)$. As before, we can think of $\boldsymbol{\theta}$ as the position component of the state, and $\mathbf{p}$ as the velocity.

The deterministic dynamics of the three families of PDMPs will be the same:

(CV) *Deterministic dynamics.* The process evolves according to a *Constant Velocity (CV)* model.

$$\frac{\mathrm{d}\boldsymbol{\theta}_t}{\mathrm{d}t} = \mathbf{p}_t, \quad \frac{\mathrm{d}\mathbf{p}_t}{\mathrm{d}t} = \mathbf{0}, \tag{5.9}$$

or in the notation we used to define PDMPs, $\boldsymbol{\phi} = (\mathbf{p}, \mathbf{0})$. The solution of the deterministic dynamics are

$$\boldsymbol{\Phi}(\mathbf{z}, t) = \boldsymbol{\Phi}((\boldsymbol{\theta}, \mathbf{p}), t) = (\boldsymbol{\theta} + t\mathbf{p}, \mathbf{p}). \tag{5.10}$$

Furthermore, they also only allow the velocity component to change at events.

The three families of PDMPs will differ in terms of the possible values for the velocity component, the event rate, and the transition kernel for the velocity at events. In each case, these are chosen so that the invariant distribution of the PDMP will have a $\boldsymbol{\theta}$-marginal that is $\pi(\boldsymbol{\theta})$, and for which the velocity component, $\mathbf{p}$, is independent of the position $\boldsymbol{\theta}$. Throughout, we will denote the invariant distribution as $\tilde{\pi}(\boldsymbol{\theta}, \mathbf{p}) = \pi(\boldsymbol{\theta})\pi_{\mathbf{p}}(\mathbf{p})$, though as mentioned, the form of $\pi_{\mathbf{p}}$ will differ between different families of PDMPs.

Before we describe the three families in detail, it is helpful to introduce some notation. We will use $\nabla_{\boldsymbol{\theta}}$ to denote the gradient vector with respect to $\boldsymbol{\theta}$ only. This is the d-dimensional column vector whose entries are the partial derivatives with respect to the components of $\boldsymbol{\theta}$. The first term in the equation for the invariant distribution of the PDMP (5.4) will be the same for all three families as they share the same deterministic dynamics and form of the invariant distribution. Ignoring the minus sign, this term is

$$\begin{aligned}\sum_{i=1}^{2d} \frac{\partial \phi_i(\mathbf{z})\tilde{\pi}(\mathbf{z})}{\partial z_i} &= \sum_{i=1}^{d} \frac{\partial p_i \tilde{\pi}(\boldsymbol{\theta}, \mathbf{p})}{\partial \theta_i} = \pi_{\mathbf{p}}(\mathbf{p}) \sum_{i=1}^{d} p_i \frac{\partial \pi(\boldsymbol{\theta})}{\partial \theta_i} \\ &= \pi_{\mathbf{p}}(\mathbf{p}) \sum_{i=1}^{d} \pi(\boldsymbol{\theta}) p_i \frac{\partial \log \pi(\boldsymbol{\theta})}{\partial \theta_i} \\ &= \tilde{\pi}(\boldsymbol{\theta}, \mathbf{p}) \ (\mathbf{p} \cdot \nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{\theta})) \,. \end{aligned} \tag{5.11}$$

Here, we have first used that only the θ_i components are changing, and then used $\tilde{\pi}(\boldsymbol{\theta}, \mathbf{p}) = \pi(\boldsymbol{\theta})\pi_{\mathbf{p}}(\mathbf{p})$. The third step comes from the definition of the derivative of $\log \pi(\boldsymbol{\theta})$ in terms of the derivative of $\pi(\boldsymbol{\theta})$, and the final step from using $\pi(\boldsymbol{\theta})\pi_{\mathbf{p}}(\mathbf{p}) = \tilde{\pi}(\boldsymbol{\theta}, \mathbf{p})$.

5.3.1 Different Samplers

The Coordinate Sampler

Possibly the simplest extension of our univariate PDMP to one that samples from a multi-dimensional distribution is the *Coordinate Sampler* of Wu and Robert (2020). For this sampler, the set of possible velocities is $\mathcal{V}_{\text{cs}} = \{\pm \mathbf{e}_i\}_{i=1}^{d}$ where $\mathbf{e}_i$ is the ith unit vector. That is, $\mathbf{e}_i$ is the unit vector whose ith component is 1, and all other components are 0. Thus, the possible velocities correspond to moving in either a positive or negative direction along one of the coordinate axes in $\mathbb{R}^d$. It can be viewed as a sampler which applies the univariate PDMP dynamics along each coordinate in turn – though the order in which different coordinate directions are chosen is random. Introducing a constant *refresh rate* $\lambda_{\text{r}} \geq 0$, the dynamics of the coordinate

sampler involve the *constant velocity* (CV) deterministic dynamics together with

(CS1) *Event rate.* Events occur with the rate

$$\lambda_{\text{cs}}(\boldsymbol{\theta}, \mathbf{p}) = \max\{0, -\mathbf{p} \cdot \nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{\theta})\} + \lambda_{\text{r}}.$$

(CS2) *Transition kernel at events.* At an event, the probability of switching to a new velocity $\mathbf{p}' \in \mathcal{V}_{\text{cs}}$ is

$$q_{\text{cs}}((\boldsymbol{\theta}, \mathbf{p}')|(\boldsymbol{\theta}, \mathbf{p})) = \frac{1}{C(\boldsymbol{\theta})} \left(\max\{0, \mathbf{p}' \cdot \nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{\theta})\} + \lambda_{\text{r}}\right),$$

where the normalising constant is

$$C(\boldsymbol{\theta}) = \sum_{\mathbf{p}' \in \mathcal{V}_{\text{cs}}} \left(\max\{0, \mathbf{p}' \cdot \nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{\theta})\} + \lambda_{\text{r}}\right) = 2d\lambda_{\text{r}} + \sum_{i=1}^{d} \left| \frac{\partial \log \pi(\boldsymbol{\theta})}{\partial \theta_i} \right|.$$

The refresh rate λ_{r} introduces additional random velocity switches. As discussed above, intuitively a larger refresh rate will lead to more random walk behaviour and thus worse mixing. However, choosing $\lambda_{\text{r}} > 0$ allows for stronger theoretical results about the sampler, including that the sampler will be irreducible unless e.g. $\pi(\boldsymbol{\theta})$ has disconnected regions where there is positive probability.

The invariant distribution of the Coordinate Sampler is given by the following result.

Theorem 5.1 *For any $\lambda_r \geq 0$, the Coordinate Sampler, whose dynamics are defined by (CV), (CS1) and (CS2), has an invariant distribution $\tilde{\pi}(\boldsymbol{\theta}, \mathbf{p}) = \pi(\boldsymbol{\theta})\pi_{\mathbf{p}}(\mathbf{p})$ where $\pi_{\mathbf{p}}$ is the uniform distribution over $\mathcal{V}_{cs}$.*

Proof We show this result by showing that (5.4) holds. To simplify expressions slightly, we will use the notation that for any scalar a we have $\{a\}_+ = \max\{0, a\}$.

Substituting in the event rate and transition kernel and distribution $\tilde{\pi}$, for $\mathbf{p} \in \mathcal{V}_{\text{cs}}$, the left-hand side of (5.4) is

$$\begin{aligned}
&\tilde{\pi}(\boldsymbol{\theta}, \mathbf{p}) \left(-\mathbf{p} \cdot \nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{\theta})\right) - \tilde{\pi}(\boldsymbol{\theta}, \mathbf{p})(\{-\mathbf{p} \cdot \nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{\theta})\}_+ + \lambda_{\text{r}}) \\
&+ \sum_{\mathbf{p}' \in \mathcal{V}_{\text{cs}}} (\{-\mathbf{p}' \cdot \nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{\theta})\}_+ + \lambda_{\text{r}}) \tilde{\pi}(\boldsymbol{\theta}, \mathbf{p}') q_{\text{cs}}((\boldsymbol{\theta}, \mathbf{p})|(\boldsymbol{\theta}, \mathbf{p}')),
\end{aligned}$$

where we have used (5.9) to simplify the first term. By the definition of $\tilde{\pi}$,

we have $\tilde{\pi}(\boldsymbol{\theta}, \mathbf{p}) = \tilde{\pi}(\boldsymbol{\theta}, \mathbf{p}')$ for all $\mathbf{p}, \mathbf{p}' \in \mathcal{V}_{\text{cs}}$. Thus this simplifies to

$$-\mathbf{p} \cdot \nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{\theta}) - (\{-\mathbf{p} \cdot \nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{\theta})\}_+ + \lambda_{\text{r}}) + \sum_{\mathbf{p}' \in \mathcal{V}_{\text{cs}}} (\{-\mathbf{p}' \cdot \nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{\theta})\}_+ + \lambda_{\text{r}}) q_{\text{cs}}((\boldsymbol{\theta}, \mathbf{p}) | (\boldsymbol{\theta}, \mathbf{p}')).$$

Now using the definition of q_{cs}, we get that the final term in this expression is

$$\begin{aligned}
&\sum_{\mathbf{p}' \in \mathcal{V}_{\text{cs}}} (\{-\mathbf{p}' \cdot \nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{\theta})\}_+ + \lambda_{\text{r}}) q_{\text{cs}}((\boldsymbol{\theta}, \mathbf{p}) | (\boldsymbol{\theta}, \mathbf{p}')) \\
&= \sum_{\mathbf{p}' \in \mathcal{V}_{\text{cs}}} (\{-\mathbf{p}' \cdot \nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{\theta})\}_+ + \lambda_{\text{r}}) \frac{1}{C(\boldsymbol{\theta})} \left(\{\mathbf{p} \cdot \nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{\theta})\}_+ + \lambda_{\text{r}}\right) \\
&= \left(\{\mathbf{p} \cdot \nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{\theta})\}_+ + \lambda_{\text{r}}\right) \frac{1}{C(\boldsymbol{\theta})} \sum_{\mathbf{p}' \in \mathcal{V}_{\text{cs}}} (\{-\mathbf{p}' \cdot \nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{\theta})\}_+ + \lambda_{\text{r}}) \\
&= \left(\{\mathbf{p} \cdot \nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{\theta})\}_+ + \lambda_{\text{r}}\right),
\end{aligned}$$

where for the last line we use

$$C(\boldsymbol{\theta}) = \sum_{\mathbf{p}' \in \mathcal{V}_{\text{cs}}} (\{-\mathbf{p}' \cdot \nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{\theta})\}_+ + \lambda_{\text{r}}).$$

Substituting into our expression for the left-hand side of (5.4) gives

$$\begin{aligned}
&-\mathbf{p} \cdot \nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{\theta}) - (\{-\mathbf{p} \cdot \nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{\theta})\}_+ + \lambda_{\text{r}}) + \left(\{\mathbf{p} \cdot \nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{\theta})\}_+ + \lambda_{\text{r}}\right) \\
&= -\mathbf{p} \cdot \nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{\theta}) - (\{-\mathbf{p} \cdot \nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{\theta})\}_+) + \left(\{\mathbf{p} \cdot \nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{\theta})\}_+\right).
\end{aligned}$$

By considering separately the cases where $\mathbf{p} \cdot \nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{\theta}) \geq 0$ and $\mathbf{p} \cdot \nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{\theta}) < 0$, it is simple to see that this expression for the left-hand side of (5.4) is 0, as required. □

The Zig–Zag Sampler

We now present the *Zig–Zag* algorithm of Bierkens et al. (2019b). This algorithm has velocities in $\mathcal{V}_{\text{zz}} = \{\pm 1\}^d$, and the state moves simultaneously along each coordinate axis, and the velocity determines which direction it moves for each axis. There are 2^d possible velocities, and if for example $d = 2$, these will be $(1, 1)$, $(1, -1)$, $(-1, 1)$ and $(-1, -1)$. At an event, one component of the velocity will change signs. The sampler gets its name from the resulting dynamics consisting of zig-zagging lines.

To define the dynamics of the Zig–Zag Sampler it is helpful to introduce

coordinate-specific rates

$$\lambda_i(\boldsymbol{\theta}, \mathbf{p}) = \max\left\{0, -p_i \frac{\partial \log \pi(\boldsymbol{\theta})}{\partial \theta_i}\right\},$$

which is of the same form as the canonical rate of the univariate PDMP if we just vary that ith component of $\boldsymbol{\theta}$. We also introduce the functions F_i, for $i = 1, \ldots, d$, which flips the sign of the ith component of a vector. So if $\mathbf{p}' = F_i(\mathbf{p})$ then $p'_i = -p_i$ and, for $j \neq i$, $p'_j = p_j$.

The PDMP process is defined by CV dynamics together with

(ZZ1) *Event rate.* Events occur with the rate

$$\lambda_{\text{zz}}(\boldsymbol{\theta}, \mathbf{p}) = \sum_{i=1}^{d} \lambda_i(\boldsymbol{\theta}, \mathbf{p}).$$

(ZZ2) *Transition kernel at events.* At an event the probability mass function of the transition is

$$q_{\text{zz}}(\boldsymbol{\theta}, \mathbf{p}' | \boldsymbol{\theta}, \mathbf{p}) = \frac{\lambda_i(\boldsymbol{\theta}, \mathbf{p})}{\lambda_{\text{zz}}(\boldsymbol{\theta}, \mathbf{p})}, \quad \text{for } \mathbf{p}' = F_i(\mathbf{p}).$$

Thus the position is unchanged, and we flip component i of the velocity with probability proportional to $\lambda_i(\boldsymbol{\theta}, \mathbf{p})$.

Here we have presented the dynamics in terms of the rate of an event and a transition probability for that event. However, by the superposition property of Poisson processes that was discussed above, one can equivalently represent the dynamics in terms of d possible event types. Event type i corresponds to flipping the ith component of the velocity, and this event occurs, independently of other events, with rate λ_i. This view of the dynamics of the Zig–Zag algorithm is often used in algorithmic implementations to sample realisations of the process.

We can relate the Zig–Zag Sampler to a limiting version of the guided random walk algorithm of Gustafson (1998) in a similar way to the argument presented in Section 5.1. The Zig–Zag Sampler is the limit of an MCMC algorithm that repeatedly applies one iteration of the guided random walk algorithm to each component of $\boldsymbol{\theta}$ in turn.

The following result gives the invariant distribution of the PDMP process.

Theorem 5.2 *The Zig–Zag Sampler, defined by (CV), (ZZ1) and (ZZ2) has invariant distribution $\tilde{\pi}(\boldsymbol{\theta}, \mathbf{p}) = \pi(\boldsymbol{\theta})\pi_{\mathbf{p}}(\mathbf{p})$ where $\pi_{\mathbf{p}}$ is the uniform distribution over $\mathcal{V}_{zz}$.*

Proof Again, we show this result by showing that (5.4) holds. By the same argument as in the first step of the proof of Theorem 5.1, if we substitute the form of $\tilde{\pi}$ and the definition of the dynamics of the Zig–Zag Sampler into the left-hand side of (5.4) we get that this is proportional to

$$-\mathbf{p} \cdot \nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{\theta}) - \sum_{i=1}^{d} \lambda_i(\boldsymbol{\theta}, \mathbf{p}) + \sum_{i=1}^{d} \lambda_i(\boldsymbol{\theta}, F_i(\mathbf{p})). \tag{5.12}$$

The first term relates to the change in probability mass due to the deterministic dynamics and is the same term as appeared in the calculations for the Coordinate Sampler. The second term is the rate of leaving the state $(\boldsymbol{\theta}, \mathbf{p})$, and the third is the rate of moving to the state $(\boldsymbol{\theta}, \mathbf{p})$ which has to be from a state of the form $(\boldsymbol{\theta}, F_i(\mathbf{p}))$. As in the argument for the Coordinate Sampler, we have removed the $\tilde{\pi}$ terms as these are the same for $(\boldsymbol{\theta}, \mathbf{p})$ and $(\boldsymbol{\theta}, F_i(\mathbf{p}))$ for all i.

To simplify this expression we use

$$-\lambda_i(\boldsymbol{\theta}, \mathbf{p}) + \lambda(\boldsymbol{\theta}, F_i(\mathbf{p})) = p_i \frac{\partial \log \pi(\boldsymbol{\theta})}{\partial \theta_i},$$

and

$$\mathbf{p} \cdot \nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{\theta}) = \sum_{i=1}^{d} p_i \frac{\partial \log \pi(\boldsymbol{\theta})}{\partial \theta_i}.$$

Thus (5.12) becomes

$$\sum_{i=1}^{d} \left(-p_i \frac{\partial \log \pi(\boldsymbol{\theta})}{\partial \theta_i} + p_i \frac{\partial \log \pi(\boldsymbol{\theta})}{\partial \theta_i} \right) = 0,$$

as required. □

Bouncy Particle Sampler

The third sampler that we introduce is the *Bouncy Particle Sampler*, which was first introduced as a way of simulating particle systems in statistical mechanics (Peters and de With, 2012), but was then proposed as a general sampling algorithm by Bouchard-Côté et al. (2018). It can be derived as a continuous-time limit of the Discrete Bouncy Particle Sampler that was introduced in Section 4.4. Like that algorithm, the transitions at events are reflections of the velocity in the contours of $\log \pi(\boldsymbol{\theta})$.

For a d-dimensional vector $\mathbf{g}$, let $\widehat{\mathbf{g}} = \mathbf{g}/(\mathbf{g} \cdot \mathbf{g})^{1/2}$ be the unit vector in the direction of $\mathbf{g}$. As in Section 4.4, define the function $\mathcal{R}_{\mathbf{g}}(\mathbf{p}) = \mathbf{p} - 2(\mathbf{p} \cdot \widehat{\mathbf{g}})\, \widehat{\mathbf{g}}$, to be the reflection of $\mathbf{p}$ in the hyperplane perpendicular to $\mathbf{g}$. An important

property of a reflection, that we will use below, is that it preserves the size of the vector. That is

$$\begin{aligned}\mathcal{R}_{\mathbf{g}}(\mathbf{p}) \cdot \mathcal{R}_{\mathbf{g}}(\mathbf{p}) &= \left(\mathbf{p} - 2(\mathbf{p} \cdot \widehat{\mathbf{g}})\, \widehat{\mathbf{g}}\right) \cdot \left(\mathbf{p} - 2(\mathbf{p} \cdot \widehat{\mathbf{g}})\, \widehat{\mathbf{g}}\right) \\ &= \mathbf{p} \cdot \mathbf{p} - 4(\mathbf{p} \cdot \widehat{\mathbf{g}})\, \mathbf{p} \cdot \widehat{\mathbf{g}} + 4(\mathbf{p} \cdot \widehat{\mathbf{g}})^2 \widehat{\mathbf{g}} \cdot \widehat{\mathbf{g}} \\ &= \mathbf{p} \cdot \mathbf{p} - 4(\mathbf{p} \cdot \widehat{\mathbf{g}})^2 + 4(\mathbf{p} \cdot \widehat{\mathbf{g}})^2 = \mathbf{p} \cdot \mathbf{p},\end{aligned}$$

where for the penultimate equality we have used that $\widehat{\mathbf{g}}$ is a unit vector. Also, reflection is an involution, that is $\mathcal{R}_{\mathbf{g}}(\mathcal{R}_{\mathbf{g}}(\mathbf{p})) = \mathbf{p}$. To see this

$$\begin{aligned}\mathcal{R}_{\mathbf{g}}(\mathcal{R}_{\mathbf{g}}(\mathbf{p})) &= \mathcal{R}_{\mathbf{g}}\left(\mathbf{p} - 2(\mathbf{p} \cdot \widehat{\mathbf{g}})\, \widehat{\mathbf{g}}\right) \\ &= \mathbf{p} - 2(\mathbf{p} \cdot \widehat{\mathbf{g}})\, \widehat{\mathbf{g}} - 2\{(\mathbf{p} - 2(\mathbf{p} \cdot \widehat{\mathbf{g}})\, \widehat{\mathbf{g}}) \cdot \widehat{\mathbf{g}}\}\, \widehat{\mathbf{g}} \\ &= \mathbf{p} - 2(\mathbf{p} \cdot \widehat{\mathbf{g}})\, \widehat{\mathbf{g}} - 2\{\mathbf{p} \cdot \widehat{\mathbf{g}} - 2(\mathbf{p} \cdot \widehat{\mathbf{g}})\}\, \widehat{\mathbf{g}} \\ &= \mathbf{p} - 2(\mathbf{p} \cdot \widehat{\mathbf{g}})\, \widehat{\mathbf{g}} + 2(\mathbf{p} \cdot \widehat{\mathbf{g}})\, \widehat{\mathbf{g}} = \mathbf{p}.\end{aligned}$$

There are two versions of the Bouncy Particle Sampler, that differ only in the set of possible velocities. We will mainly work with the version where the velocities take values in $\mathbb{R}^d$, and have an invariant distribution that is standard normal. For any refresh rate $\lambda_{\mathrm{r}} \geq 0$, the Bouncy Particle Sampler in this case is a PDMP with constant velocity dynamics (CV) and

(BPS1) *Event rate.* Events occur at a rate

$$\lambda_{\mathrm{BPS}}(\boldsymbol{\theta}, \mathbf{p}) = \max\{0, -\mathbf{p} \cdot \nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{\theta})\} + \lambda_{\mathrm{r}}.$$

(BPS2) *Transition at events.* At an event with probability $1 - \lambda_{\mathrm{r}}/\lambda_{\mathrm{BPS}}(\boldsymbol{\theta}, \mathbf{p})$, reflect the velocity in the hyperplane perpendicular to $\nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{\theta})$, that is the new velocity is

$$\mathbf{p}' = \mathcal{R}_{\mathbf{g}}(\mathbf{p}), \text{ with } \mathbf{g} = \nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{\theta});$$

otherwise sample a new velocity, $\mathbf{p}'$ from a standard normal distribution. The position is unchanged at an event.

As with the Zig–Zag Sampler, we can interpret the dynamics in terms of events of different types. In this case, we have reflection events that occur with rate $\max\{0, -\mathbf{p} \cdot \nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{\theta})\}$, and refresh events that occur with rate λ_{r}. If $\lambda_{\mathrm{r}} > 0$ then Bouchard-Côté et al. (2018) prove that the resulting process is irreducible, assuming weak conditions on $\pi(\boldsymbol{\theta})$. Furthermore, they give an example where if $\lambda_{\mathrm{r}} = 0$, the sampler will be reducible, and this occurs if we were to use the Bouncy Particle Sampler to sample from a Gaussian distribution. As many target distributions can be close to Gaussian, we may have a reducible sampler, or one which mixes slowly if $\lambda_{\mathrm{r}} = 0$. In practice, tuning λ_{r} is important, as not only can the sampler mix poorly for $\lambda_{\mathrm{r}} \approx 0$,

but if we choose λ_{r} too large it will introduce random walk behaviour which will also lead to poor mixing. We will return to this issue later in this section and in Section 5.3.3

The alternative version of the algorithm has velocities that lie on the unit d-dimensional hypersphere. The only difference in terms of the dynamics is that at a refresh event, we sample a new velocity from the uniform distribution on the sphere rather than from a standard normal distribution. Furthermore, there are extensions of the Bouncy Particle Sampler that only partially refresh the velocity. That is at a refresh event we sample a new velocity from a Markov kernel which has a standard normal distribution (or for the alternative version a uniform distribution on the sphere) as its stationary distribution.

The following result gives the invariant distribution of the Bouncy Particle Sampler.

Theorem 5.3 *For any $\lambda_r \geq 0$, the Bouncy Particle Sampler, whose dynamics are defined by (CV), (BPS1) and (BPS2), has an invariant distribution $\tilde{\pi}(\boldsymbol{\theta}, \mathbf{p}) = \pi(\boldsymbol{\theta})\pi_{\mathbf{p}}(\mathbf{p})$ where $\pi_{\mathbf{p}}$ is the density of a d-dimensional standard normal distribution.*

Proof As our presentation of PDMPs has focussed on discrete transitions at events, we will prove the result for $\lambda_{\mathrm{r}} = 0$ only. The extension to $\lambda_{\mathrm{r}} > 0$ is straightforward, as the additional refresh rates trivially keep $\tilde{\pi}$ invariant.

As shown above, a reflection does not change the length of a vector. As for the proposed invariant distribution, $\pi_{\mathbf{p}}(\mathbf{p})$ depends on $\mathbf{p}$ only through its length, we have $\pi_{\mathbf{p}}(\mathbf{p}) = \pi_{\mathbf{p}}(\mathcal{R}_{\mathbf{g}}(\mathbf{p}))$, for any direction of reflection $\mathbf{g}$. Thus a reflection event does not change the value of $\tilde{\pi}$. This means we can use the same argument as at the start of the proofs of Theorem 5.1 and Theorem 5.2 to get that the left-hand side of (5.4) is proportional to

$$-\mathbf{p} \cdot \nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{\theta}) + \lambda_{\mathrm{BPS}}(\boldsymbol{\theta}, \mathcal{R}_{\mathbf{g}}(\mathbf{p})) - \lambda_{\mathrm{BPS}}(\boldsymbol{\theta}, \mathbf{p}), \tag{5.13}$$

where to simplify notation we have used $\mathbf{g} = \nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{\theta})$. The middle term is the rate of transitioning to state $(\boldsymbol{\theta}, \mathbf{p})$ and uses, as shown above, that reflections are involutions, so it is the state $(\boldsymbol{\theta}, \mathcal{R}_{\mathbf{g}}(\mathbf{p}))$ that will transition to $(\boldsymbol{\theta}, \mathbf{p})$ at a reflection event. Substituting in the form of λ_{BPS}, and using the definition of $\mathbf{g}$, we have

$$\lambda_{\mathrm{BPS}}(\boldsymbol{\theta}, \mathcal{R}_{\mathbf{g}}(\mathbf{p})) - \lambda_{\mathrm{BPS}}(\boldsymbol{\theta}, \mathbf{p}) = \max\{0, \mathcal{R}_{\mathbf{g}}(\mathbf{p}) \cdot \mathbf{g}\} - \max\{0, \mathbf{p} \cdot \mathbf{g}\}.$$

Now

$$\mathcal{R}_{\mathbf{g}}(\mathbf{p}) \cdot \mathbf{g} = (\mathbf{p} - 2(\mathbf{p} \cdot \widehat{\mathbf{g}})\,\widehat{\mathbf{g}}) \cdot \mathbf{g} = \mathbf{p} \cdot \mathbf{g} - 2(\mathbf{p} \cdot \widehat{\mathbf{g}})(\widehat{\mathbf{g}} \cdot \mathbf{g}) = \mathbf{p} \cdot \mathbf{g} - 2(\mathbf{p} \cdot \mathbf{g})(\widehat{\mathbf{g}} \cdot \widehat{\mathbf{g}}),$$

where the last equality follows as $\mathbf{g}$ is proportional to $\widehat{\mathbf{g}}$. As $\widehat{\mathbf{g}}$ is a unit vector, we have $\mathcal{R}_{\mathbf{g}}(\mathbf{p}) \cdot \mathbf{g} = -\mathbf{p} \cdot \mathbf{g}$. Substituting gives

$$\lambda_{\text{BPS}}(\boldsymbol{\theta}, \mathcal{R}_{\mathbf{g}}(\mathbf{p})) - \lambda_{\text{BPS}}(\boldsymbol{\theta}, \mathbf{p}) = \max\{0, -\mathbf{p} \cdot \mathbf{g}\} - \max\{0, \mathbf{p} \cdot \mathbf{g}\} = -\mathbf{p} \cdot \mathbf{g}.$$

By definition of $\mathbf{g}$ this is just $\mathbf{p} \cdot \nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{\theta})$. Substituting this into (5.13) we see that the left-hand side of (5.4) is 0 as required.

□

Example: Sampling from a Gaussian Target

To gain an initial understanding of these algorithms in practice, how they differ from each other and how they compare to HMC, we will consider their implementation for sampling from a Gaussian distribution. This is an example where all methods can be implemented exactly and enables a simple comparison of the dynamics of the different samplers.

To simplify notation we will assume our target Gaussian distribution has a mean zero. This can be assumed without loss of generality in terms of the behaviour of the samplers, as we can re-centre any Gaussian distribution with non-zero mean and this would not change the samplers' dynamics. A mean-zero Gaussian distribution is commonly parameterised by its covariance matrix, $\boldsymbol{\Sigma}$ say, but in terms of its density function, it is easier to use the precision matrix, which is the inverse of the covariance matrix. We will denote the precision matrix by $\mathbf{Q} = \boldsymbol{\Sigma}^{-1}$. We assume that $\boldsymbol{\Sigma}$, and hence $\mathbf{Q}$, is positive-definite. Then up to an additive constant, we have

$$\log \pi(\boldsymbol{\theta}) = -\frac{1}{2}\boldsymbol{\theta}^{\top}\mathbf{Q}\boldsymbol{\theta}.$$

The rates of the PDMP samplers depend on π through $-\nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{\theta})$, which for the Gaussian target is $\mathbf{Q}\boldsymbol{\theta}$.

Before showing the output from the different PDMP samplers for this model, we will describe an approach to simulating the PDMPs. Each of the three PDMPs introduced in the previous section can be viewed as having multiple types of event, with each event having a deterministic transition. For the Zig–Zag Sampler, we have one event associated with each component of $\boldsymbol{\theta}$, and that flips the associated component of the velocity. For the Bouncy Particle Sampler, and the Coordinate Sampler, there are two events, one of which is a refresh of the velocity. Our approach to simulating these PDMPs is to use the idea of superposition, that is we will simulate the time for each of the possible events, find which occurs first, and then this type of event with its associated time is the next event for our PDMP.

As described in Section 5.2.2, to simulate times of events we need,

for each type of event, to calculate the rate of the time until the next event, given the current state. We will describe how to calculate this for the bounce event of the Bouncy Particle Sampler, and for a flip event in the Zig–Zag Sampler. Simulating the refresh events is trivial, and simulating the events of the Coordinate Sampler follows by similar arguments.

Let the current state of our PDMP be $\mathbf{z} = (\boldsymbol{\theta}, \mathbf{p})$, and let $\tilde{\lambda}_{\mathbf{z}}(t)$ be the rate at which an event occurs in terms of the future time t. We will first consider the bounce event of the Bouncy Particle Sampler. Let the current time be s, so $\mathbf{z}_s = \mathbf{z}$, then

$$\begin{aligned}\tilde{\lambda}_{\mathbf{z}}(t) &= \max\{0, \mathbf{p}_{s+t} \cdot (-\nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{\theta}_{t+s})\} = \max\{0, \mathbf{p}_{s+t}^{\top} \mathbf{Q} \boldsymbol{\theta}_{s+t}\} \\ &= \max\{0, \mathbf{p}^{\top} \mathbf{Q}(\boldsymbol{\theta} + t\mathbf{p})\} = \max\{0, \mathbf{p}^{\top} \mathbf{Q} \boldsymbol{\theta} + t\mathbf{p}^{\top} \mathbf{Q} \mathbf{p}\}.\end{aligned}$$

Here we have used the definition of $\tilde{\lambda}_{\mathbf{z}}(t)$, substituted in $\log \pi(\boldsymbol{\theta}_{t+s})$ for our target and then used the fact that up until the next event, $\mathbf{p}_{s+t} = \mathbf{p}_s = \mathbf{p}$ and $\boldsymbol{\theta}_{s+t} = \boldsymbol{\theta}_s + t\mathbf{p}_s = \boldsymbol{\theta} + t\mathbf{p}$.

We are viewing $\tilde{\lambda}_{\mathbf{z}}(t)$ as a function of the further time until the event, t. We can see that $\tilde{\lambda}$ is the maximum of zero and a linear function of t, and defining $a = \mathbf{p}^{\top} \mathbf{Q} \boldsymbol{\theta}$ and $b = \mathbf{p}^{\top} \mathbf{Q} \mathbf{p}$, the linear function is equal to $a + bt$. Furthermore, as $\mathbf{Q}$ is positive-definite we have $b > 0$. For this rate, we can simulate event times directly using Algorithm 7. To do this, we need to solve $\int_0^t \tilde{\lambda}_{\mathbf{z}}(u) \mathrm{d}u = w$. There are two cases for the integral. First, if $a > 0$, then $\tilde{\lambda}_{\mathbf{z}}(u) = a + bu$ for all $u > 0$, so

$$\int_0^t \tilde{\lambda}_{\mathbf{z}}(u) \mathrm{d}u = \int_0^t (a + bu) \mathrm{d}u = at + \frac{bt^2}{2},$$

and this is equal to $w > 0$ if $t = -a/b + \sqrt{a^2 + 2wb}/b$. If $a < 0$ then $a + bt$ is only positive for $t > |a|/b$, thus for such t

$$\int_0^t \tilde{\lambda}_{\mathbf{z}}(u) \mathrm{d}u = \int_{|a|/b}^t (a + bu) \mathrm{d}u = \int_0^{t-|a|/b} bu'^2 \mathrm{d}u' = \frac{b(t - |a|/b)^2}{2}.$$

This is equal to $w > 0$ when $t = |a|/b + \sqrt{2w/b}$.

The resulting algorithm for one iteration of the Bouncy Particle Sampler is given in Algorithm 9. The algorithm simulates the time of a refresh event and a bounce event. It sets the time of the next event to be the smaller of these two times, and updates the position of the state. Then, depending on which type of event occurred first, it updates the velocity. For a refresh event, this involves simulating from the sampler's invariant distribution for the velocity, which depending on the type of Bouncy Particle Sampler can be either a standard Gaussian distribution or the uniform distribution on

the unit hyper-sphere. For a bounce event, the velocity is reflected in the hyperplane perpendicular to $-\nabla \log \pi(\boldsymbol{\theta})$ at the current position $\boldsymbol{\theta}'$, which for our model is $\mathbf{Q}\boldsymbol{\theta}'$.

Algorithm 9: Bouncy Particle Sampler: Gaussian Target

Input: Precision Matrix, $\mathbf{Q}$, current state $(\boldsymbol{\theta}, \mathbf{p})$, refresh rate $\lambda_r > 0$
Calculate $a = \mathbf{p}^\top \mathbf{Q}\boldsymbol{\theta}$ and $b = \mathbf{p}^\top \mathbf{Q}\mathbf{p}$.
Simulate w_1 and w_2, independent realisations of a standard exponential random variable.
Calculate time until a refresh event $\tau_1 = w_1/\lambda_r$.
Calculate time until a bounce event: if $a < 0$ $\tau_2 = \sqrt{2w_2/b} + |a|/b$, otherwise $\tau_2 = -a/b + \sqrt{a^2 + 2w_2 b}/b$.
Calculate event time $t = \min\{\tau_1, \tau_2\}$.
Update position $\boldsymbol{\theta}' = \boldsymbol{\theta} = t\mathbf{p}$.
Decided on event type and update velocity:
if $\tau_1 < \tau_2$ **then**
  Refresh event. Simulate $\mathbf{p}'$ from its invariant distribution.
else
  Bounce event. Set

$$\mathbf{p}' = \mathbf{p} - 2(\mathbf{p}^\top \mathbf{Q}\boldsymbol{\theta}') \frac{\mathbf{Q}\boldsymbol{\theta}'}{\sqrt{\boldsymbol{\theta}'^\top \mathbf{Q}^\top \mathbf{Q}\boldsymbol{\theta}'}}.$$

end
Output: Time to next event t, and new state $(\boldsymbol{\theta}', \mathbf{p}')$

A similar derivation is possible for the Zig–Zag Sampler. The only difference is that the flip rate of the ith component of p_i is

$$\max\{0, -p_i(\nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{\theta}))_i\} = \max\{0, p_i(\mathbf{Q}\boldsymbol{\theta})_i\},$$

where we write, for example, $(\mathbf{Q}\boldsymbol{\theta})_i$ to denote the ith component of $\mathbf{Q}\boldsymbol{\theta}$. Thus, the associated rate as a function of time until this event, if the current state of $\mathbf{z} = (\boldsymbol{\theta}, \mathbf{p})$, is

$$\tilde{\lambda}_{\mathbf{z}}^{(i)}(t) = \max\{0, p_i[\mathbf{Q}(\boldsymbol{\theta} + t\mathbf{p})]_i\} = \max\{0, p_i(\mathbf{Q}\boldsymbol{\theta})_i + t p_i(\mathbf{Q}\mathbf{p})_i\}.$$

This again is the maximum of 0 and a linear function of t and can be simulated as described above. As above, we can set $\tilde{\lambda}_{\mathbf{z}}^{(i)}(t) = \max\{0, a + bt\}$ but now with $a = p_i(\mathbf{Q}\boldsymbol{\theta})_i$ and $b = p_i(\mathbf{Q}\mathbf{p})_i$. The only difference is that in this case, b can be negative. If $b < 0$ and $a < 0$, then this event can never

happen. If $a > 0$, then an event can happen for $t < a/|b|$. In this case

$$w = \int_0^t (a + bu)\mathrm{d}u \Rightarrow w = at + \frac{bt^2}{2}.$$

This has a solution for $t > 0$ only if $w \leq a^2/(2|b|)$, in which case $t = -a/b + \sqrt{a^2 + 2wb}/b$. If $w > a^2/(2|b|)$ then this event does not happen. If an event cannot or does not happen, then algorithmically we set the associated event time to infinity.

An algorithm for one iteration of the Zig–Zag Sampler is given in Algorithm 10. It has a similar form as the Bouncy Particle Sampler. We calculate the event time for each type of event – though due to the four possible cases we have not given the formulae for calculating the event times within the algorithm. We then set the event time to the smallest of these times and update the position. Finally, the event type is calculated and we apply the appropriate transition to the velocity, remembering that $F_i(\mathbf{p})$ flips the ith component of the vector $\mathbf{p}$.

Algorithm 10: Zig–Zag Sampler: Gaussian Target

Input: Precision Matrix, $\mathbf{Q}$, current state $(\boldsymbol{\theta}, \mathbf{p})$, refresh rate $\lambda_r > 0$
for $i = 1, \ldots, d$ **do**
  Calculate $a_i = p_i(\mathbf{Q}\boldsymbol{\theta})_i$ and $b_i = p_i(\mathbf{Q}\mathbf{p})_i$.
  Simulate w_i, a realisation of a standard exponential random variable.
  Calculate τ_i, the event time for a flip of the ith component of $\mathbf{p}$.
end
Calculate event time $t = \min_{i=1,\ldots,d}\{\tau_i\}$.
Update position $\boldsymbol{\theta}' = \boldsymbol{\theta} = t\mathbf{p}$.
Decide on event type, $i^* = \arg\min\{\tau_i\}$.
Update velocity $\mathbf{p}' = F_{i^*}(\mathbf{p})$.
Output: Time to next event t, and new state $(\boldsymbol{\theta}', \mathbf{p}')$

To gain some intuition of the properties of these PDMP algorithms, we will first look qualitatively at the output of running the algorithms for a bivariate Gaussian – as we can plot the realisations from the paths generated by the position component of the PDMPs. First, we look at the importance of the refresh rate with the Bouncy Particle Sampler. To most clearly see the potential issues for this sampler, it is helpful to observe it sampling from a target with uncorrelated, equal variance components. See Figure 5.2 for output from the Bouncy Particle Sampler for different refresh rates.

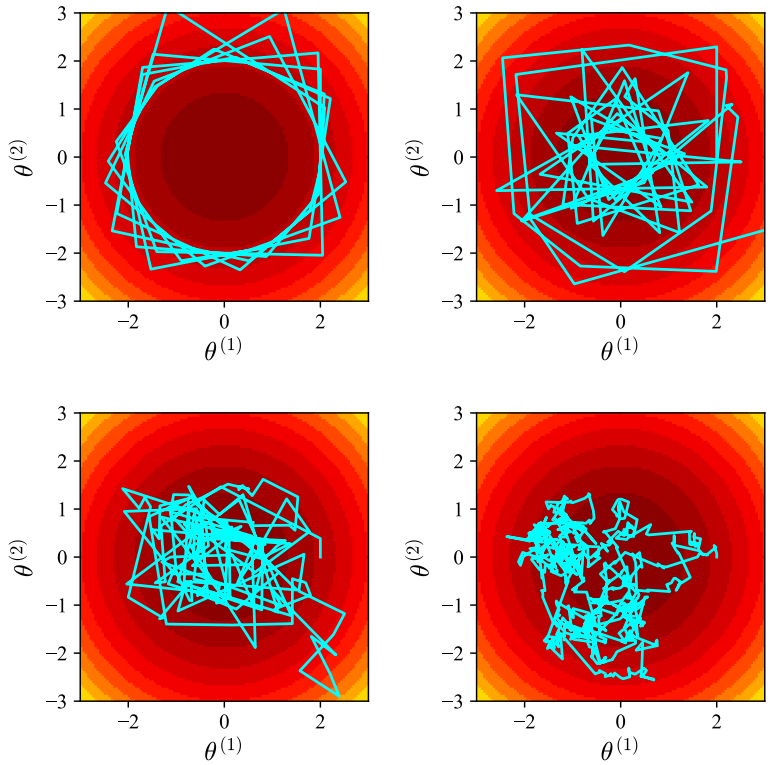

Figure 5.2 Plots of realisations of the trajectory or path of the Bouncy Particle Sampler when sampling from a standard (i.e. uncorrelated, equal variance) bivariate Gaussian distribution. Trajectories shown for no refresh events (top left), $\lambda_r = 0.1$ (top right), $\lambda_r = 1$ (bottom left) and $\lambda_r = 10$ (top right). The heat map shows the log posterior density of the target in each case.

The top-left plot shows the trajectory if we do not use any refresh events. We can clearly see evidence of the sampler being reducible – as the sampler does not enter a large region around the mode. Bouchard-Côté et al. (2018) prove that the Bouncy Particle Sampler is in fact reducible for this example. Furthermore, they show that this is avoided if we use any non-zero refresh rate, and if we do so the sampler will converge to the target distribution. However, we get very different behaviour for different values of the refresh rate, as shown in the remaining plots of Figure 5.2. A small rate produces a

sampler, that whilst irreducible, has poor mixing properties (see top-right plot). The sampler has long periods between refresh events, and for each of these periods, there are regions of the state space that the sampler cannot reach. Too large a refresh rate means that the sampler shows random-walk behaviour, which can also adversely affect its mixing (see bottom-right plot). Tuning of the refresh rate to a good intermediate value can result in a sampler that mixes well and avoids this random-walk behaviour (see bottom-left plot). We will return to how to tune the refresh rate later. Finally, whilst the Coordinate Sampler also has refresh events, it does not suffer from the same problems as the Bouncy Particle Sampler, and will be irreducible even if $\lambda_r = 0$.

We now compare the outputs of the three different PDMP samplers, with the Bouncy Particle Sampler using an appropriately tuned refresh rate, $\lambda_r = 1$. These are shown in Figure 5.3 together with the output from HMC. At this stage, there are two main points we wish to make. First, one can see the qualitative similarities and differences between the different PDMP samplers. Each sampler explores $\boldsymbol{\theta}$ space with straight-line trajectories, and they all have the property that they continue in the same direction whilst moving to areas of higher probability density – though for the Zig–Zag Sampler, this has to be interpreted separately for each axis component. However, the trajectories themselves are very different due to the different possible velocities and transitions. The Coordinate Sampler explores the posterior by exploring a single component of $\boldsymbol{\theta}$ at a time. This has some similarities with a Gibbs sampler (see Section 2.1.1), and intuition from results on mixing of Gibbs samplers suggest that this sampler will perform best when there is no strong correlation between the components of $\boldsymbol{\theta}$. The Zig–Zag Sampler has trajectories consisting of diagonal lines, and the transitions that flip a single component of the velocity lead to trajectories that look like zig-zags, which gives the sampler its name. Finally, the Bouncy Particle Sampler can have trajectories that explore the space in any direction.

Second, it is interesting to compare PDMP samplers with HMC (see bottom-right plot). For this model, we can solve the Hamiltonian dynamics for each proposal of the HMC algorithm exactly, and thus we always will accept a proposal. The trajectories of the Hamiltonian dynamics are elliptical, as compared to the straight-line segments of our PDMP samplers. However, the main difference is that the output of a PDMP sampler is a continuous path, whilst for HMC, we obtain a set of points. For non-Gaussian target distributions, the HMC sampler will not always accept the proposal which can lead to worse mixing. In such cases, whilst we cannot directly sample

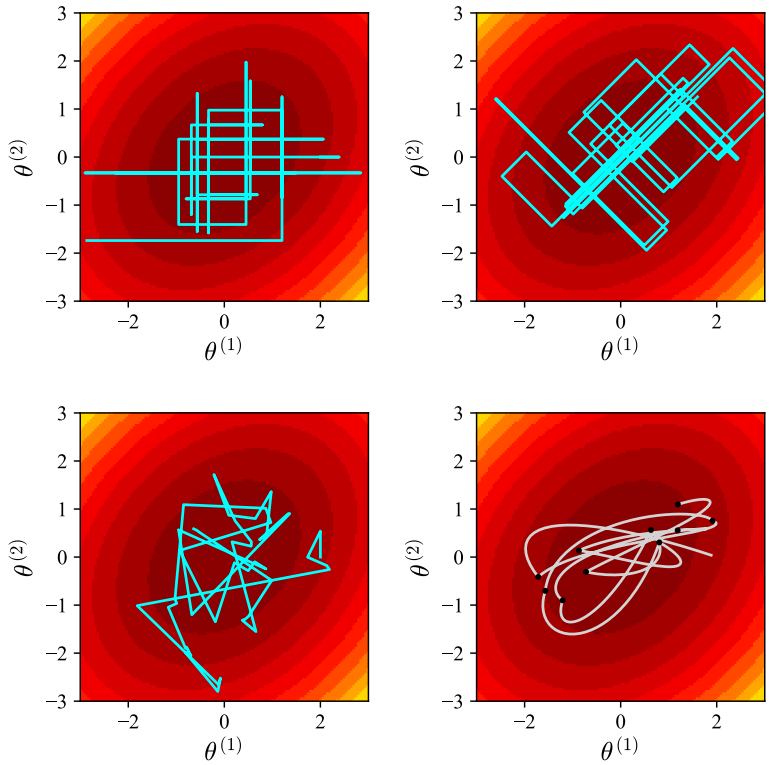

Figure 5.3 Comparison of three PDMP algorithms and HMC for sampling from a bivariate Gaussian with unit marginal variances and correlation of 0.5. Realisations of the trajectories (blue lines) of the Coordinate Sampler (top left), the Zig–Zag Sampler (top right) and the Bouncy Particle Sampler with $\lambda_r = 1$ (bottom left). Trajectories of Hamiltonian dynamics (line) and the sampled points (dots) from HMC (bottom right). The heat map shows the log posterior density of the target in each case.

the trajectories of the PDMP samplers, this only impacts the computational cost of simulation, rather than the output or the mixing properties of the sampler.

5.3.2 Use of PDMP Output

Simulating a PDMP, as described in Section 5.2.2, produces a skeleton of the sample path of the process. Assume that we have simulated the process up to some time T. We will denote this skeleton by the set $\{\tau_i, (\boldsymbol{\theta}_{\tau_i}, \mathbf{p}_{\tau_i})\}_{k=0}^{n+1}$ that gives the initial state of the process, with $\tau_0 = 0$, the time and state after each event, for $k = 1, \ldots, n$, and the final state of the process at time $\tau_{n+1} = T$. How do we use this output to approximate the target distribution?

For any Monte Carlo method, an approximation to the target distribution, $\pi(\boldsymbol{\theta})$ comes from the ability to estimate the expectation for arbitrary functions of $\boldsymbol{\theta}$ with respect to π. Thus consider estimating $\mathbb{E}_\pi[h(\boldsymbol{\theta})]$ for some function h for which this expectation exists. First, we describe how we do not estimate this expectation! We cannot just use the sample average of $h(\cdot)$ at the skeleton points for $\boldsymbol{\theta}$, even after allowing some burn-in. In general, the skeleton points will not be sampled from π at stationarity, as they will be biased towards values where there is a large average rate of an event occurring.

Instead, we need to estimate the expectation with respect to the continuous-time sample path, after allowing for a suitable burn-in. There are two ways of doing this. Assume we choose the burn-in to be some time S. Then one approach is to calculate the average value of the integral of $h(\boldsymbol{\theta}_t)$ for our sample path for $S < t \leq T$. This can be calculated from the skeleton as follows. First, we work out the value of $\boldsymbol{\theta}_S$ for our sample path. This is possible by finding l such that $\tau_l \leq S < \tau_{l+1}$, and using linear interpolation

$$\boldsymbol{\theta}_S = \frac{\tau_{l+1} - S}{\tau_{l+1} - \tau_l}\boldsymbol{\theta}_l + \frac{S - \tau_l}{\tau_{l+1} - \tau_l}\boldsymbol{\theta}_l.$$

Then our estimator can be calculated as

$$\hat{\mathbb{E}}_\pi[h(\boldsymbol{\theta})] = \frac{1}{T-S}\left(\int_S^{\tau_{l+1}} h\left(\boldsymbol{\theta}_S + (t-S)\frac{\boldsymbol{\theta}_{\tau_{l+1}} - \boldsymbol{\theta}_S}{\tau_{l+1} - S}\right)\mathrm{d}t + \sum_{k=l+1}^{K}\int_{\tau_k}^{\tau_{k+1}} h\left(\boldsymbol{\theta}_{\tau_k} + (t-\tau_k)\frac{\boldsymbol{\theta}_{\tau_{k+1}} - \boldsymbol{\theta}_{\tau_k}}{\tau_{k+1} - \tau_k}\right)\mathrm{d}t\right)$$

This estimator is only practical if we can analytically calculate the integrals along the linear segments of the path. A more general, and arguably simpler approach is to evaluate $\boldsymbol{\theta}_t$ at N evenly spaced points between S and T and then use standard Monte Carlo averages with respect to this set of values. That is, let $\delta = (T - S)/N$ and using linear interpolation as above evaluate

$\boldsymbol{\theta}_{S+j\delta}$ for $j = 1, \ldots, N$. Then our estimator of $\mathbb{E}_\pi [h(\boldsymbol{\theta})]$ would now be

$$\hat{\mathbb{E}}_\pi [h(\boldsymbol{\theta})] = \frac{1}{N} \sum_{j=1}^{N} h(\boldsymbol{\theta}_{S+j\delta}).$$

One advantage of this approach is the final output is similar to that for standard MCMC, which eases comparison and enables us to use methods for assessing the accuracy of standard MCMC estimators such as the integrated auto-correlation time and effective sample size (see Section 1.3.2).

5.3.3 Comparison of Samplers

Bierkens et al. (2022) examines the mixing properties of the Bouncy Particle Sampler and the Zig–Zag Sampler in the limit as the dimension of the space, $d \to \infty$, for the special case where the posterior of interest is a d-dimensional standard normal distribution. In related work, Bierkens et al. (2023a) investigates finite-dimensional normal targets where some principal components have a much smaller length scale than others. We summarise the findings of these two papers and provide some intuition for them.

To compare the samplers, we need to consider both their computational cost for simulating a trajectory of fixed duration, and how the mixing of the process depends on time. First, consider the computational cost, and the intensity of bounce events on a d-dimensional standard normal target, so that at stationarity $\nabla \log \pi(\boldsymbol{\theta}) = -\boldsymbol{\theta}$. The momentum for the Zig–Zag Sampler is $\mathbf{p}_{\text{ZZ}} \in \{-1, +1\}^d$. For the Bouncy Particle Sampler, we describe the case where $\mathbf{p}$ is sampled from U_d, the uniform distribution on the unit hypersphere. If, instead, $\mathbf{p}_{\text{BPS}} \sim \mathsf{N}(\mathbf{0}, \frac{1}{d}\mathbf{I}_d)$, for large d, $\|\mathbf{p}_{\text{BPS}}\|^2 \approx 1$ and the analysis is the same as for the setting where $\mathbf{p}_{\text{BPS}} \sim \mathsf{U}_d$. Speeding up time by a factor of $\sqrt{d}$ leads to the version of the BPS that samples $\mathbf{p} \sim \mathsf{N}(\mathbf{0}, \mathbf{I}_d)$, but makes no difference to the overall efficiency in terms of mixing per unit of computational effort.

For either sampler, let p_i be the ith component of its momentum. For the Bouncy Particle Sampler, the intensity is $\lambda_{\text{BPS}}(\boldsymbol{\theta}, \mathbf{p}) = \max\{0, \sum_{i=1}^d p_i\theta_i\} = O(1)$, since each term in the sum has the same expectation of 0 and a variance of $O(1/d)$. Thus, there are $O(1)$ events per unit of time. In contrast, for the Zig–Zag Sampler, the intensity is $\lambda_{\text{ZZ}}(\boldsymbol{\theta}, \mathbf{p}) = \sum_{i=1}^d \max\{0, p_i\theta_i\} = O(d)$, since each term in the sum has the same positive expectation. Hence, there are $O(d)$ events per unit time. In general, for each sampler, the computational cost of performing a bounce is $O(d)$: for the Bouncy Particle Sampler, this is the order of the cost of calculating the gradient required for

both the event rate and for calculating the new velocity at a bounce event; for Zig-Zag, after a bounce event we will need to update the rates for each of the d possible events (though see Section 5.4.2 for situations where this can be reduced). Thus the total cost per unit time is $O(d)$ for the Bouncy Particle Sampler and $O\left(d^2\right)$ for the Zig–Zag Sampler.

The above costing generalises to any reasonably well-behaved target. However, because it only updates a component at a time, in cases where components have a sparse conditional dependence graph the cost of performing a Zig–Zag bounce can be reduced to $O(1)$ (see Section 5.4.2).

Now we turn to the mixing properties. In this special case of an isotropic target, the state of the Markov process can be encapsulated by the radial component, $\|\boldsymbol{\theta}\|$, and the angle between the radius and the momentum, which is proportional to $\boldsymbol{\theta} \cdot \mathbf{p}$.

For Bouncy Particle Samplers, the radial component mixes in $O(d)$ time, whereas the angular component mixes in $O(1)$ time. To see why, for simplicity, we ignore any refresh events. Consider a single straight-line path between bounces, which we will call a segment: because the contours on a $\mathsf{N}(\mathbf{0}, \mathbf{I}_d)$ target are spherical, $\log \pi$ increases monotonically to a maximum along the segment and then decreases monotonically until the next bounce. Let E be the size of the drop in $\log \pi$ from its maximum until the next bounce. Start a clock at a time when $\log \pi$ is at a maximum and let t be the time since the clock started. Since there are no refresh events, if there has been no bounce since the clock started, the size of the total drop in $\log \pi$ by time t is

$$D(t) = -\int_0^t \mathbf{p} \cdot \nabla \log \pi(\boldsymbol{\theta}_s) \mathrm{d}s.$$

Now, for $t > 0$, $\pi(\boldsymbol{\theta}_t)$ decreases until a bounce occurs, so

$$E > D(t) \Leftrightarrow \text{"No bounce by time } t\text{."}$$

However, while $\log \pi$ is decreasing, bounce events follow an inhomogeneous Poisson process with a rate at time s of $\lambda(s) = -\mathbf{p} \cdot \nabla \log \pi(\boldsymbol{\theta}_s)$, so the probability that there has been no event by time t is

$$\exp\left[-\int_0^t \lambda(s) \mathrm{d}s\right] = \exp[-D(t)].$$

Thus $\mathbb{P}(E > D(t)) = \exp[-D(t)]$, so E has an exponential distribution with rate parameter 1.

Over the $O(1)$ time between bounces, the angle $\boldsymbol{\theta} \cdot \mathbf{p}$ moves monotonically from its most negative extent (just after a bounce) to its most positive extent

(just before the next bounce). Thus, $\boldsymbol{\theta} \cdot \mathbf{p}$ mixes in $O(1)$ time. However,

$$\log \pi(\boldsymbol{\theta}) = \text{constant} - \frac{1}{2}\|\boldsymbol{\theta}\|^2 = \text{constant} - \frac{1}{2}\chi_d^2$$

at stationarity, and a χ_d^2 random variable has a standard deviation of $\sqrt{2d}$. So, to mix, the $\log \pi$ process needs to move by $O(d^{1/2})$, yet it only moves by $O(1)$ in $O(1)$ time. In an analogous manner to the limit for the random walk Metropolis in Section 2.1.3, speeding up time by a factor of d leads to a limiting diffusion for $\|\boldsymbol{\theta}\|$; thus $\|\boldsymbol{\theta}\|$ mixes in $O(d)$ time.

By maximising the speed of the limiting diffusion for $\|\boldsymbol{\theta}\|$, the same analysis advises on tuning the refresh rate for the Bouncy Particle Sampler, λ_{r}. This suggests choosing λ_{r} so that the ratio of refresh to bounce events is ≈ 0.78.

In the case of an isotropic normal target, the Zig–Zag Sampler simplifies to d independent one-dimensional instances of Gustafson's algorithm (Section 4.3). Thus, each individual component mixes in $O(1)$ time, so both $\|\boldsymbol{\theta}\|$ and $\boldsymbol{\theta} \cdot \mathbf{p}$ mix in $O(1)$ time.

Multiplying the mixing times by the computational costs, we can define mixing costs. For Bouncy Particle Samplers, these are $O(d)$ for the angular component and $O(d^2)$ for the radial component, whereas the costs for the Zig–Zag Sampler are $O(d^2)$ for both components. If one follows the adage that a sampler is only as good as its worst-mixing component, this suggests that, at least for well-behaved targets, both algorithms have a similar efficiency.

What about the mixing of the Bouncy Particle Sampler for other functions of the state? Bierkens et al. (2022) show that it has the same $O(d^2)$ rate for marginal components of the state – i.e. if we are interested in θ_i for some i. This compares to results in Deligiannidis et al. (2021) which suggest that mixing costs for marginal components are $O(d)$. The difference in results comes from different choices of how the refresh rate depends on d. Bierkens et al. (2022) have a constant rate, whereas Deligiannidis et al. (2021) assume that the rate decays like $O(d^{-1/2})$. The latter choice improves mixing for marginal components but worsens the mixing of the radial component, so that as $d \to \infty$, the limiting process for the radial component is degenerate.

To see this in practice, we compared the performance of Zig–Zag and the Bouncy Particle Sampler at sampling from a Gaussian with $d = 20$ and $d = 1000$. Results are shown in Figure 5.4, where we scale the number of events simulated to be proportional to the dimension, d. The theory states that for this scaling we would expect similar mixing for the Zig–Zag sampler over the length of the simulation. We observe this qualitatively for

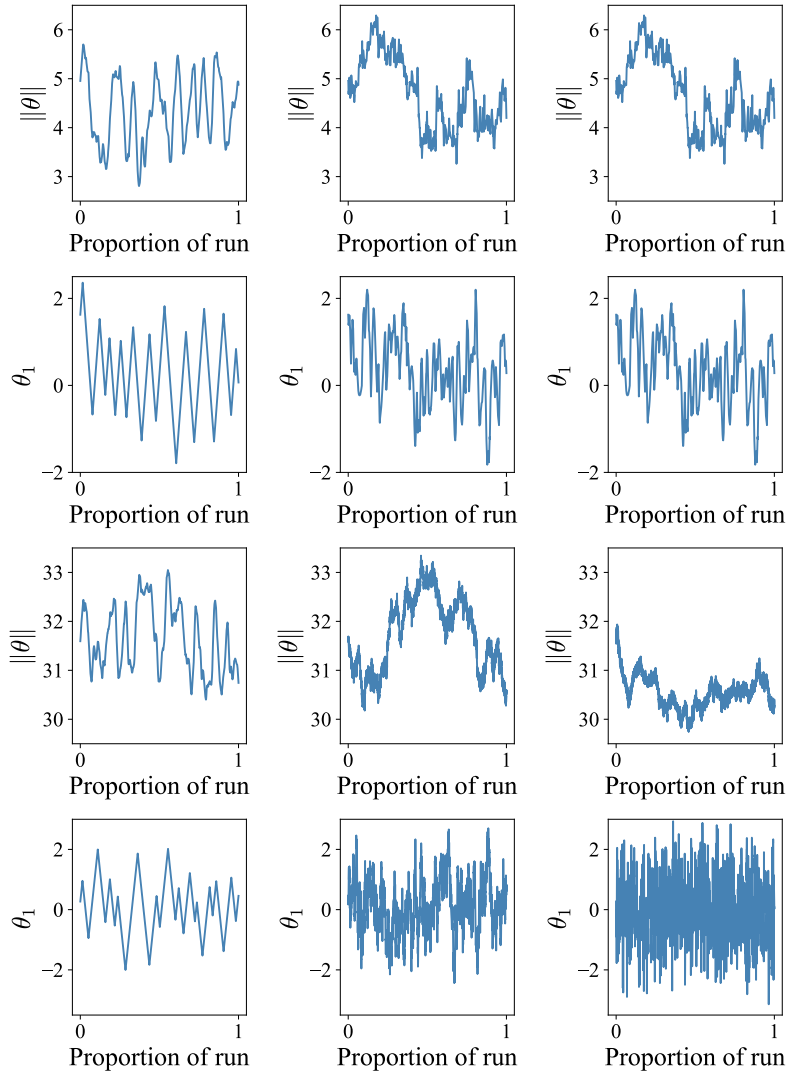

Figure 5.4 Trace plots for Zig–Zag (left-hand column), Bouncy Particle Sampler with $\lambda_r = 1.5\sqrt{d/20}$ (middle column) and Bouncy Particle Sampler with $\lambda_r = 1.5$ (right-hand column) for a Gaussian target with $d = 20$ (top two rows) and $d = 1000$ (bottom two rows). In each case we show trace plots for the radial component of the state, $\|\boldsymbol{\theta}\|$, and the first component of the state, θ_1. We ran all samplers for $20d$ bounce events, and scaled the time axis by the proportion of the resulting simulation time.

both $\|\boldsymbol{\theta}\|$ and θ_1. The theory also suggests similar qualitative behaviour for the Bouncy Particle Sampler with a refresh rate that scales with $\sqrt{d}$, so that the proportion of refresh events is roughly similar for different d; we observe this in the middle column of the plot. By comparison, if we use a fixed refresh rate, as in Deligiannidis et al. (2021), then we observe better mixing for θ_1 but worse mixing for $\|\boldsymbol{\theta}\|$ as we increase d – see the right-hand column of Figure 5.4.

One important consequence of the theoretical results, and that is seen in the results in Figure 5.4, is that care is needed when we assess mixing and convergence of, in particular, the Bouncy Particle Sampler – as we can get substantially different measures of mixing, such as auto-correlation time, depending on which function of the state we consider. Observing a fast mixing chain for one component may mask that other functions of the state are mixing very slowly (see Section 6.2.4 for more discussion on summarising convergence in multivariate settings and measures that can give differing importance to different coordinates).

Bierkens et al. (2023a) investigate fixed, finite-dimensional normal targets where between 1 and $d-1$ principal components have a length scale of ϵ, while the remainder have a length scale of 1. Both algorithms have $O(\epsilon^{-1})$ events per unit time because the momentum is $O(1)$ but some length scales are $O(\epsilon)$. The Bouncy Particle Sampler mixes in $O(1)$ time; however, the alignment of the Zig–Zag Sampler is crucial to its mixing time: if the principal axes of the target are aligned with the d Zig–Zag momentum components then it also mixes in $O(1)$ time, but in almost all other scenarios its mixing time is $O(\epsilon^{-1})$. Preconditioning is usually advised for any MCMC algorithm; this analysis highlights that the need for preconditioning the Zig–Zag Sampler is even more marked than it is for Bouncy Particle Samplers.

5.4 Efficient Simulation of PDMP Samplers

We now consider various approaches for simulating the PDMP samplers. As a running example that we will use to help explain some of the ideas, we will consider the logistic regression model with a Gaussian prior, which was introduced in Section 1.2.1.

5.4.1 Simulating PDMPs

When sampling from the Gaussian target, the rates that determine the time until the next event were linear and thus we could simulate the event times

of the PDMP exactly. What happens for more complicated targets where this is not possible? Here we describe three possible methods for simulating these events.

The most common approaches for simulating events of a PDMP are based on the idea of Poisson thinning (see Section 5.2.2). Remember this involves upper-bounding the event rate, simulating events with this bounding rate, and then accepting the simulated events with the ratio of the true rate to the bounding rate. The challenge with Poisson thinning is finding good upper bounds that are simple enough that we can simulate events analytically, and, ideally, close to the true rate, as the computational efficiency of Poisson thinning depends on how close the bounding rate is to the true rate. The first two methods we describe are based on this idea but differ in the assumptions they make on the target and how the bounding rates are constructed.

The first approach comes from Bierkens et al. (2019b) and assumes that the Hessian of the minus log-target is bounded. To simplify the notation, define the vector $\mathbf{U}(\boldsymbol{\theta}) = -\nabla \log \pi(\boldsymbol{\theta})$, so $\mathbf{U} = (U_1(\boldsymbol{\theta}), \ldots, U_d(\boldsymbol{\theta}))$ with

$$U_j(\boldsymbol{\theta}) = -\frac{\partial \log \pi(\boldsymbol{\theta})}{\partial \theta_j}.$$

Now the Hessian of $-\log \pi(\boldsymbol{\theta})$ is a $d \times d$ matrix $\mathbf{H}(\boldsymbol{\theta})$ defined as

$$\mathbf{H}(\boldsymbol{\theta}) = \begin{pmatrix} \nabla U_1(\boldsymbol{\theta}) & \cdots & \nabla U_d(\boldsymbol{\theta}) \end{pmatrix}.$$

Then we assume that there is some matrix $\mathbf{J}$ such that for any vector, $\mathbf{w}$, and any $\boldsymbol{\theta}$,

$$\mathbf{w}^\top \mathbf{H}(\boldsymbol{\theta}) \mathbf{w} \leq \mathbf{w}^\top \mathbf{J} \mathbf{w}.$$

This holds for Gaussian target distributions, as $\mathbf{H}(\boldsymbol{\theta}) = \mathbf{H}$ is a constant. More importantly, it holds for targets which are heavier-tailed than Gaussian, including the posterior distribution for logistic regression or many versions of robust regression if we have e.g. Gaussian priors on the parameters of the model.

Now, as introduced before, consider a rate for the next event, or the next specific type of event, in a PDMP, $\tilde{\lambda}_{\mathbf{z}}(t)$, in terms of the further time until the event t. Assume that

$$\tilde{\lambda}_{\mathbf{z}}(t) = \max\{0, \mathbf{w} \cdot \mathbf{U}(\boldsymbol{\theta} + \mathbf{p}t)\},$$

where $\mathbf{U} = -\nabla \log \pi$ as defined above, the current state is $(\boldsymbol{\theta}, \mathbf{p})$, and $\mathbf{w}$ is some vector that depends on the sampler. For the Bouncy Particle Sampler or the Coordinate Sampler, if we are considering time until the next non-refresh event, then $\mathbf{w} = \mathbf{p}$, whereas for the Zig–Zag Sampler if we are

considering the next flip of component j then $\mathbf{w}$ will be either the unit vector with 1 or -1 in the $j$the component and zero elsewhere.

Now consider the term $\mathbf{w} \cdot \mathbf{U}(\boldsymbol{\theta} + \mathbf{p}t)$. This can be rewritten as

$$\begin{aligned}\mathbf{w} \cdot \mathbf{U}(\boldsymbol{\theta} + \mathbf{p}t) &= \mathbf{w} \cdot \mathbf{U}(\boldsymbol{\theta}) + \sum_{i=1}^{d} w_i \int_0^t \frac{\mathrm{d}U_i(\boldsymbol{\theta} + \mathbf{p}s)}{\mathrm{d}s} \mathrm{d}s \\ &= \mathbf{w} \cdot \mathbf{U}(\boldsymbol{\theta}) + \sum_{i=1}^{d} w_i \int_0^t \mathbf{p} \cdot \nabla U_i(\boldsymbol{\theta} + \mathbf{p}s) \mathrm{d}s \\ &= \mathbf{w} \cdot \mathbf{U}(\boldsymbol{\theta}) + \int_0^t \mathbf{w}^\top \mathbf{H}(\boldsymbol{\theta} + \mathbf{p}s)\mathbf{p} \mathrm{d}s.\end{aligned}$$

The first equality comes from writing the value of a function at time t as its value at time 0 plus the integral of its derivative from time 0 to time t. We then use the chain rule to get the derivative of $U_i(\boldsymbol{\theta} + \mathbf{p}s)$ with respect to s, and finally the definition of the Hessian matrix.

Now, by Cauchy–Schwarz for vectors, $\mathbf{w}^\top \mathbf{H}\mathbf{p} \leq \|\mathbf{w}\| \|\mathbf{H}\mathbf{p}\|$, where $\|\mathbf{w}\| = (\sum_{i=1}^{d} w_i)^{1/2}$ is the L_2 norm of the vector $\mathbf{w}$. This, together with our assumption on the bound of the Hessian, gives

$$\mathbf{w}^\top \mathbf{H}(\boldsymbol{\theta} + \mathbf{p}s)\mathbf{p} \leq \|\mathbf{w}\| \|\mathbf{H}(\boldsymbol{\theta} + \mathbf{p}s)\mathbf{p}\| \leq \|\mathbf{w}\| \|\mathbf{J}\mathbf{p}\|$$

which is a constant. Thus we get the linear bound

$$\mathbf{w} \cdot \mathbf{U}(\boldsymbol{\theta} + \mathbf{p}t) \leq \mathbf{w} \cdot \mathbf{U}(\boldsymbol{\theta}) + \|\mathbf{w}\| \|\mathbf{J}\mathbf{p}\| t,$$

or equivalently the piecewise linear bound on the rate

$$\tilde{\lambda}_{\mathbf{z}}(t) \leq \max\{0, \mathbf{w} \cdot \mathbf{U}(\boldsymbol{\theta}) + \|\mathbf{w}\| \|\mathbf{J}\mathbf{p}\| t\}.$$

We can simulate events from this upper-bounding rate analytically, in an equivalent way to which we simulate the events for the Gaussian target. If $\mathbf{w} \neq \mathbf{p}$, we can use instead $\mathbf{w}^\top \mathbf{H}\mathbf{p} \leq \|\mathbf{p}\| \|\mathbf{H}\mathbf{w}\|$ to get the alternative bound

$$\tilde{\lambda}_{\mathbf{z}}(t) \leq \max\{0, \mathbf{w} \cdot \mathbf{U}(\boldsymbol{\theta}) + \|\mathbf{p}\| \|\mathbf{J}\mathbf{w}\| t\}.$$

Our second approach (Sutton and Fearnhead, 2023) is based on a different assumption for the target, namely that we can decompose $-\nabla \log \pi(\boldsymbol{\theta} + \mathbf{p}t)$, as a function of t for any $\boldsymbol{\theta}$ and $\mathbf{p}$ into the sum of convex and concave functions, and assume the concave function is differentiable everywhere. A function $f_\cup(t)$ for $t \geq 0$ is a convex function if for any $0 \leq r < s < t$, we have

$$f_\cup(s) \leq \frac{t-s}{t-r} f_\cup(r) + \frac{s-r}{t-r} f_\cup(t),$$

while a function $f_\cap(t)$ for $t \geq 0$ is a concave function if $-f_\cap(t)$ is convex.

That is if we pick any two points on the function and join them by a straight line, then the function lies below the line if it is convex, and above the line if it is concave. See Figure 5.5 for an example.

We will assume that we can decompose the rate until the next event as

$$\tilde{\lambda}_{\mathbf{z}}(t) = \max\{0, f_{\cup}(t) + f_{\cap}(t)\},$$

i.e. in terms of a single convex and single concave function – though the ideas below apply trivially if our decomposition involves multiple concave and convex functions. (In fact, the sum of convex functions is convex, and the sum of concave functions is concave, so we can immediately simplify the decomposition to the case we are considering.) Our starting point is the bound

$$\tilde{\lambda}_{\mathbf{z}}(t) = \max\{0, f_{\cup}(t) + f_{\cap}(t)\} \leq \max\{0, f_{\cup}(t)\} + \max\{0, f_{\cap}(t)\}. \quad (5.14)$$

Sutton and Fearnhead (2023) then use the fact that we can bound $f_{\cup}(t)$ and $f_{\cap}(t)$ by piecewise linear functions just by evaluating the functions at a set of grid points. Once we have these piecewise linear bounds, they immediately give us a piecewise linear bound on $\tilde{\lambda}_{\mathbf{z}}(t)$ by substituting them into (5.14).

How do we get piecewise linear bounds on $f_{\cup}(t)$ and $f_{\cap}(t)$? It is simplest to see this through a picture – see Figure 5.5. For the convex function, the bound comes immediately from the definition: if we evaluate $f_{\cup}(t)$ at $t = 0$ and $t = t_1$, then the straight line that joins these points gives an upper bound on $[0, t_1]$. For a concave function, we need to evaluate $f_{\cap}(t)$ and its derivative at $t = 0$ and $t = t_1$. We then construct the tangents to $f_{\cap}(t)$ at $t = 0$ and $t = t_1$, and the function lies below both tangents.

The above approach gives upper bounds on some interval $[0, t_1]$, and we can proceed by using these upper bounds to simulate events, if any, on $[0, t_1]$. If there are no events, we then choose some $t_2 > t_1$ and calculate an upper bound on $[t_1, t_2]$ and repeat the process. In practice, Sutton and Fearnhead (2023) suggest choosing $t_1, t_2, \ldots$ to be equally spaced, and suggest ways of choosing the spacing in an adaptive way that balances the number of times we do not simulate an event from the bounding process on an interval, against the number of times we simulate events from the bounding process that are not accepted. The idea is that if the intervals are too low, we waste time by having too small an interval and thus having to calculate the upper bound, whereas if the interval is too large then the upper bound can become loose and we waste time by simulating lots of events that are rejected. It is also possible to recycle calculations used to decide whether to accept an event to improve the bounds.

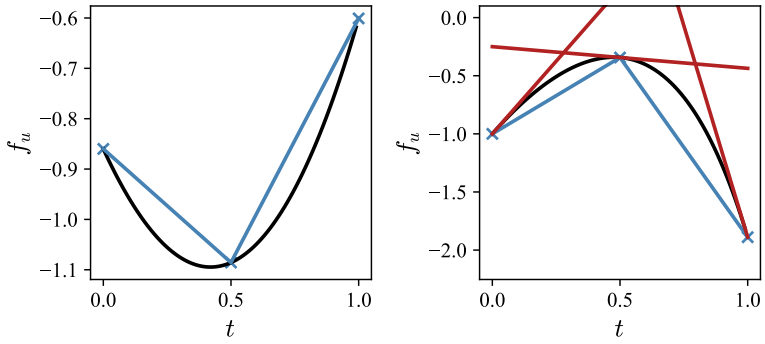

Figure 5.5 Example of a convex function (left) and a concave function (right). For a convex function, the straight line (shown in grey) that joins any two points will upper-bound the function between those two points. For a concave function, the straight line (shown in grey dashed) that joins any two points will lower-bound the function between those two points. For a concave function, we can upper-bound the function by any tangent to the function (shown in grey). Example bounds for $[0, 0.5]$ and $[0.5.1]$ are shown by the grey line (left-hand plot) and minimum of the grey lines (right-hand plot).

The final set of methods we will overview is based on simulating the events using numerical methods. There have been two distinct approaches that have been suggested. The first is based on the equation for directly simulating the event times. Remember that we can simulate the next event time for a process with rate $\tilde{\lambda}_{\mathbf{z}}(t)$ by simulating u, a realisation of a standard uniform random variable, and then finding τ the solution to

$$-\int_0^{\tau} \tilde{\lambda}_{\mathbf{z}}(s)\mathrm{d}s = \log(1-u).$$

The smallest solution, τ, of this equation, is the time until the next event. Pagani et al. (2020) suggest solving this equation numerically using Brent's method (Press et al., 2007). The other approach is to use numerical methods to find an upper bound. Corbella et al. (2022) suggest such a method, where they use Brent's algorithm to find the maximum of $\tilde{\lambda}_{\mathbf{z}}(t)$ on some interval $[0, t_1]$, then propose points from a constant rate set to this maximum and use Poisson thinning. If no event is simulated over the interval $[0, t_1]$ they repeat the process on the next interval.

The advantage of using such numerical methods is that they are fully general, that is they can be applied to any model in an automatic manner. The disadvantages are two-fold. First, computationally they can be slow, depending on the numerical methods used. The second is that numerical errors may lead to errors in the simulation of the dynamics of the PDMP, so that the resulting PDMP may no longer target the correct distribution. The hope is that any numerical error is small so that the PDMP will have a stationary distribution that is still close to the target distribution. Pagani et al. (2020) give results on how numerical error will impact the distribution that the PDMP is sampling from.

Example: Bounded Hessian for Logistic Regression

Logistic regression is one example where we have a bounded Hessian. To see this, let $\pi(\boldsymbol{\theta})$ be the posterior for the logistic regression model of Section 1.2.1 with a Gaussian prior. Then differentiating (1.5) gives

$$-\frac{\partial^2 \log \pi(\boldsymbol{\theta})}{\partial\theta_i \partial\theta_l} = \left[\boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{-1}\right]_{i,l} + \sum_{j=1}^{N} x_j^{(i)} x_j^{(l)} \left\{\frac{\exp\{\mathbf{x}_j^\top \boldsymbol{\theta}\}}{1+\exp\{\mathbf{x}_j^\top \boldsymbol{\theta}\}}\right\}\left\{\frac{1}{1+\exp\{\mathbf{x}_j^\top \boldsymbol{\theta}\}}\right\},$$

where the subscripts i, l denote the (i, l)th element of the corresponding matrix. Now, for any probability q we have $q(1-q) \leq 1/4$, so

$$\frac{\partial^2 \log \pi(\boldsymbol{\theta})}{\partial\theta_i \partial\theta_l} \leq \left[\boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{-1}\right]_{i,l} + \frac{1}{4}\sum_{j=1}^{N} x_j^{(i)} x_j^{(l)}.$$

If we introduce a $N \times d$ matrix $\mathbf{X}$ whose (j, l)th entry is $x_j^{(l)}$, then this gives a bound on the Hessian of $-\log \pi(\boldsymbol{\theta})$ that is $\mathbf{J} = \boldsymbol{\Sigma}^{-1} + (1/4)\mathbf{X}^\top\mathbf{X}$.

For the Bouncy Particle Sampler, the rate of the next bounce event, if the current state is $(\boldsymbol{\theta}, \mathbf{p})$, is

$$\max\{0, \mathbf{p} \cdot \nabla(-\log \pi(\boldsymbol{\theta} + t\mathbf{p})\} \leq \max\{0, -\mathbf{p} \cdot \nabla(\log \pi(\boldsymbol{\theta}) + \|\mathbf{p}\|\|\mathbf{J}\mathbf{p}\|t\},$$

by the above argument. For the Zig–Zag Sampler, the rate of the next flip of component i of the velocity is bounded above by, for example,

$$\max\left\{0, -p_i \frac{\partial \log \pi(\boldsymbol{\theta})}{\partial\theta_i} + \sqrt{d}\|\mathbf{J}\mathbf{e}_i\|t\right\}.$$

Example: Concave–convex Sampling for Bayesian Matrix Factorisation

Consider the Bayesian matrix factorisation model of Section 1.2.2. To simplify notation we will assume an improper uniform prior for the parameters

$\boldsymbol{\theta} = \{\mathbf{U}, \mathbf{V}\}$ and set $\sigma^2 = 1$. The resulting log-posterior is

$$\log \pi(\mathbf{U}, \mathbf{V}|\mathbf{Y}) = -\frac{1}{2}\left\{\sum_{i=1}^{n}\sum_{j=1}^{m}\left(Y_{ij} - \sum_{k=1}^{d} U_{ik}V_{kj}\right)^2\right\}.$$

As we will see, the event rates for this model for the Zig–Zag Sampler or the Bouncy Particle Sampler are polynomials of the time to an event, and such events can be simulated by concave–convex sampling. We will show this for the Zig–Zag Sampler, but the extension to the Bouncy Particle Sampler is simple.

Consider the rate for update $U_{i,l}$. This depends on the derivative of minus $\log \pi$, which is

$$-\frac{\partial \log \pi(\mathbf{U}, \mathbf{V}|\mathbf{Y})}{\partial U_{i,l}} = \sum_{j=1}^{m}\left(Y_{i,j} - \sum_{k=1}^{d} U_{i,k}V_{k,j}\right)V_{l,j}.$$

If the current state is given by a position $(\mathbf{U}, \mathbf{V})$ and a velocity $(\dot{\mathbf{U}}, \dot{\mathbf{V}})$ then the rate of an event as a function of the time to the next event t is

$$\begin{aligned}
&-\dot{U}_{i,l}\frac{\partial \log \pi(\mathbf{U} + t\dot{\mathbf{U}}, \mathbf{V} + t\dot{\mathbf{V}}|\mathbf{Y})}{\partial U_{i,l}} \\
&= \dot{U}_{i,l}\sum_{j=1}^{m}\left(Y_{i,j} - \sum_{k=1}^{d}(U_{i,k} + t\dot{U}_{i,k})(V_{k,j} + t\dot{V}_{k,j})\right)(V_{l,j} + t\dot{V}_{l,j}).
\end{aligned}$$

This is cubic in t, and it is easy to obtain a concave–convex decomposition of this rate. The convex function will be the sum of terms in the cubic expression that have positive coefficients, and the concave function will be the sum of terms with negative coefficients.

5.4.2 Exploiting Model Sparsity

One advantage of PDMP samplers is that they can take advantage of a certain type of sparsity in the target distribution to speed up computation. This is most easily described for the Zig–Zag Sampler, though similar ideas can be used for an adapted version of the Bouncy Particle Sampler (see the section on the Local Bouncy Particle Sampler in Bouchard-Côté et al., 2018).

We have described how we can simulate the Zig–Zag Sampler by simulating d event times, one for each possible transition of the velocity. We then implement the event that occurs first, and then resimulate the further times for each of the d possible events. We repeat this process multiple times in

order to simulate the skeleton of a realisation of the process. However, we can improve on this implementation if the event that occurs does not affect the rates of many of the other types of events. This can happen if the model has a form of sparsity in terms of

$$\frac{\partial \log \pi(\boldsymbol{\theta})}{\partial \theta_i} \tag{5.15}$$

only depending on a small number of the components of $\boldsymbol{\theta}$.

To describe the idea formally, it is helpful to introduce some notation. For $i = 1, \ldots, d$ let $\mathcal{S}_i \subset \{1, \ldots, d\}$ denote the components of $\boldsymbol{\theta}$ that (5.15) depends on. So if j is not in $\mathcal{S}_i$, then (5.15) will not change as we vary θ_j if we keep all other components of $\boldsymbol{\theta}$ fixed. This means that if we have an event that changes the ith component of $\boldsymbol{\theta}$, then this will only affect the future rate of events that change the jth component of $\boldsymbol{\theta}$ for $j \in \mathcal{S}_i$. By the Markov property of the PDMP, if the rates are unchanged, we can re-use the simulated event times for $j \notin \mathcal{S}_i$ if $j \neq i$. We obviously need to re-simulate the event that flips the ith component of the velocity even if $i \notin \mathcal{S}_i$. The resulting algorithm is shown in Algorithm 11, with the key part being that after each event we only re-simulate some of the event times, for other events we just update and re-use the previously simulated times.

As one example of the potential advantage of this algorithm, consider simulating a Gaussian target where the precision matrix is tri-diagonal. That is, the (i, j) entry of the precision matrix is zero if $|i - j| > 1$. Such a model occurs if there is some form of Markov or AR(1) structure to the components of $\boldsymbol{\theta}$. In this case, $\mathcal{S}_i = \{i - 1, i, i + 1\}$ for $i = 2, \ldots, d - 1$, with $\mathcal{S}_1 = \{1, 2\}$ and $\mathcal{S}_d = \{d - 1, d\}$. To understand the idea of Algorithm 11, imagine $d = 5$ say, and that we have simulated that the further time to the five possible events is 0.3, 0.7, 1.3, ∞ and 0.5. The event that occurs first corresponds to flipping the first component of the velocity. This change only affects the rate at which future events that affect the first and second components of the velocity occur. So we need to re-simulate the further time to the next event of these types. For the other three types of events, we just update the further time to take account of the fact that a time of length 0.3 has passed. Thus the events at which the third to fifth components of the velocity change will now occur after a further time period of 1.0, ∞ and 0.2, respectively.

In terms of the computational advantage of this scheme, in this example, if d is large then the computational cost per iteration involves resampling at most 3 event times rather than d event times. It is also possible to further improve on Algorithm 11 by using the fact that the order of occurrence of

Algorithm 11: Zig–Zag Sampler: Exploiting Sparsity

Input: Event rates for each type of event λ_i, initial state $(\boldsymbol{\theta}, \mathbf{p})$, simulation time T.

Set $s = 0$ and $k = 0$.

for $i = 1, \ldots, d$ **do**

  Simulate t_i the time until the next event that flips the ith component of the velocity.

end

while $s<T$ **do**

  Calculate further time to next event $t = \min_{i=1,\ldots,d}\{t_i\}$.

  Update position $\boldsymbol{\theta}_{s+t} = \boldsymbol{\theta}_s + t\mathbf{p}$.

  Decide on event type, $i^* = \arg\min\{t_i\}$.

  Update velocity $\mathbf{p}_{s+t} = F_{i^*}(\mathbf{p})$.

  Update time $s = s + t$.

  Store skeleton points: set $k = k + 1$, $\tau_k = s$ and $\boldsymbol{\theta}_{\tau_k} = \boldsymbol{\theta}_s$.

  Update further time to events:

  for $i = 1, \ldots, d$ **do**

    if $i \in \mathcal{S}_{i^*} \cup \{i^*\}$ **then**

      Simulate t_i the time until the next event that flips the ith component of the velocity.

    end

    else

      Set $t_i = t_i - t$.

    end

  end

end

Output: Skeleton of events $\{(\tau_k, \boldsymbol{\theta}_{\tau_k})\}_{k=0}^{n}$

events that we do not re-simulate will not change (see Bouchard-Côté et al., 2018) – and this can reduce the cost per iteration to be of the order of the number of event times that we need to re-simulate.

Example: Logistic Regression

When would this idea be useful for sampling from the posterior of our logistic regression model? The rate of an event that flips component i of the velocity depends on the θ_i derivative of $\log \pi(\boldsymbol{\theta})$, which from (1.5), is

$$-\left[\boldsymbol{\theta}^\top \boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{-1}\right]_i + \sum_{j=1}^{N} x_j^{(i)} \left\{ y_j - \frac{\exp\{\mathbf{x}_j^\top \boldsymbol{\theta}\}}{1 + \exp\{\mathbf{x}_j^\top \boldsymbol{\theta}\}} \right\}.$$

We need this to depend on only a small set of components of $\boldsymbol{\theta}$. This would require two things. First that $\boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{-1}$ is sparse so only a small number of entries of the ith row or column of $\boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{-1}$ are non-zero. Second, we would require that the observations for which $\theta_j^{(i)} \neq 0$ would, combined, only have a small number of components of the covariates that are non-zero. Formally, we can define the set of rates that we would need to update after a flip of component i of $\boldsymbol{\theta}$ as

$$\mathcal{S}_i = \left\{ k : (\boldsymbol{\Sigma}^{-1})_{i,k} \neq 0, \text{ or } \exists\, j \text{ such that } \mathbf{x}_j^{(i)} \mathbf{x}_j^{(k)} \neq 0 \right\}.$$

This can happen for models with random effects which are included within $\boldsymbol{\theta}$. If we set the random effects to be $\theta_1, \ldots, \theta_N$, then for $j \in \{1, \ldots, N\}$, $\mathbf{x}_j^{(j)} = 1$ and $\mathbf{x}_j^{(i)} = 0$ for $i \neq j$. If further, we have that the random effects are independent of each other and the other parameters, $\mathcal{S}_i$ will only include the parameters of the fixed effects.

5.4.3 Data Subsampling Ideas

One potential advantage of PDMP samplers in Bayesian statistics is that they can use subsampling ideas to reduce the computational cost per iteration. This was first suggested for the Zig–Zag Sampler by Bierkens et al. (2019b), though the ideas apply more widely.

The starting point is a more general observation that we can potentially simulate from a target distribution if we have an unbiased estimator of $\nabla \log \pi$. This is most easily seen for the Zig–Zag Sampler, and we will focus just on this case for simplicity. See Fearnhead et al. (2018) and related ideas for the local Bouncy Particle Sampler in Bouchard-Côté et al. (2018) for how this is generalised to other PDMPs.

The Zig–Zag Sampler has d possible types of event. Consider the ith such event. If the current position is $\boldsymbol{\theta}$ and the ith component of the velocity is p_i, then this component flips with a rate

$$\max\left\{0, -p_i \frac{\partial \log \pi(\boldsymbol{\theta})}{\partial \theta_i}\right\}.$$

The key property of this rate that means that the sampler targets $\pi(\boldsymbol{\theta})$ is that the difference in rate between a flip from p_i to $-p_i$ and the rate of the reverse event is

$$\max\left\{0, -p_i \frac{\partial \log \pi(\boldsymbol{\theta})}{\partial \theta_i}\right\} - \max\left\{0, p_i \frac{\partial \log \pi(\boldsymbol{\theta})}{\partial \theta_i}\right\} = -p_i \frac{\partial \log \pi(\boldsymbol{\theta})}{\partial \theta_i}.$$

In practice, for Zig–Zag, one of the two rates is equal to 0, and this is the

most efficient choice and corresponds to the Zig–Zag Sampler using the canonical rates (see Section 5.3.1).

Now imagine we have a family of vector-valued random variables $\mathbf{G}(\boldsymbol{\theta})$, such that $\mathbb{E}\left[\mathbf{G}(\boldsymbol{\theta})\right] = -\nabla \log \pi(\boldsymbol{\theta})$ for all $\boldsymbol{\theta}$. Then if we implement the same dynamics as the Zig–Zag Sampler, but with the rate of flipping the ith component of the velocity equal to

$$\mathbb{E}\left[\max\{0, p_i G_i(\boldsymbol{\theta})\}\right],$$

then this will also produce a PDMP sampler that targets π. To see this, as above, consider the difference in the rate of flipping p_i to $-p_i$ and the rate of the reverse event. This is

$$\begin{aligned}
&\mathbb{E}\left[\max\{0, p_i G_i(\boldsymbol{\theta})\}\right] - \mathbb{E}\left[\max\{0, -p_i G_i(\boldsymbol{\theta})\}\right] \\
&= \mathbb{E}\left[\max\{0, p_i G_i(\boldsymbol{\theta})\} - \max\{0, -p_i G_i(\boldsymbol{\theta})\}\right] \\
&= \mathbb{E}\left[p_i G_i(\boldsymbol{\theta})\right] = p_i \mathbb{E}\left[G_i(\boldsymbol{\theta})\right] = -p_i \frac{\partial \log \pi(\boldsymbol{\theta})}{\partial \theta_i},
\end{aligned}$$

where we have used the standard result $\max\{0, x\} - \max\{0, -x\} = x$, linearity of expectation, and the definition of the expectation of $\mathbf{G}$. This is precisely the condition we need on the rates for a PDMP Sampler with the Zig–Zag dynamics to target π, the only difference is the rates being used are no longer the canonical rates.

In order to use the Zig–Zag Sampler with these rates we need a way of simulating the events. This is more challenging than for standard Zig–Zag as we need to deal with the rates being defined implicitly by an expectation. To do this, the standard approach is to find a bounding $b_i(\boldsymbol{\theta})$ such that for any realisation of $\mathbf{G}(\boldsymbol{\theta})$ we have $p_i G_i(\boldsymbol{\theta}) \leq b_i(\boldsymbol{\theta})$. If we can find such a bound then we can still use Poisson thinning to simulate the events. Let the time until the next event which flips p_i be

$$\tilde{\lambda}_{\mathbf{z}}^{(i)}(t) = \mathbb{E}\left[\max\{0, p_i G_i(\boldsymbol{\theta} + t\mathbf{p})\}\right],$$

and the corresponding bounding rate, which is used to simulate potential events, be $b_i(\boldsymbol{\theta} + t\mathbf{p})$. Then Poisson thinning for events of rate $\tilde{\lambda}_{\mathbf{z}}^{(i)}(t)$ is possible by using the following steps:

(T0) Set current time to $s = 0$
(T1) Simulate the time $\tau > s$ of the next event a process with rate $\bar{\lambda}(t) = b_i(\boldsymbol{\theta} + t\mathbf{p})$.
(T2) Simulate g_i, a realisation of $G_i(\boldsymbol{\theta} + \tau\mathbf{p})$.
(T3) Accept the event time with probability $\max\{0, p_i g_i\}/b_i(\boldsymbol{\theta} + t\mathbf{p})$. Otherwise set $s = \tau$ and return to (T1).

To see that this is a valid Poisson thinning algorithm to simulate events with rate $\tilde{\lambda}_{\mathbf{z}}^{(i)}(t)$, we just need to calculate the probability of accepting an event in step (T3). By averaging over the possible realisation of g_i in step (T2) and using the fact that by definition for any g_i, the probability in step (T3) is less than or equal to 1, this is

$$\mathbb{E}\left[\max\{0, p_i G_i\}/b_i(\boldsymbol{\theta} + t\mathbf{p})\right] = \frac{\mathbb{E}\left[\max\{0, p_i G_i\}\right]}{b_i(\boldsymbol{\theta} + t\mathbf{p})} = \frac{\tilde{\lambda}_{\mathbf{z}}^{(i)}(t)}{b_i(\boldsymbol{\theta} + t\mathbf{p})},$$

as required.

How does this idea relate to the use of subsampling? Consider π being a posterior distribution, and suppose that $\log \pi$ can be written as a sum

$$\log \pi(\boldsymbol{\theta}) = \sum_{j=1}^{N} \log \pi_j(\boldsymbol{\theta}),$$

where $\log \pi_j$ for $j = 1, \ldots, N$ is $1/N$ times the log-prior plus the log-likelihood contributions from the jth data point. Then this gives a simple way of constructing an unbiased estimator of $-\nabla \log \pi(\boldsymbol{\theta})$, by simulating I uniformly on $\{1, \ldots, N\}$ and setting $\mathbf{G}(\boldsymbol{\theta})$ to $-N\nabla \log \pi_I(\boldsymbol{\theta})$.

The advantage of using such an unbiased estimator within the Zig–Zag Sampler is that at each iteration of the Poisson thinning algorithm used to simulate an event, i.e. step (T2) and (T3) above, we need to process only one data point. This gives a per-iteration saving of a factor of N over the standard Zig–Zag Sampler which requires calculating derivatives of $\log \pi$. However, there are additional costs to using this subsampling idea. First, often the bounds that we use for Poisson thinning will be larger if we use subsampling – as they have to bound the rate for all possible realisations of g_i. This will lead to more iterations of the Poisson thinning algorithm to simulate the PDMP for the same amount of (stochastic process) time. Second, as we are no longer using the canonical rates we will introduce more events, and this will lead to more random-walk-like behaviour and slower mixing. Empirical results in Bierkens et al. (2019b) suggest that the overall effect of these is to counteract the factor of N improvement in per-iteration cost.

So can subsampling ideas within PDMPs be beneficial? It turns out they can, but we must use a better, i.e. lower variance, estimator for $\mathbf{G}$. This can be done using control variate ideas that are common in SGLD (see Section 3.3.1). We first run an optimisation algorithm, such as SGD, to find the mode or a value close to the mode of $\log \pi$. Denote this value by $\widehat{\boldsymbol{\theta}}$. Then

we can write

$$\log \pi(\boldsymbol{\theta}) = \log \pi(\widehat{\boldsymbol{\theta}}) + \sum_{j=1}^{N} \{\log \pi_j(\boldsymbol{\theta}) - \log \pi_j(\widehat{\boldsymbol{\theta}})\}.$$

So an unbiased estimator can be obtained by first sampling I uniformly on $\{1, \ldots, N\}$ and then setting $\mathbf{G}(\boldsymbol{\theta})$ to

$$-\log \pi(\widehat{\boldsymbol{\theta}}) - N\{\nabla \log \pi_I(\boldsymbol{\theta}) - \nabla \log \pi_I(\widehat{\boldsymbol{\theta}})\}.$$

Importantly the term $-\log \pi(\widehat{\boldsymbol{\theta}})$ is a constant, so requires a single up-front $O(N)$ cost to calculate it. Then evaluating a realisation of this estimator has $O(1)$ cost. If the Hessian of $-\log \pi$ is bounded, then Bierkens et al. (2019b) show that we can obtain the bounds needed to simulate the Zig–Zag Sampler using this unbiased estimator using the linear bounds described for bounded Hessian targets in Section 5.4.1. In this case, for a fixed accuracy of the final Monte Carlo sample, we can obtain a speed-up by a factor of N, after finding $\widehat{\boldsymbol{\theta}}$ and calculating $-\log \pi(\widehat{\boldsymbol{\theta}})$, relative to the standard Zig–Zag Sampler.

Example: Subsampling for Logistic Regression

To further explain how to implement the Zig–Zag Sampler with subsampling, we will consider the logistic regression model. To simplify exposition we will assume that we have an improper flat prior, so in the notation above $\pi_0(\boldsymbol{\theta}) \propto 1$. Thus we can drop the contribution of the prior to π and we have $\pi(\boldsymbol{\theta}) \propto \prod_{j=1}^{N} \pi_j(\boldsymbol{\theta})$ with

$$\pi_j(\boldsymbol{\theta}) = \left(\frac{\exp\{y_j \mathbf{x}_j^\top \boldsymbol{\theta}\}}{1 + \exp\{\mathbf{x}_j^\top \boldsymbol{\theta}\}} \right),$$

where y_j is the binary response and $\mathbf{x}_j$ is the vector of covariates for the jth observation.

Taking the first derivatives of $-\log \pi(\boldsymbol{\theta})$ gives

$$-\frac{\partial \log \pi_j(\boldsymbol{\theta})}{\partial \theta_i} = x_j^{(i)} \left\{ \frac{\exp\{\mathbf{x}_j^\top \boldsymbol{\theta}\}}{1 + \exp\{\mathbf{x}_j^\top \boldsymbol{\theta}\}} - y_j \right\}.$$

Using $0 \leq \exp(a)/(1 + \exp(a)) \leq 1$, we have that the modulus of this derivative is bounded by $x_j^{(i)}$. Thus if we use the unbiased estimator

$$G_i(\boldsymbol{\theta}) = -N \frac{\partial \log \pi_I(\boldsymbol{\theta})}{\partial \theta_i}, \quad I \text{ uniformly distributed on } \{1, \ldots, N\},$$

then we can bound the rate of an event by

$$b_i(\boldsymbol{\theta}) = N \max_{j=1,\ldots,N} |x_j^{(i)}|.$$

In the following, we will call the resulting sampler Zig–Zag with subsampling.

How about if we use control variates? Fix $\widehat{\boldsymbol{\theta}}$, and consider the estimator of the gradient

$$G_i^{(CV)}(\boldsymbol{\theta}) = -\frac{\partial \log \pi(\widehat{\boldsymbol{\theta}})}{\partial \theta_i} - N\left(\frac{\partial \log \pi_I(\boldsymbol{\theta})}{\partial \theta_i} - \frac{\partial \log \pi_I(\widehat{\boldsymbol{\theta}})}{\partial \theta_i}\right),$$

with again, I is uniformly distributed on $\{1, \ldots, N\}$. To obtain appropriate bounds for $\mathbf{p}^{(j)}\mathbf{G}_{CV}^{(j)}(\boldsymbol{\theta})$, consider the second derivatives

$$-\frac{\partial^2 \log \pi_j(\boldsymbol{\theta})}{\partial \theta_i \partial \theta_l} = x_j^{(i)} \mathbf{x}_j^{(l)} \left\{ \frac{\exp\{\mathbf{x}_j^\top \boldsymbol{\theta}\}}{(1+\exp\{\mathbf{x}_j^\top \boldsymbol{\theta}\})\}^2} \right\}.$$

As before using that $0 \leq \exp(a)/(1+\exp(a))^2 \leq 1/4$, we can bound the modulus of this second derivative by $(1/4)|\mathbf{x}_j^{(i)}\mathbf{x}_j^{(l)}|$. This gives the following bound

$$\begin{aligned}
&\left|\frac{\partial \log \pi_j(\boldsymbol{\theta}+t\mathbf{p})}{\partial \theta_i} - \frac{\partial \log \pi_j(\widehat{\boldsymbol{\theta}})}{\partial \theta_i}\right| \\
&\leq \left|\frac{\partial \log \pi_j(\boldsymbol{\theta}+t\mathbf{p})}{\partial \theta_i} - \frac{\partial \log \pi_j(\boldsymbol{\theta})}{\partial \theta_i}\right| + \left|\frac{\partial \log \pi_j(\boldsymbol{\theta})}{\partial \theta_i} - \frac{\partial \log \pi_j(\widehat{\boldsymbol{\theta}})}{\partial \theta_i}\right| \\
&\leq \frac{1}{4}|x_j^{(i)}|\|\mathbf{x}_i\|\left(\|\boldsymbol{\theta} - \widehat{\boldsymbol{\theta}}\| + t\|\mathbf{p}\|\right), \qquad (5.16)
\end{aligned}$$

where $\|\cdot\|$ denotes the Euclidean norm. The latter inequality comes from (i) bounding the change in a function by the size of the change in the argument times a bound on the gradient in the direction of the change; and (ii) as the gradient $\partial \log \pi_j / \partial \theta_i$ is bounded by $(1/4)|\mathbf{x}_j^{(i)}\mathbf{x}_j^{(l)}|$ in the lth direction, we can bound the dot-product of the gradient with a vector $\mathbf{v}$ by

$$\sum_{l=1}^{d} \frac{1}{4}|x_j^{(i)} x_j^{(l)} v_l| = \frac{1}{4}|x_j^{(i)}| \sum_{l=1}^{d} |x_j^{(l)} v_l| \leq \frac{1}{4}|x_j^{(i)}|\|\mathbf{x}_i\|\|\mathbf{v}\|,$$

with the last step using Cauchy–Schwarz.

Using (5.16), we get the following linear bound on the rate of events

when we use control variates.

$$\begin{aligned}&\max\{0, p_i G_i^{(CV)}(\boldsymbol{\theta} + t\mathbf{p})\}\\ &\leq \max\left\{0, -p_i \frac{\partial \log \pi(\widehat{\boldsymbol{\theta}})}{\partial \theta_i} + \frac{1}{4} N M_i \left(\|\boldsymbol{\theta} - \widehat{\boldsymbol{\theta}}\| + t\|\mathbf{p}\|\right)\right\},\end{aligned}$$

where $M_i = \max_{j=1,\ldots,N}(|x_j^{(i)}|\|\mathbf{x}_j\|)$. In the following, we will call the resulting sampler Zig–Zag with control variates.

To demonstrate how subsampling works in practice and the scaling of the methods with sample size we will compare three PDMP samplers for logistic regression with sample sizes of $N = 100$ and $N = 900$. These are the standard Zig–Zag Sampler, Zig–Zag with subsampling and Zig–Zag with control variates. In particular, we want to give intuition about the different impacts of the computational cost for estimating gradients, the efficiency of the Poisson thinning bounds on simulating events, and the mixing of the PDMP on the three algorithms.

Results are shown in Figure 5.6. There are a number of points to draw out. First, we see the potential advantage of subsampling in reducing the cost per iteration of the samplers. If this is dominated by accessing and calculating gradients for each data point, then the subsampling versions of Zig–Zag are able to propose many more event times (as seen by the larger number of blue dots for the middle and bottom rows). However, this in itself does not lead to a more accurate MCMC method for the same computational cost – as the drawback of subsampling is that the bounds used for Poisson thinning are worse. About two proposed events are needed for one actual event with standard Zig–Zag, but this reduces to over 10 proposed events for the two samplers which use subsampling.

One impact of this is, if we consider $N = 100$ then the standard Zig–Zag process is simulated for stochastic process time of 80 time units, and this is only increased to 130 time units for Zig–Zag with subsampling and 160 time units for Zig–Zag with control variates. Moreover, the Zig–Zag processes which use subsampling have a higher overall rate of events, particularly when control variates are not used. This leads to more random-walk behaviour for Zig–Zag with subsampling (see middle row) which worsens the mixing of the sampling. This is less of an issue when we use control variates.

Next, consider the scaling of the algorithms by comparing $N = 100$ with $N = 900$. First, the posterior for $N = 900$ is more concentrated, but if we re-scale axes (as has been done in the plots) then the posterior contours and the dynamics of standard Zig–Zag are similar for the two cases. However, if

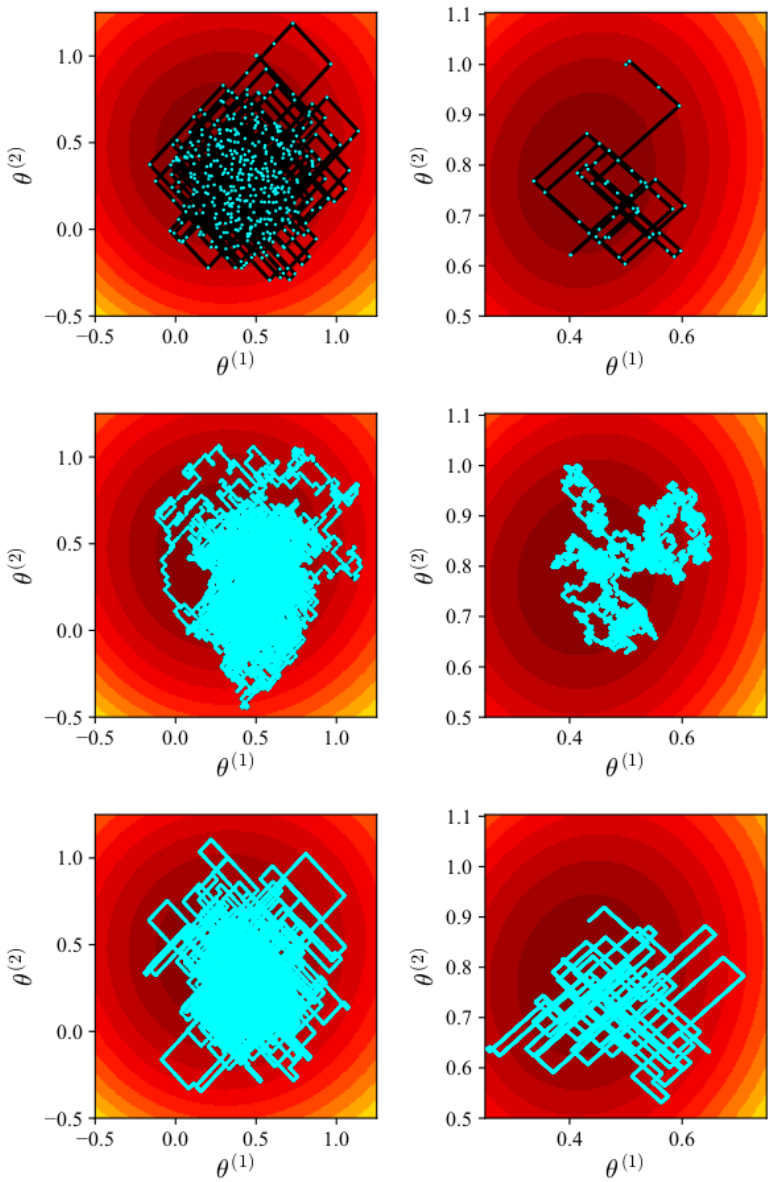

Figure 5.6 Example output of Zig–Zag Samplers for logistic regression: standard Zig–Zag (top row), Zig–Zag with subsampling (middle row) and Zig–Zag with control variates (bottom row). For each plot, the heat map shows the contours of $\log \pi$, the black line shows the trajectory of Zig–Zag and the blue dots show the points where we propose a possible event. In all cases, we ran Zig–Zag for the same number of data-point gradient calculations. As the top row does not involve subsampling, this means we propose N-times as many events for the middle and bottom rows as for the top row. Results for $N = 100$ (left-hand column) and $N = 900$ (right-hand column).

we have fixed the computational resource, then the impact on standard Zig–Zag is that we can only simulate the process for a shorter period – in this case simulating 9 times fewer actual events. For Zig–Zag with subsampling, we are able to propose the same number of events, but the higher rate of events and the looser bound for Poisson thinning means that we are only able to simulate a trajectory that is only 4 times as long as for the standard Zig–Zag, despite proposing 900 times as many events. Moreover, we see that the trajectory is increasingly diffusive and thus the overall exploration of the state-space of this sampler qualitatively looks no better than for standard Zig–Zag. This property is shown more rigorously by Bierkens et al. (2019b), who show that the scaling with N of the standard Zig–Zag and Zig–Zag with subsampling is similar. Moreover, their empirical results suggest that for the same computational cost, standard Zig–Zag is more accurate.

By comparison, we see better mixing behaviour for Zig–Zag with control variates. The sampler is able to simulate a trajectory that is approximately ten times as long as the standard Zig–Zag. Moreover, the mixing looks qualitatively similar to that of the algorithm for $N = 100$ and to that of the standard Zig–Zag for $N = 900$. This is again shown more rigorously in Bierkens et al. (2019b), where results suggest that the accuracy as measured, say, by effective sample size per CPU cost scales by a factor of N better for Zig–Zag with control variates than for the other two samplers. However, Zig–Zag with control variates does need an additional pre-processing step to find the mode, or a value near the mode, of the posterior and to calculate the gradient of the log posterior at this estimate of the mode. This improved scaling has been termed *super-efficiency*, and the fact that we can only achieve super-efficiency after a pre-processing step is shown theoretically by Johndrow et al. (2020).

5.5 Extensions

The PDMP samplers we have considered so far are appropriate for sampling from continuous densities that are differentiable almost everywhere and are based on specific constant velocity dynamics. We now describe recent work at generalising the samplers: to allow sampling from discontinuous targets; introduce reversible-jump moves to allow sampling from targets defined across spaces of different dimension; and generalise the velocity space and the constant velocity dynamics.

5.5.1 Discontinuous Target Distribution

The PDMP samplers we have described can sample from target densities that are differentiable everywhere. It is also easy to see that they are suitable for densities that are non-differentiable, providing the set of points where the density is not differentiable is a null set. However, as described, they cannot be used for densities that are not continuous everywhere.

It is possible to extend PDMP samplers so they are suitable for many discontinuous target densities. The idea is to use standard PDMP dynamics in regions where the target is continuous, and then add additional dynamics whenever the PDMP sampler reaches a point of discontinuity. This approach has been suggested by Bierkens et al. (2018) for continuous densities, but defined only on a compact region, and by Chevallier et al. (2021) in more generality. We will outline the basic idea and give some examples of appropriate dynamics at points of discontinuity for different samplers.

Assume our target density $\pi(\boldsymbol{\theta})$ can be defined in terms of a set of continuous densities, $\pi^{(i)}$, each constrained to a set of open regions E_i of $\mathbb{R}^d$, for $i = 1, \ldots, K$ for some K. Let Γ be the set of $\boldsymbol{\theta}$ points that lie on the boundary of one or more regions, and assume this is a null-set with respect to Lebesgue measure of $\mathbb{R}^d$. We assume that E_i and Γ partition $\mathbb{R}^d$, so that any $\boldsymbol{\theta} \in \mathbb{R}^d$ lies in precisely one of $E_1, \ldots, E_K, \Gamma$. For $\boldsymbol{\theta} \notin \Gamma$, our target is

$$\pi(\boldsymbol{\theta}) = \pi^{(i)}(\boldsymbol{\theta}) \quad \text{for } \boldsymbol{\theta} \in E_i.$$

The simplest example of such a density will be for $\boldsymbol{\theta}$ constrained to some compact region, E_1. In this case we have Γ as the boundary of E_1 and E_2 is the complement of $E_1 \cup \Gamma$, with $\pi^{(2)}(\boldsymbol{\theta}) = 0$ for $\boldsymbol{\theta} \in E_2$.

The idea is that we can define a PDMP sampler that is appropriate for each $\pi^{(i)}$, but need to now define what happens when the $\boldsymbol{\theta}$ component of the state tries to leave E_i. To explain this, whilst keeping the notation simple, we will consider the case of $K = 2$ regions, and describe conditions for the PDMP sampler on the boundary between E_1 and E_2 that are sufficient for it to target $\pi(\boldsymbol{\theta})$ as defined above. Extending this to $K > 2$ is trivial as we just apply the same conditions to each boundary between two regions. (We do not need to consider behaviour at points that lie on the boundary between three or more regions as the sampler will not hit such points with probability 1.)

For $\boldsymbol{\theta} \in \Gamma$, let $\mathbf{n}(\boldsymbol{\theta})$ be the normal to the boundary and assume that the normal is defined to point out of region E_1. Assume we have a PDMP sampler with velocity set $\mathcal{V}$ and with stationary distribution for the velocity component that is independent of $\boldsymbol{\theta}$ and is denoted by $\pi_{\mathbf{p}}$. Once we hit the

boundary, the velocity will determine whether the state is moving out of E_1 and into E_2 or vice-versa. To distinguish these two possibilities, define for each $\boldsymbol{\theta} \in \Gamma$

$$\mathcal{V}_{\boldsymbol{\theta}}^{+} = \left\{\mathbf{p} \in \mathcal{V} : \mathbf{n}(\boldsymbol{\theta})^{\top}\mathbf{p} > 0\right\}, \text{ and } \mathcal{V}_{\boldsymbol{\theta}}^{-} = \left\{\mathbf{p} \in \mathcal{V} : \mathbf{n}(\boldsymbol{\theta})^{\top}\mathbf{p} < 0\right\}.$$

So, for example, if the state is $(\boldsymbol{\theta}, \mathbf{p})$ for $\boldsymbol{\theta} \in \Gamma$ and $\mathbf{p} \in \mathcal{V}_{\boldsymbol{\theta}}^{+}$ then the sampler was in region E_1 and is moving into E_2.

Now define a family of probability density, or probability mass, functions for $\mathbf{p} \in \mathcal{V}$, as

$$\ell_{\boldsymbol{\theta}}(\mathbf{p}) = \begin{cases} |\mathbf{n}(\boldsymbol{\theta})^{\top}\mathbf{p}|\, \pi_{\mathbf{p}}(\mathbf{p})\pi^{(2)}(\boldsymbol{\theta}) & \text{if } \mathbf{p} \in \mathcal{V}_{\boldsymbol{\theta}}^{+} \\ |\mathbf{n}(\boldsymbol{\theta})^{\top}\mathbf{p}|\, \pi_{\mathbf{p}}(\mathbf{p})\pi^{(1)}(\boldsymbol{\theta}) & \text{otherwise} \end{cases}$$

This is just proportional to the density $\pi_{\mathbf{p}}(\mathbf{p})$ weighted by the size of the velocity in the direction of the normal $\mathbf{n}(\boldsymbol{\theta})$ and weighted by the density at $\boldsymbol{\theta}$ in the region that the sampler is moving to.

Finally, define a family of transition kernels for the velocity component, $\mathbb{Q}_{\boldsymbol{\theta}}^{b}(\mathbf{p}' \in \cdot|\mathbf{p})$ for each $\boldsymbol{\theta} \in \Gamma$. The following theorem, taken from Chevallier et al. (2021), gives appropriate dynamics for our PDMP sampler on the boundary.

Theorem 5.4 *Assume $\pi_{\mathbf{p}}$ is symmetric, so $\pi_{\mathbf{p}}(\mathbf{p}) = \pi_{\mathbf{p}}(-\mathbf{p})$, and that for each $\boldsymbol{\theta} \in \Gamma$ the transition kernel $\mathbb{Q}_{\boldsymbol{\theta}}^{b}$ has $\ell_{\boldsymbol{\theta}}$ as its invariant distribution. Then a PDMP sampler with:*

(i) dynamics for $\boldsymbol{\theta} \in E_i$, for $i = 1, 2$, that have invariant distribution $\pi^{(i)}(\boldsymbol{\theta})\pi_{\mathbf{p}}(\mathbf{p})$; and
(ii) for $\boldsymbol{\theta} \in \Gamma$ has a transition that keeps $\boldsymbol{\theta}$ unchanged but that:
(B1) flips the velocity $\mathbf{p}' = \mathbf{p}$;
(B2) updates the velocity according to $\mathbb{Q}_{\boldsymbol{\theta}}^{b}$, i.e. $\mathbf{p}'' \sim \mathbb{Q}_{\boldsymbol{\theta}}^{b}(\cdot, \mathbf{p}')$; and
(B3) updates the state of the PDMP to $(\boldsymbol{\theta}, \mathbf{p}'')$;

will have invariant distribution $\pi(\boldsymbol{\theta})\pi_{\mathbf{p}}(\mathbf{p})$.

Steps (B1) – (B3) of the theorem give appropriate dynamics for the velocity when we hit the boundary, with (B2) stated in terms of a transition kernel that has $\ell_{\boldsymbol{\theta}}$ as its invariant distribution.

There are various possible choices of dynamics due to different choices for the transition kernel. A trivial choice for $\mathbb{Q}_{\boldsymbol{\theta}}^{b}$ is the identity map. In this case, the transition at the boundary is to reverse the sign of the velocity, which means that the sampler will never leave the region that it starts with. Thus this choice is only suitable for the case where we start the sampler in E_1 and where $\pi^{(2)}(\boldsymbol{\theta}) = 0$, i.e. there is no probability mass in E_2. Even in

this case, this choice may not be a good one, as it will force the sampler to retrace its steps once it hits the boundary, which will slow down mixing.

An alternative choice is to define $\mathbb{Q}^b_{\boldsymbol{\theta}}$ to be independent of the current velocity, and just to involve sampling from $\ell_{\boldsymbol{\theta}}$. The problem with this is that sampling from $\ell_{\boldsymbol{\theta}}$ may be difficult. For both the Coordinate Sampler and the Zig–Zag Sampler, $\ell_{\boldsymbol{\theta}}$ is a discrete distribution, and can be calculated exactly. For the Coordinate Sampler, $\mathbf{p}$ can take $2d$ possible values, and this approach can be reasonable. However, for the Zig–Zag Sampler, $\mathbf{p}$ can take 2^d possible values, and thus calculating and simulating from $\ell_{\boldsymbol{\theta}}$ is prohibitive unless d is small. In cases where $\ell_{\boldsymbol{\theta}}$ is difficult to sample from one can instead use Metropolis–Hastings to define a transition kernel that has the required invariant distribution. This involves proposing a new velocity from an arbitrary proposal distribution and then accepting or rejecting that proposal. The problem with this is that if the acceptance probability is not high then we are likely to reject the proposal, and our new velocity will just be minus the velocity with which we hit the boundary, from step (B1), which will inhibit mixing. A partial solution is to define $\mathbb{Q}^b_{\boldsymbol{\theta}}$ to involve $L > 1$ Metropolis–Hastings steps, though this comes with an increased computational cost of sampling from $\mathbb{Q}^b_{\boldsymbol{\theta}}$.

For the Bouncy Particle Sampler, there is a simple and very natural choice of dynamics at the boundary which satisfies the condition of Theorem 5.4. If $\mathbf{p} \in \mathcal{V}^+_{\boldsymbol{\theta}}$, so we are currently moving out of E_1 then the dynamics are:

(R1) With probability $\min\{1, \pi^{(2)}(\boldsymbol{\theta})/\pi^{(1)}(\boldsymbol{\theta})\}$ the velocity is unchanged, i.e. $\mathbf{p}'' = \mathbf{p}$;

(R2) Otherwise, we reflect the velocity in the tangent to the boundary at $\boldsymbol{\theta}$, i.e. using the notation introduced for reflections $\mathbf{p}'' = \mathcal{R}_{\mathbf{n}(\boldsymbol{\theta})}(\mathbf{p})$.

If $\mathbf{p} \in \mathcal{V}^-_{\boldsymbol{\theta}}$, then the dynamics are as above but with $\pi^{(1)}$ and $\pi^{(2)}$ interchanged in (R1). Under these dynamics, if the sampler is moving to a region of higher probability density, it continues. If not, then with some probability it continues, otherwise it reflects back. A special case where we have $\pi(\boldsymbol{\theta})$ defined only on the compact region E_1, so that $\pi^{(2)}(\boldsymbol{\theta}) = 0$, in which case the sampler will always reflect if it hits the boundary.

We cannot apply similar dynamics for the Zig–Zag Sampler unless the boundary aligns appropriately with the velocity axes, as the reflected velocity in step (R2), $\mathcal{R}_{\mathbf{n}(\boldsymbol{\theta})}(\mathbf{p})$, may no longer be a valid velocity. However, Chevallier et al. (2021) show that the above dynamics for the Bouncy Particle Sampler can be viewed as the behaviour of the Bouncy Particle Sampler if we approximate the discontinuous target by a continuous one, by smoothing out the discontinuity, but then consider the limit where we allow the

transition between $\pi^{(1)}$ and $\pi^{(2)}$ to occur more quickly. By the same strategy, we can construct an appropriate transition kernel for Zig–Zag, see Chevallier et al. (2021) for more details.

Example: Logistic Regression with Constraints

As an example application for this algorithm, consider the logistic regression model but with constraints on the parameters. There are two natural constraints, one is that we may know that the effect of a covariate is such that larger values will increase, say, the probability of observing a response of 1. If the ith component of the $\mathbf{x}_j$s is such a covariate, then $\theta_i \geq 0$. Similarly, if an increase in the ith component of the covariate vector is known to lead to a decrease in the probability, then $\theta_i \leq 0$. Alternatively, we may have some constraints on the size of the effect, e.g. $\theta_i \geq \theta_l$.

For the Bouncy Particle Sampler, the simplest way of including such a constraint is to calculate the time when the current deterministic dynamics will first violate the constraint. If an event does not happen by this time, then we just reflect the velocity off the boundary of $\boldsymbol{\theta}$ space implied by the constraint. For the constraint $\theta_i \geq 0$ or $\theta_i \leq 0$ this would involve flipping the ith component of the velocity. For the constraint $\theta_i \geq \theta_l$, this involves the reflection $\mathcal{R}_{\mathbf{g}}$ with $\mathbf{g}$ defined so $g_i = 1$, $g_l = -1$ and all other entries set to zero.

In this example, we can use the same adaption to the Zig–Zag Sampler, as for each constraint the reflection at the boundary will produce a new velocity that is valid for the sampler. However, this would not be the case if, for example, we wanted to enforce a constraint such as $\theta_1 > 2\theta_2$.

5.5.2 Reversible Jump PDMP Samplers

Another class of distributions that the standard PDMP samplers are not suitable for are distributions defined on a set of spaces of different dimensions. In the following, we think of this in terms of a distribution over a discrete set of models, with a continuous density for each model. Whilst PDMPs can be used to explore the distribution defined for each model, we would need a way of moving between these models – which would lead to a type of reversible jump version of PDMP samplers.

One approach to do this is to add additional events that introduce discrete jumps from one space to another. Such moves have been proposed for other continuous-time samplers, see Grenander and Miller (1994), Phillips and Smith (1996) and Stephens (2000). However, they can be hard to implement due to challenges with simulating these moves with the correct rate.

A computationally more efficient procedure exists if the different models are defined in terms of some common d-dimensional parameter $\boldsymbol{\theta}$, but each model fixes one or more components of $\boldsymbol{\theta}$ to a specific value. The most common example of such a distribution is the posterior distribution for the coefficients of a linear or generalised linear regression model, where different models correspond to including different sets of covariates in the model. Thus we can define $\boldsymbol{\theta}$ to be the coefficients for the full set of covariates, and a given model will fix the coefficients of covariates not in the model to zero.

To motivate the form of the PDMP samplers we will introduce, consider first the case where we have a single covariate, so $d = 1$. Also, ignore having any intercept in the linear or generalised linear model. We now have two models, depending on whether we include the covariate or not. If $\ell(\theta)$ denotes the likelihood as a function of θ and if we have a prior that is a mixture of point mass at 0, which we will denote by $\delta_0(\theta)$, and a prior defined on $\mathbb{R}$, $\nu(\theta)$, then the posterior distribution is

$$\pi(\theta) \propto w\delta_0(\theta)\ell(\theta) + (1-w)\nu(\theta)\ell(\theta),$$

where w is the prior probability of excluding the covariate. Whilst we cannot use a standard PDMP to sample from this target, we can approximate the prior by replacing the point mass at zero with a distribution concentrated around zero. If we use a Gaussian density with variance σ^2 for $\sigma \approx 0$, then we have the target

$$\pi_\sigma(\theta) \propto w\mathsf{N}(\theta; 0, \sigma^2)\ell(\theta) + (1-w)\nu(\theta)\ell(\theta), \tag{5.17}$$

where $\mathsf{N}(\theta; 0, \sigma^2)$ denotes the density of a Gaussian with mean 0 and variance σ^2. We can now use a PDMP to sample from this posterior distribution. Also by letting $\sigma \to 0$ we have that this posterior will tend, in some sense, to the posterior with the point mass at 0. So a natural approach is to consider the dynamics of the PDMP targeting π_σ and whether we get well-defined dynamics in the limit of $\sigma \to 0$.

Figure 5.7 shows the dynamics for three different values of σ. We see that in each case the sampler will spend periods of time in the neighbourhood of 0. As we reduce σ, these neighbourhoods concentrate closer to 0 but the time the sampler spends in them seems to be similarly distributed. Away from the neighbourhood, the dynamics are those of sampling from a density proportional to $\nu(\theta)\ell(\theta)$. This suggests the limiting behaviour would be that of a PDMP targeting $\nu(\theta)\ell(\theta)$ but that if θ_t hits zero then the sampler stays at zero for a random amount of time.

Chevallier et al. (2023) propose such a PDMP sampler, and show that if

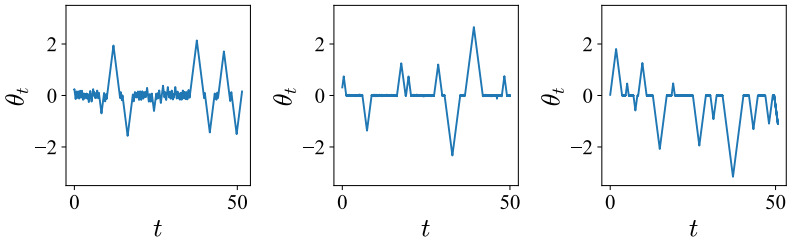

Figure 5.7 Example output from PDMP sampler for a target that is an equal mixture of a $\mathsf{N}(0, 1)$ distribution and a $N(0, \sigma^2)$ distribution for $\sigma = 0.1$ (left), $\sigma = 0.01$ (middle) and $\sigma = 0.001$ (right). As $\sigma \to 0$ the trajectories qualitatively converge to periods at, or close to 0, and periods where the trajectory is governed by the $\mathsf{N}(0, 1)$ distribution. This limit is the form for the reversible jump PDMP sampler. The computational advantage of using the limiting dynamics is that it avoids simulating the larger number of events close to 0 – which was of the order of 10000 for the right-hand plot.

we specify the dynamics as given below it will target the correct distribution across models. To describe the sampler for a d-dimensional parameter $\boldsymbol{\theta}$, assume that

$$\pi(\boldsymbol{\theta}) = \sum_{k=1}^{2^d} \pi_k(\boldsymbol{\theta}),$$

where we assume that each model k corresponds to a different set of components of $\boldsymbol{\theta}$ being set to zero. Here, $\pi_k(\boldsymbol{\theta})$ is, up to proportionality, the density of $\boldsymbol{\theta}$ associated with the kth model. These densities must be defined up to a common constant of proportionality across all models.

For model k, let $\mathcal{S}_k$ be the active set of model k, that is the set of indices of components of $\boldsymbol{\theta}$ that are non-zero for model k. The idea of the PDMP is that if we are currently exploring model k we will simulate a PDMP that targets $\pi_k(\boldsymbol{\theta})$, and that has non-zero velocity only for components of $\boldsymbol{\theta}$ that are in $\mathcal{S}_k$. But in addition, we allow two further events. The first is that if a component of $\boldsymbol{\theta}$ hits zero, and we denote this component by i, then we move to the model with active set $\mathcal{S}_k \backslash \{i\}$. The second is that for each $j \notin \mathcal{S}_k$ we have a rate of moving to the model with active set $\mathcal{S}_k \cup \{j\}$. If we move to such a model we do not change $\boldsymbol{\theta}$, but simulate a new velocity $\mathbf{p}$, which will have a non-zero velocity for component j.

To define such a sampler we just need to specify the rate of adding

a component to the model, and the distribution of the velocity after the transition. We will describe these for the Zig–Zag Sampler and the Bouncy Particle Sampler. For general results, and generalisations of this sampler which does not always move between models when a component of $\boldsymbol{\theta}$ hits zero, see Chevallier et al. (2023).

For the Zig–Zag Sampler, assuming we are in model k with current state $(\boldsymbol{\theta}, \mathbf{p})$, we have the following process for adding a variable to the model.

(Add–ZZ) For each $j \notin \mathcal{S}_k$, move to model k' with active set $\mathcal{S}_k \cup \{j\}$ at rate $\pi_{k'}(\boldsymbol{\theta})/\pi_k(\boldsymbol{\theta})$. Set the new velocity, $\mathbf{p}'$, such that $p'_i = p_i$ for $i \neq j$ and p'_j is drawn uniformly at random from $\{-1, 1\}$.

For the Bouncy Particle Sampler, with standard Gaussian distribution for the velocity, again assuming we are in model k, with current state $(\boldsymbol{\theta}, \mathbf{p})$, we have the following

(Add–BPS) For each $j \notin \mathcal{S}_k$, move to model k' with active set $\mathcal{S}_k \cup \{j\}$ at rate

$$\frac{2}{\sqrt{2\pi}} \frac{\pi_{k'}(\boldsymbol{\theta})}{\pi_k(\boldsymbol{\theta})}.$$

Set the new velocity, $\mathbf{p}'$, such that $p'_i = p_i$ for $i \neq j$ and $p'_j = x$ is simulated from a distribution with density function proportional to

$$|x| \exp\left\{-\frac{1}{2}x^2\right\},$$

this is a standard normal density function scaled by $|x|$.

The intuition for the density of the new velocity component for the Bouncy Particle Sampler is that it is skewed, relative to its invariant distribution, to larger absolute values of the velocity as these correspond to velocities that would hit zero more quickly.

Importantly for both samplers, the rates at which we add components will often be simple. If our target distribution is defined as a posterior distribution, with common likelihood for each model, then the likelihood components of the posteriors will cancel and the rates will just depend on the ratio of priors. For many priors, the distribution of each component of $\boldsymbol{\theta}$ will be independent, in which case these rates become constant.

For the Zig–Zag Sampler, one can improve on this sampler by remembering the velocity of each inactive component prior to it becoming inactive. Then, when that component is re-introduced to the model we re-use the same velocity. This is the Sticky Zig–Zag Sampler of Bierkens et al. (2023b). It

can mix better as it ensures that the dynamics of each component of $\boldsymbol{\theta}$ reflects less often.

Example: Logistic Regression with Model Choice

An extension to the logistic regression model of Section 1.2.1 is to include a choice as to which covariates to include in the model. We will consider two example priors, and calculate the rate of adding a covariate to the model for the Zig–Zag Sampler in each case.

A common prior would be to assume independence across covariates, so $\theta_i = 0$ with probability w_i and is drawn from a normal distribution with mean 0 and variance σ_i^2 with probability $1 - w_i$. In this case, because of the independence, if covariate i is not in the current model, the rate at which we add it will not depend on the value of θ_l for $l \neq i$. Thus, this rate will be constant and equal to

$$\frac{(1-w_i)}{w_i}\frac{1}{\sqrt{2\pi\sigma_i^2}},$$

the ratio of the prior probability of a model which includes covariate i to the prior probability of the same model without covariate i, times the prior probability density of $\theta_i = 0$ under the former model.

What about when we have a prior under which components of $\boldsymbol{\theta}$ are dependent? Assume we are currently in model k which does not include the ith covariate, and that adding this covariate will produce model k'. Let q_k and $q_{k'}$ be the prior probability of the two models and assume that they both have Gaussian priors on $\boldsymbol{\theta}$ with mean 0 and covariance matrices on the active components of $\boldsymbol{\theta}$ denoted by $\boldsymbol{\Sigma}_k$ and $\boldsymbol{\Sigma}_{k'}$ respectively. Let a_k be the number of active components of model k, with $a_{k'} = a_k + 1$. We can introduce matrices $\mathbf{A}_k$ and $\mathbf{A}_{k'}$ so that the prior for the active components of $\boldsymbol{\theta}$ under our prior for model k is

$$\left(\frac{1}{2\pi}\right)^{a_k/2} \det(\boldsymbol{\Sigma}_k)^{-1/2} \exp\left\{-\frac{1}{2}\boldsymbol{\theta}^\top \mathbf{A}_k \boldsymbol{\theta}\right\}.$$

This is possible by padding $\mathbf{A}$ with zeroes, so if covariate l is not in the model then $A_{lj} = A_{jl} = 0$, for $j = 1, \ldots, d$.

With this definition of the prior, the rate of moving from model k to k' as a function of $\boldsymbol{\theta}$ becomes

$$\frac{q_{k'}}{q_k}\left(\frac{1}{2\pi}\right)^{1/2}\left(\frac{\det(\boldsymbol{\Sigma}_k)}{\det(\boldsymbol{\Sigma}_{k'})}\right)^{1/2} \exp\left\{-\frac{1}{2}\boldsymbol{\theta}^\top(\mathbf{A}_{k'} - \mathbf{A}_k)\boldsymbol{\theta}\right\}.$$

Define C to be the constant before the exponential term. If our current state is $(\boldsymbol{\theta}, \mathbf{p})$ then the rate until we move to model k' is

$$C \exp\left\{-\frac{1}{2}(\boldsymbol{\theta}^\top + t\mathbf{p})(\mathbf{A}_{k'} - \mathbf{A}_k)(\boldsymbol{\theta} + t\mathbf{p})\right\} =$$

$$C \exp\left\{-\frac{1}{2}\boldsymbol{\theta}^\top(\mathbf{A}_{k'} - \mathbf{A}_k)\boldsymbol{\theta}\right\} \exp\left\{-\boldsymbol{\theta}^\top(\mathbf{A}_{k'} - \mathbf{A}_k)\mathbf{p}t - \frac{1}{2}\mathbf{p}^\top(\mathbf{A}_{k'} - \mathbf{A}_k)\mathbf{p}t^2\right\}.$$

This is of the form $a \exp\{bt + ct^2\}$ for some constants a, b and c. We can simulate the time of the next event with this rate if $c \leq 0$ as the integral of the rate is analytic for $c = 0$, and can be expressed in terms of probabilities of a normal distribution for $c < 0$. For $c > 0$, we can use Poisson thinning with e.g. bounds of the form $A \exp\{Bt\}$ over suitable intervals for t.

5.5.3 More General Velocity Models

Another possible way of extending PDMP samplers is to consider more general models for the dynamics. There are two simple ways of doing this, the first is to alter the distribution of the velocities for the Bouncy Particle Sampler or the Zig–Zag Sampler so that they are not spherically symmetric. The other is to consider non-constant velocity models. We will briefly describe each of these in turn.

First, we will focus on the Zig–Zag Sampler. If we let $\mathbf{e}_i$ be the ith unit vector, i.e. the vector whose ith component is 1 and all other components are 0, then the set of velocities of the Zig–Zig are the velocities of the form $\sum_{i=1}^d a_i \mathbf{e}_i$, where $a_i \in \{-1, 1\}$ for $i = 1, \ldots, d$. That is, they are the set of velocities that one obtains by adding plus or minus each unit vector. The rate of flipping a_i is equal to the maximum of 0 and minus the dot product of $a_i \mathbf{e}_i$ with $\nabla \log \pi(\boldsymbol{\theta})$.

To generalise this we just need to replace the unit vectors with another set of vectors that span $\mathbb{R}^d$. Denote this set by $\mathbf{p}_1, \ldots, \mathbf{p}_d$. Importantly for Zig–Zag, the change to the process is trivial – as the event rates are of a similar form but with $\mathbf{e}_1, \ldots, \mathbf{e}_d$ replaced with $\mathbf{p}_1, \ldots, \mathbf{p}_d$. There are two natural approaches to choosing $\mathbf{p}_1, \ldots, \mathbf{p}_d$. One is to just change the speed in each direction, so $\mathbf{p}_i = c_i \mathbf{e}_i$ for some set of positive scalars $c_1, \ldots, c_d$. This can be helpful if different components of $\boldsymbol{\theta}$ under π are on different scales. A natural choice is to set c_i to be an estimate of the marginal standard deviation of $\boldsymbol{\theta}_i$ under π. The other approach is to also change the directions of the velocities as well. If we have an estimate of the variance-matrix of $\boldsymbol{\theta}$ under π, say $\boldsymbol{\Sigma}$, then one choice is to choose the $\mathbf{p}_i$s to be the eigenvectors of $\boldsymbol{\Sigma}$.

To see why this is a natural choice, consider $\pi(\boldsymbol{\theta})$ being a Gaussian distribution with variance $\boldsymbol{\Sigma}$. Centre this distribution so it has a mean of 0. If we use $\mathbf{p}_1, \ldots, \mathbf{p}_d$ as our basis for the velocities, and $\mathbf{p} = \sum_{i=1}^{d} a_i \mathbf{p}_i$, then the rate at which we flip a_i is equal to

$$\max\{0, -a_i \mathbf{p}_i^\top \nabla \log \pi(\boldsymbol{\theta} + \mathbf{p}t)\} = \max\left\{0, -a_i \mathbf{p}_i^\top \left(-\boldsymbol{\Sigma}^{-1}(\boldsymbol{\theta} + \mathbf{p}t)\right)\right\}.$$

But using that $\mathbf{p}_i$ is an eigenvector of $\boldsymbol{\Sigma}$, and hence also of $\boldsymbol{\Sigma}^{-1}$, and if we assume the corresponding eigenvalue of $\boldsymbol{\Sigma}$ is γ_i, we have that

$$-a_i \mathbf{p}_i^\top \left(-\boldsymbol{\Sigma}^{-1}(\boldsymbol{\theta} + \mathbf{p}t)\right) = a_i \frac{1}{\gamma_i} \mathbf{p}_i^\top \left(\boldsymbol{\theta} + t \sum_{j=1}^{d} a_j \mathbf{p}_j\right) t = a_i \frac{1}{\gamma_i} \left(\mathbf{p}_i^\top \boldsymbol{\theta} + a_i t\right).$$

In this case, the event rate does not depend on the velocity in other components and essentially Zig–Zag will reduce to independent processes along each component, $\mathbf{p}_i^\top \boldsymbol{\theta}$, of $\boldsymbol{\theta}$.

A similar idea can be used to generalise the Bouncy Particle Sampler. It is simplest to describe this for the case where the invariant distribution for $\mathbf{p}$ is Gaussian, as the generalisation is to allow a non-identity covariance matrix for this invariant distribution. In the following, we will assume the invariant distribution is Gaussian with mean 0 and variance $\boldsymbol{\Sigma}$.

As we change $\boldsymbol{\Sigma}$, we have to change the reflection events of the sampler. To see why, note that a key property of the reflection event of the standard Bouncy Particle Sampler, wherein step (BPS2)

$$\mathbf{p}' = \mathcal{R}_{\mathbf{g}}(\mathbf{p}), \text{ with } \mathbf{g} = \nabla_\theta \log \pi(\boldsymbol{\theta}),$$

was that $||\mathbf{p}'||_2^2 = ||\mathbf{p}||_2^2$, so this transition does not change the density of the state under $\pi_{\mathbf{p}}$. This is no longer the case if the variance of $\mathbf{p}$ under $\pi_{\mathbf{p}}$ is not a multiple of the identity.

So we need to generalise the reflection so that it depends on $\boldsymbol{\Sigma}$. It turns out that the appropriate reflection is

$$\mathcal{R}_{\mathbf{g}}^{\Sigma}(\mathbf{p}) = \mathbf{p} - \frac{2\mathbf{g}^\top \mathbf{p}}{(\mathbf{g}^\top \boldsymbol{\Sigma} \mathbf{g})} \boldsymbol{\Sigma} \mathbf{g}.$$

Importantly, if $\mathbf{p}' = \mathcal{R}_{\mathbf{g}}^{\Sigma}(\mathbf{p})$ for any $\mathbf{g}$, then

$$\mathbf{p}'^\top \boldsymbol{\Sigma}^{-1} \mathbf{p}' = \mathbf{p}^\top \boldsymbol{\Sigma}^{-1} \mathbf{p},$$

so it does not change the density under a Gaussian with variance $\boldsymbol{\Sigma}$. Furthermore, we still have that if $\mathbf{g} = \nabla \log \pi(\boldsymbol{\theta})$ then

$$\mathbf{p}' \cdot \nabla \log \pi(\boldsymbol{\theta}) = -\mathbf{p} \cdot \nabla \log \pi(\boldsymbol{\theta}),$$

which is the other key requirement of the transition needed for the validity

of the sampler. Using these two properties it is straightforward to show that the Bouncy Particle Sampler with (BPS2) replaced by (BPS2') below will have $\pi(\boldsymbol{\theta})\pi_{\mathbf{p}}(\mathbf{p})$ as its invariant distribution, where $\pi_{\mathbf{p}}(\mathbf{p})$ is the density of a Gaussian distribution with mean 0 and variance $\boldsymbol{\Sigma}$.

(BPS2') *Transition at events.* At an event with probability $1 - \lambda_{\mathrm{r}}/\lambda_{\mathrm{BPS}}(\boldsymbol{\theta}, \mathbf{p})$, reflect the velocity

$$\mathbf{p}' = \mathcal{R}^{\boldsymbol{\Sigma}}_{\mathbf{g}}(\mathbf{p}), \text{ with } \mathbf{g} = \nabla_{\theta} \log \pi(\boldsymbol{\theta});$$

otherwise sample a new velocity, $\mathbf{p}'$ from a normal distribution with mean 0 and variance $\boldsymbol{\Sigma}$. The position is unchanged at an event.

As above, a natural choice of $\boldsymbol{\Sigma}$ to use in the distribution for the velocity is to choose it to be an estimate of the variance of $\boldsymbol{\theta}$ under π. Furthermore, for both the Bouncy Particle Sampler and Zig–Zag one can relate the choice of distribution on the velocity to running the canonical version of the PDMP but after applying a linear reparameterisation to the random variable of the distribution we wish to sample from. We will describe the link for the Bouncy Particle Sampler, though a similar argument applies to other PDMP samplers.

Consider the Bouncy Particle Sampler for $(\boldsymbol{\theta}, \mathbf{p})$ with target distribution $\pi(\boldsymbol{\theta})$ and a standard Gaussian distribution for $\mathbf{p}$. For some invertible matrix $\mathbf{L}$ define $\boldsymbol{\psi} = \mathbf{L}\boldsymbol{\theta}$, and consider the dynamics of the PDMP but viewed in terms of $\boldsymbol{\psi}$. If $\boldsymbol{\theta}$ is drawn from $\pi(\boldsymbol{\theta})$, and $\boldsymbol{\psi} = \mathbf{L}\boldsymbol{\theta}$, then the density of $\boldsymbol{\psi}$ is $\pi_{\psi}(\boldsymbol{\psi}) \propto \pi(\mathbf{L}^{-1}\boldsymbol{\psi})$, as the Jacobian of the transformation is constant. If we consider derivatives then

$$\frac{\partial \log \pi_{\psi}(\boldsymbol{\psi})}{\partial \psi_i} = \frac{\partial \log \pi(\mathbf{L}^{-1}\boldsymbol{\psi})}{\partial \psi_i} = \sum_{j=1}^{d} \frac{\partial \log \pi(\mathbf{L}^{-1}\boldsymbol{\psi})}{\partial \theta_j} \left(\mathbf{L}^{-1}\right)_{ji}.$$

This is just the ith entry of $\mathbf{L}^{-\top}\nabla_{\theta} \log \pi(\mathbf{L}^{-1}\boldsymbol{\psi})$, which gives that

$$\nabla_{\psi} \pi_{\psi}(\boldsymbol{\psi}) = \mathbf{L}^{-\top} \nabla_{\theta} \log \pi(\mathbf{L}^{-1}\boldsymbol{\psi}). \tag{5.18}$$

Now let us consider the dynamics of the Bouncy Particle Sampler in $\boldsymbol{\psi}$ space. We will consider each aspect of the dynamics in turn:
Deterministic Dynamics: If we transform the constant velocity dynamics into $\boldsymbol{\psi}$ space we have

$$\frac{\mathrm{d}\boldsymbol{\psi}}{\mathrm{d}t} = \mathbf{L}\frac{\mathrm{d}\boldsymbol{\theta}}{\mathrm{d}t} = \mathbf{L}\mathbf{p},$$

so these are still constant velocity dynamics but with velocity $\mathbf{w} = \mathbf{L}\mathbf{p}$.

Furthermore, if $\mathbf{p}$ has a Gaussian distribution with an identity covariance matrix, then $\mathbf{w}$ is Gaussian with covariance $\mathbf{L}\mathbf{L}^\top$.

Rate of Bounce Events: If the current state is $(\boldsymbol{\theta}, \mathbf{p})$ then the rate of a bounce event is $\max\{0, \mathbf{p} \cdot \nabla \log \pi(\boldsymbol{\theta})\}$. Now

$$\mathbf{p} \cdot \nabla \log \pi(\boldsymbol{\theta}) = \mathbf{p}^\top \nabla \log \pi(\boldsymbol{\theta}) = \mathbf{w}^\top (\mathbf{L}^{-1})^\top \nabla \log \pi(\mathbf{L}^{-1}\boldsymbol{\psi}) = \mathbf{w}^\top \nabla_{\boldsymbol{\psi}} \pi_{\boldsymbol{\psi}}(\boldsymbol{\psi}),$$

where we have transformed $(\boldsymbol{\theta}, \mathbf{p})$ to $(\boldsymbol{\psi}, \mathbf{w})$ and used (5.18). This is the rate for the Bouncy Particle Sampler targeting $\pi_{\boldsymbol{\psi}}(\boldsymbol{\psi})$.

Reflection at Bounce Events: If the current state is $(\boldsymbol{\theta}, \mathbf{p})$ then at a bounce event the new velocity is

$$\mathbf{p}' = \mathbf{p} - 2(\mathbf{p} \cdot \nabla_\theta \log \pi(\boldsymbol{\theta})) \frac{\nabla_\theta \log \pi(\boldsymbol{\theta})}{(\nabla_\theta \log \pi(\boldsymbol{\theta})^\top \nabla_\theta \log \pi(\boldsymbol{\theta}))^{1/2}}.$$

So if we consider the velocity for the $\boldsymbol{\psi}$ process, $\mathbf{w}' = \mathbf{L}\mathbf{p}'$, and use $\boldsymbol{\Sigma} = \mathbf{L}\mathbf{L}^\top$, this is

$$\begin{aligned}
\mathbf{w}' &= \mathbf{L}\mathbf{p} - 2(\mathbf{p} \cdot \nabla_\theta \log \pi(\boldsymbol{\theta})) \frac{\mathbf{L}\nabla_\theta \log \pi(\boldsymbol{\theta})}{(\nabla_\theta \log \pi(\boldsymbol{\theta})^\top \nabla_\theta \log \pi(\boldsymbol{\theta}))^{1/2}} \\
&= \mathbf{w} - 2(\mathbf{w}^\top \mathbf{L}^{-T} \nabla_\theta \log \pi(\mathbf{L}^{-1}\boldsymbol{\psi})) \\
&\quad \times \frac{\boldsymbol{\Sigma}\mathbf{L}^{-T}\nabla_\theta \log \pi(\mathbf{L}^{-1}\boldsymbol{\psi})}{(\nabla_\theta \log \pi(\mathbf{L}^{-1}\boldsymbol{\psi})^\top \mathbf{L}^{-1}\boldsymbol{\Sigma}\mathbf{L}^{-T}\nabla_\theta \log \pi(\mathbf{L}^{-1}\boldsymbol{\psi}))^{1/2}} \\
&= \mathbf{w} - 2(\mathbf{w}^\top \nabla_{\boldsymbol{\psi}} \log \pi_{\boldsymbol{\psi}}(\boldsymbol{\psi})) \frac{\boldsymbol{\Sigma}\nabla_{\boldsymbol{\psi}} \log \pi_{\boldsymbol{\psi}}(\boldsymbol{\psi})}{(\nabla_{\boldsymbol{\psi}} \log \pi_{\boldsymbol{\psi}}(\boldsymbol{\psi})^\top \boldsymbol{\Sigma}\nabla_{\boldsymbol{\psi}} \log \pi_{\boldsymbol{\psi}}(\boldsymbol{\psi}))^{1/2}}.
\end{aligned}$$

This is just $\mathcal{R}^{\boldsymbol{\Sigma}}_{\mathbf{g}}(\mathbf{w})$ with $\mathbf{g} = \nabla \log_{\boldsymbol{\psi}}(\boldsymbol{\psi})$, the reflection of the Bouncy Particle Sampler with covariance matrix $\boldsymbol{\Sigma}$.

Refresh Events: These events occur at a constant rate, which is unaffected by the transformation to $\boldsymbol{\psi}$. At a refresh event, we simulate $\mathbf{p}$ from a standard Gaussian, which corresponds to simulating $\mathbf{w} = \mathbf{L}\mathbf{p}$ from a Gaussian with covariance $\boldsymbol{\Sigma} = \mathbf{L}\mathbf{L}^\top$.

Thus the process in $\boldsymbol{\psi}$ space is a Bouncy Particle Sampler with covariance matrix $\boldsymbol{\Sigma} = \mathbf{L}\mathbf{L}^\top$ for the velocity.

A second generalisation is to alter the constant velocity dynamics. This has been suggested in particular as a way of generalising the Bouncy Particle Sampler with covariance matrix $\boldsymbol{\Sigma}$ for the velocity, with the resulting algorithm called the Boomerang Sampler (Bierkens et al., 2020), though similar ideas also appear under the name of Hamiltonian-BPS in Vanetti et al. (2017).

Consider a velocity model with marginal distribution such that $\log \pi_{\mathbf{p}}(\mathbf{p}) = -(1/2)\mathbf{p}^\top \boldsymbol{\Sigma}^{-1}\mathbf{p}$. Write $\log \pi(\boldsymbol{\theta}) = U(\boldsymbol{\theta}) - (1/2)(\boldsymbol{\theta} - \boldsymbol{\theta}^*)^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{\theta} - \boldsymbol{\theta}^*)$ for

some function $U(\boldsymbol{\theta})$ and constant vector $\boldsymbol{\theta}^*$. The idea is to have deterministic dynamics that move along contours of $-(1/2)(\boldsymbol{\theta}-\boldsymbol{\theta}^*)^\top\boldsymbol{\Sigma}^{-1}(\boldsymbol{\theta}-\boldsymbol{\theta}^*) - (1/2)\mathbf{p}^\top\boldsymbol{\Sigma}^{-1}\mathbf{p}$ in $(\boldsymbol{\theta}, \mathbf{p})$ space. Such dynamics are given by Hamiltonian dynamics, which are tractable in this case, and are

$$\frac{\mathrm{d}\boldsymbol{\theta}}{\mathrm{d}t} = \mathbf{p}, \quad \frac{\mathrm{d}\mathbf{p}}{\mathrm{d}t} = \boldsymbol{\theta} - \boldsymbol{\theta}^*.$$

The solution of these dynamics are $\boldsymbol{\theta}_t = \boldsymbol{\theta}^* + (\boldsymbol{\theta}_0 - \boldsymbol{\theta}^*)\cos(t) + \mathbf{p}_0\sin(t)$ and $\mathbf{p}_t = \mathbf{p}_0\cos(t) - (\boldsymbol{\theta}_0 - \boldsymbol{\theta}^*)\sin(t)$. The rate of bounce events for the Boomerang Sampler is just $\max\{0, \mathbf{p}_t \cdot \nabla U(\boldsymbol{\theta}_t)\}$, with bounces as per the Bouncy Particle Sampler when the velocity has covariance matrix $\boldsymbol{\Sigma}$. As before, we can also introduce refresh events.

If $U(\boldsymbol{\theta}) = 0$, so we are targeting a Gaussian distribution for $\boldsymbol{\theta}$ with mean $\boldsymbol{\theta}^*$ and covariance $\boldsymbol{\Sigma}$, then this sampler just undergoes Hamiltonian dynamics. In this case, a strictly positive refresh rate is needed to avoid the sampler being reducible, and the resulting process is a form of randomised Hamiltonian dynamics, that is HMC but with a random refresh time for the velocity. For non-Gaussian targets, this sampler will have additional bounce events, but the hope is that if the target is close to Gaussian with mean $\boldsymbol{\theta}^*$ and covariance $\boldsymbol{\Sigma}$, then the rate of bounce events will be much lower than for the standard Bouncy Particle Sampler.

Care is needed with one aspect of simulating the Boomerang Sampler, as the different dynamics require slightly different approaches to simulate the event times. If the current state is $(\boldsymbol{\theta}_0, \mathbf{p}_0)$ then the rate until the next event is now

$$\begin{aligned}\tilde{\lambda}_{(\boldsymbol{\theta}_0,\mathbf{p}_0)}(t) &= \max\{0, \mathbf{p}_t \cdot \nabla U(\boldsymbol{\theta}_t)\} = \max\{0,\\ &(\mathbf{p}_0\cos(t) + (\boldsymbol{\theta}_0 - \boldsymbol{\theta}^*)\sin(t)) \cdot \nabla U(\boldsymbol{\theta}^* + (\boldsymbol{\theta}_0 - \boldsymbol{\theta}^*)\cos(t) + \mathbf{p}_0\sin(t))\},\end{aligned}$$

where we have substituted in the definitions of $\mathbf{p}_t$ and $\boldsymbol{\theta}_t$. Bierkens et al. (2020) give some general approaches to bounding this rate, which uses the property that the deterministic dynamics of the Boomerang Sampler are such that $|\boldsymbol{\theta}_t - \boldsymbol{\theta}^*|^2 + |\mathbf{p}_t|^2$ is a constant. To keep the notation simple, we will show this for $\boldsymbol{\theta}^* = \mathbf{0}$, but the same argument applies more generally. In this case

$$\begin{aligned}|\boldsymbol{\theta}_t|^2 + |\mathbf{p}_t|^2 &= (\boldsymbol{\theta}_0\cos(t) + \mathbf{p}_0\sin(t)) \cdot (\boldsymbol{\theta}_0\cos(t) + \mathbf{p}_0\sin(t))\\ &\quad +(\mathbf{p}_0\cos(t) - \boldsymbol{\theta}_0\sin(t)) \cdot (\mathbf{p}_0\cos(t) - \boldsymbol{\theta}_0\sin(t))\\ &= |\boldsymbol{\theta}_0|^2\cos^2(t) + |\mathbf{p}_0|^2\sin^2(t) + 2\boldsymbol{\theta}_0 \cdot \mathbf{p}_0\sin(t)\cos(t)\\ &\quad +|\mathbf{p}_0|\cos^2(t) + |\boldsymbol{\theta}_0|^2\sin^(t) - 2\boldsymbol{\theta}_0 \cdot \mathbf{p}_0\sin(t)\cos(t)\\ &= |\boldsymbol{\theta}_0|^2(\sin^2(t) + \cos^2(t) + |\mathbf{p}_0|^2(\sin^2(t) + \cos^2(t)) = |\boldsymbol{\theta}_0|^2 + |\mathbf{p}_0|^2.\end{aligned}$$

How is this useful? This property means that we can bound the distance from $\boldsymbol{\theta}^*$ that the current deterministic trajectory can take. Thus if we can bound the Hessian of U, which is the derivative of ∇U, then this enables us to bound ∇U for the current trajectory based on the value of ∇U at $\boldsymbol{\theta}^*$ plus a term that depends on the bound on the Hessian and the distance the trajectory can be from $\boldsymbol{\theta}^*$. One such bound, that we will use below is that if the spectral norm of the Hessian of U is bounded by M, so that $\|\nabla^2 U(\boldsymbol{\theta})\mathbf{x}\| \leq M\|\mathbf{x}\|_2$ for any vector $\mathbf{x}$ with $\|\cdot\|$ denoting Euclidean distance, then for the current deterministic trajectory, we have a constant bound:

$$\lambda(\boldsymbol{\theta}_t, \mathbf{p}_t) \leq |\nabla U(\boldsymbol{\theta}^*)|(|\boldsymbol{\theta}_0 - \boldsymbol{\theta}^*|^2 + |\mathbf{p}_0|^2)^{1/2} + \frac{1}{2}M(|\boldsymbol{\theta}_0 - \boldsymbol{\theta}^*|^2 + |\mathbf{p}_0|^2). \quad (5.19)$$

Example: Boomerang for Logistic Regression

As an example, consider again the logistic regression model. There are two natural choices for the centring value and covariance of the Boomerang Sampler. The first is to set them to the prior mean and covariance, $\boldsymbol{\theta}^* = \mathbf{0}$ and $\boldsymbol{\Sigma} = \boldsymbol{\Sigma}_{\boldsymbol{\theta}}$. The second is to estimate the mode of $\log \pi$, $\widehat{\boldsymbol{\theta}}_{MAP}$ say, and the inverse of the Hessian of $-\log \pi$ at $\widehat{\boldsymbol{\theta}}_{MAP}$. We will compare these two options.

If we set them to the prior values then $U(\boldsymbol{\theta})$ is minus the log-likelihood. As described in Section 5.4.1 we can bound the Hessian of minus the log-likelihood by $(1/4)\mathbf{X}^\top\mathbf{X}$, where $\mathbf{X}$ is the $N \times d$ matrix of covariates.

What about if we set $\boldsymbol{\theta}^*$ and $\boldsymbol{\Sigma}$ based on the estimate of the mode of $\log \pi$ and the inverse of the Hessian of $-\log \pi$ at the mode? Denoting the log-likelihood of the logistic model by $\ell(\boldsymbol{\theta}; \mathcal{D})$, and the Hessian of minus the log-likelihood by $\mathbf{H}(\boldsymbol{\theta})$. This choice gives

$$U(\boldsymbol{\theta}) = -\ell(\boldsymbol{\theta}; \mathcal{D}) - \frac{1}{2}(\boldsymbol{\theta} - \widehat{\boldsymbol{\theta}}_{MAP})^\top \mathbf{H}(\widehat{\boldsymbol{\theta}}_{MAP})(\boldsymbol{\theta} - \widehat{\boldsymbol{\theta}}_{MAP}),$$

where we have used the fact that the contribution from the prior will cancel. Taking second derivatives, the Hessian of this at $\boldsymbol{\theta}$ will be the difference between two matrices, $\mathbf{H}(\boldsymbol{\theta}) - \mathbf{H}(\widehat{\boldsymbol{\theta}}_{MAP})$. These matrices are both positive semi-definite, with spectral norm bounded by $(1/4)\mathbf{X}^\top\mathbf{X}$, thus the spectral norm of the difference is also bounced by $(1/4)\mathbf{X}^\top\mathbf{X}$. This is because the eigenvalues of $\mathbf{H}(\boldsymbol{\theta})$ are bounded between 0 and M for some constant M, and the eigenvalues of $-\mathbf{H}(\widehat{\boldsymbol{\theta}}_{MAP})$ are bounded between $-M$ and 0, so the eigenvalues of $\mathbf{H}(\boldsymbol{\theta}) - \mathbf{H}(\widehat{\boldsymbol{\theta}}_{MAP})$ are between $-M$ and M.

Thus we can implement the Boomerang Sampler for both choices of $\boldsymbol{\theta}^*$ and $\boldsymbol{\Sigma}$ with the constant bound given by (5.19) using the same value for

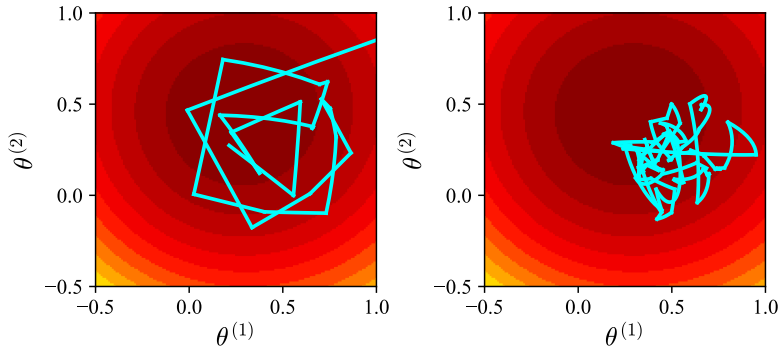

Figure 5.8 Trace plots of the Boomerang Sampler for the posterior of a logistic regression model. The heat plot shows the contours of the posterior. Example realisation when $\boldsymbol{\Sigma} = \boldsymbol{\Sigma}_{\boldsymbol{\theta}}$ and $\boldsymbol{\theta}^*$ is the prior mean (left) and when $\boldsymbol{\theta}^*$ is an estimate of the posterior mode and $\boldsymbol{\Sigma}$ is based on the Hessian of $\log \pi$ at $\boldsymbol{\theta}^*$ (right).

M. The bounds will differ though due to the different values for $\boldsymbol{\theta}^*$ and U and hence for $|\nabla U(\boldsymbol{\theta}^*)|$. In particular, this will be 0 for $\boldsymbol{\theta}^* = \widehat{\boldsymbol{\theta}}_{MAP}$, or at least close to 0 if we have a reasonable approximation for the mode of $\log \pi$. Example realisations for the two samplers are shown in Figure 5.8, for data with $N = 100$ and $d = 2$ and with a prior covariance of 2 for each component of $\boldsymbol{\theta}$.

The main difference between the Boomerang Sampler and the previous PDMP samplers is most obviously seen in the right-hand plot of Figure 5.8, as we have elliptical trajectories between events. This is reminiscent of the trajectories for HMC. For the left-hand plot, where Σ is large compared to the curvature of the posterior, the contours are elliptical but with larger radii and thus the output looks more similar to our previous PDMP samplers. In this example, one of the main advantages of basing $\boldsymbol{\theta}^*$ and Σ on the mode and Hessian at the mode is that the computational cost of simulating the Boomerang Sampler is lower. Both have been simulated with roughly the same length of trajectory, but using the prior values has required proposing four times as many events. This is because of the looser bound on the event rate that we have in this case.

5.6 Chapter Notes

The initial idea of using PDMP processes for sampling comes from the physical sciences, see for example Turitsyn et al. (2011), Peters and de With (2012) and Michel et al. (2014). These were introduced into statistics by Bouchard-Côté et al. (2018) and Bierkens and Roberts (2017), and one of the early papers to describe the link to PDMPs was Fearnhead et al. (2018). The latter paper also gives an example where avoiding refresh events in the Bouncy Particle Sampler can lead to slow mixing. How the subsampling ideas of Section 5.4.3 extend to samplers other than the Zig–Zag Sampler is also discussed in Fearnhead et al. (2018), with related ideas for the local Bouncy Particle Sampler in Bouchard-Côté et al. (2018).

As well as the PDMP samplers mentioned in the chapter, there have been various papers suggesting extensions to PDMP methods. For example Vanetti et al. (2017), Wu and Robert (2017) and Michel et al. (2020). The continuous-time methods can be related to discrete-time MCMC methods such as reflective slice sampling (Neal, 2003) and the Discrete Bouncy Particle Sampler (Sherlock and Thiery, 2022).

The theoretical analysis of PDMP samplers has been active, including showing ergodicity (Deligiannidis et al., 2019; Bierkens et al., 2019a) and exploring limiting behaviour of the Bouncy Particle Sampler and the Zig Zag sampler (Deligiannidis et al., 2021; Bierkens et al., 2022; Andrieu et al., 2021). Particularly strong results are available for the one-dimensional case (Bierkens and Duncan, 2017; Bierkens and Verduyn Lunel, 2022).

Complementary results on scaling of the Discrete Bouncy Particle Sampler to those shown in Section 5.3.3 are given in Sherlock and Thiery (2022), which shows similar scaling to the Bouncy Particle Sampler and also provides supporting theory to help choose the refresh rate.

6

Assessing and Improving MCMC

The development of more sophisticated and, especially, approximate sampling algorithms, aimed at improving scalability in one or more of the senses already discussed in this book, raises important considerations about how a suitable algorithm should be selected for a given task, how its tuning parameters should be determined, and how its convergence should be assessed. This chapter presents recent solutions to the above problems, whose starting point is to derive explicit upper bounds on an appropriate distance between the posterior and the approximation produced by MCMC. Further, we explain how these same tools can be adapted to provide powerful post-processing methods that can be used retrospectively to improve approximations produced using scalable MCMC.

6.1 Diagnostics for MCMC

The approximations provided by MCMC are only useful if we can be confident that the samples collectively form a reasonable approximation to the intended target. This, however, appears to be a circular requirement – how can we verify that MCMC has worked without access to the limiting target to check? Several *diagnostics* have emerged as pragmatic solutions, enabling a practitioner to detect certain failure modes of MCMC. In particular, we highlight *convergence diagnostics*, which aim to determine whether the Markov chain has converged to *some* stationary distribution, and *bias diagnostics*, which aim to detect whether the stationary distribution of the Markov chain is indeed the target distribution of interest. For context, both classes of diagnostic will be briefly discussed. Throughout this Chapter, we shall restrict attention to distributions defined on $\mathbb{R}^d$ for simplicity of presentation.

6.1.1 Convergence Diagnostics

To limit the scope, here we describe the convergence diagnostics that are most widely used. The *Gelman–Rubin* diagnostic is based on realisations of L independent Markov chains, each of length n, where practical considerations typically restrict L to be a small number, such as 3, 4 or 5. For a univariate target distribution, the Gelman–Rubin diagnostic is defined as the square root of the ratio of two estimators of the variance σ^2 of the target distribution

$$\widehat{R} := \sqrt{\frac{\widehat{\sigma}^2}{\widehat{s}^2}}, \tag{6.1}$$

where $\widehat{s}^2$ is the (arithmetic) mean of the sample variances s_l^2 along the L sample paths,

$$\widehat{s}^2 := \frac{1}{L}\sum_{l=1}^{L} s_l^2,$$

which typically provides an underestimate of σ^2, since it is possible that one or more of the L chains has not explored the posterior well, while $\widehat{\sigma}^2$ is constructed as

$$\widehat{\sigma}^2 := \frac{n-1}{n}\widehat{s}^2 + \frac{1}{L-1}\sum_{l=1}^{L}\left(m_l - \frac{1}{L}\sum_{l'=1}^{L} m_{l'}\right)^2, \tag{6.2}$$

where m_l is the sample mean from the lth sample path, which typically provides an overestimate of the target variance. Indeed, the second term in (6.2) is an estimate of the asymptotic variance of the sample mean of the Markov chain, which is typically larger than the variance $\frac{\sigma^2}{n}$ we would obtain if our samples were truly independent; recall the discussion of effective samples sizes and (1.15) from Chapter 1. For an ergodic Markov chain, $\widehat{R}$ converges to 1 as $n \to \infty$. In practice, it is common to discard a burn-in period of length $\frac{n}{2}$, where n is the smallest integer for which $\widehat{R} < 1 + \delta$, where δ is a suitable threshold. The somewhat arbitrary choices of $\delta = 0.1$ and $\delta = 0.01$ are often used.

Convergence diagnostics are widely and successfully used. However, it remains the case that the performance of $\widehat{R}$ and related convergence diagnostics depends strongly on how the independent realisations of MCMC are initialised. Indeed, consider the task of generating approximate samples from the mixture distribution

$$\pi(x) = \frac{1}{2}\mathsf{N}(x; -2, 0.5^2) + \frac{1}{2}\mathsf{N}(x; 2, 0.5^2). \tag{6.3}$$

To avoid the situation where all chains are confined to the same local high-probability region due to chance, standard practice is to initialise the Markov chains by sampling their initial state from a distribution that is over-dispersed with respect to the target. Thus, we may initialise Markov chains by sampling from, say, $\mathsf{N}(0, 5^2)$. Running $L = 3$ chains of length $n = 1000$ leads to the two sets of sample paths shown in Figure 6.1. In both sets of sample paths, the length n of the sample paths was insufficient to enable the Markov chains to explore both components of π, and each of the chains remained in the component in which it was initialised. On the left side of the figure, one of the sample paths is clearly qualitatively distinct from the other two, since the Markov chains explored different components of π, and the Gelman–Rubin diagnostic correctly detects that the Markov chains have not converged. Unfortunately, on the right side of the figure, it so happened that each of the chains was initialised in the high probability region of the same component. As a result, the Gelman–Rubin diagnostic appears to be converging to values below the commonly used thresholds $\delta = 0.1$ and $\delta = 0.01$, and fails to detect that the Markov chains have explored only one of the components of π.

What went wrong with the convergence diagnostic (6.1) in this example? Well, the Markov processes exhibited a form of *quasi-stationarity*; transitions from one component of the mixture to the other is a rare event, and conditional on such a transition not occurring the behaviour of the Markov chains is arguably excellent. The rarity of transitions between components makes it fundamentally difficult to distinguish between quasi-stationarity and convergence of a Markov chain when the sample paths are confined to the same component; some knowledge of the invariant distribution π is required. This motivates the discussion of an alternative diagnostic which does indeed leverage information about π, a *bias diagnostic*, which we describe next.

6.1.2 Bias Diagnostics

Even in favourable situations, such as the case of a uni-modal target, convergence diagnostics do not provide a guarantee that Markov chain samples constitute a faithful approximation of the target. Indeed, convergence diagnostics are not capable of detecting *bias* in sampler output, for example as introduced when using stochastic gradient methods (Chapter 3), or introduced when a coding error has occurred. Instead, *bias diagnostics* can be

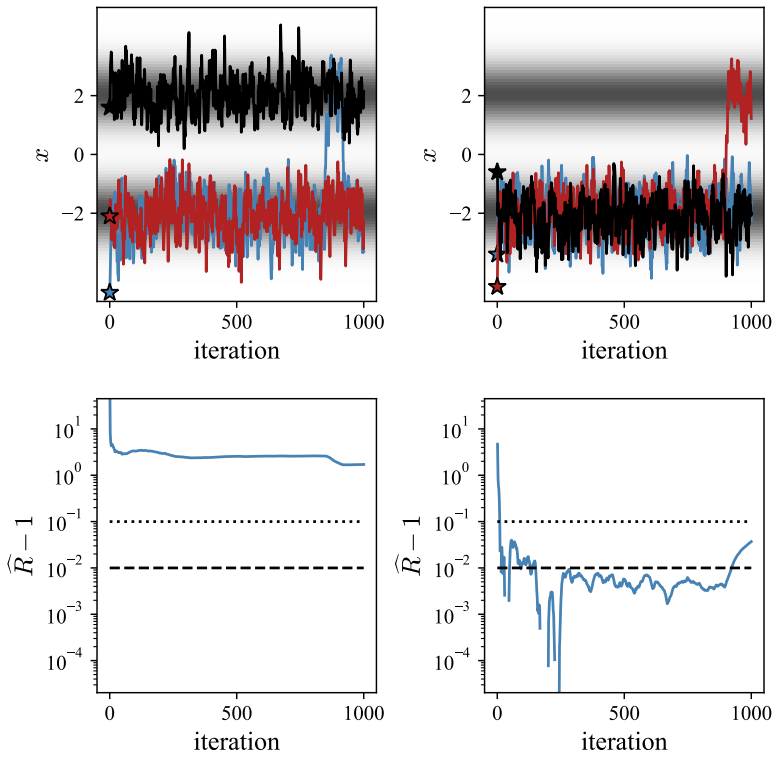

Figure 6.1 Convergence diagnostics for MCMC. Three independent Markov chains were simulated to generate samples from the Gaussian mixture target π in (6.3). In the first scenario (left panels) the chains explore different components of π, and the Gelman–Rubin diagnostic $\widehat{R}$ correctly detects that the Markov chains have not converged. In the second scenario (right panels) the chains explore the same component of π, and the Gelman–Rubin diagnostic does not detect that the Markov chains have not converged. [Stars indicate the initial state of each Markov chain. The density π is shaded. Dotted lines indicate the two commonly used thresholds $\delta = 0.1$ and 0.01 for $\widehat{R} - 1$.]

used to identify such situations, a simple example of which is

$$\widehat{B} := \left\| \frac{1}{n} \sum_{k=1}^{n} (\nabla \log \pi)(X_k) \right\|. \tag{6.4}$$

Provided that $\nabla \log \pi \in \mathcal{L}^1(\pi)$, which we recall means that $\int \|\nabla \log \pi\| \, \mathrm{d}\pi < \infty$ from Section 1.1.1, from the strong law of large numbers for Markov chains the series in (6.4) almost surely converges to the limit

$$\int \nabla \log \pi \, \mathrm{d}\pi = \int \frac{(\nabla \pi)(x)}{\pi(x)} \pi(x) \, \mathrm{d}x = \int (\nabla \pi)(x) \, \mathrm{d}x = 0$$

whenever the Markov chain is ergodic and π-invariant. The final equality is integration by parts; a special case of Lemma 6.3 in the sequel. On the other hand, just as the passing of a convergence diagnostic test does not guarantee that the MCMC has converged, the convergence of (6.4) to 0 does not guarantee that the Markov chain preserves the correct target distribution. Surprisingly, bias diagnostics are not widely used, at least compared to convergence diagnostics, which may be due to (in our example) the requirement to compute a gradient, or may simply be because they have been historically overlooked.

Consider again the mixture distribution π in (6.3) and suppose that, due to a coding error, we have implemented a Markov chain whose stationary distribution is $\mathsf{N}(\mu, 0.5^2)$. Running $L = 3$ chains of length $n = 1000$ leads to the two sets of sample paths shown in Figure 6.2 for $\mu = 2$ (left) and $\mu = 0$ (right). In both sets of sample paths, the Gelman–Rubin convergence diagnostic test is passed, despite the Markov chains failing to be π-invariant. On the left side of the figure, the bias diagnostic (6.4) clearly does not converge to zero, and thus the bias in the Markov chain output is detected. Unfortunately, on the right side of the figure, the bias diagnostic appears to be decreasing for all chains as the number n of samples is increased, and we do not diagnose the failure of MCMC.

What went wrong with the bias diagnostic (6.4) in this example? Well, information about a *finite-dimensional* generalised moment $\int \nabla \log \pi \, \mathrm{d}\nu \in \mathbb{R}^d$ is insufficient to characterise a probability distribution; there are an infinitude of distributions ν for which all d components of this generalised moment are 0. This suggests a potential solution; find an *infinite-dimensional* generalised moment that fully determines whether or not π and ν are equal. Surprisingly, this can be achieved without needing to explicitly deal with an infinite-dimensional generalised moment, due to the *kernel trick* from machine learning, which was introduced in Section 1.5.3 for finite-dimensional inner-product spaces, and will now be explored in the infinite-dimensional setting.

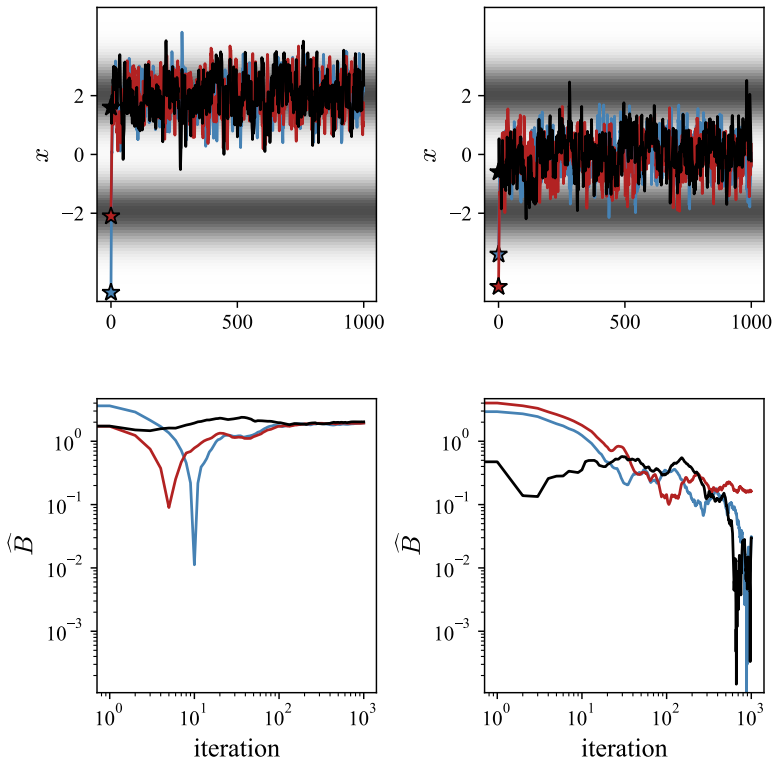

Figure 6.2 Bias diagnostics for MCMC. Three independent biased Markov chains were simulated, so that the invariant distribution differs from the Gaussian mixture target π in (6.3). In the first scenario (left panels) the chains explore a Gaussian centred at $x = 2$, and the bias diagnostic $\widehat{B}$ correctly detects that the Markov chains have not converged. In the second scenario (right panels) the chains explore a Gaussian target centred at $x = 0$, and the bias diagnostic does not detect that the Markov chains have not converged. [Stars indicate the initial state of each Markov chain. The density π is shaded.]

6.1.3 Improved Bias Diagnostics via the Kernel Trick

Though finite-dimensional bias diagnostics can be misled, the same may not be true of a bias diagnostic that is infinite-dimensional. The aim of this section is to indicate, at a high level, how such an infinite-dimensional bias

diagnostic can be constructed. A more rigorous mathematical treatment is then provided in Section 6.2.

Suppose that we are able to write down a countable set $\{\phi_1, \phi_2, \dots\}$ of functions $\phi_j : \mathbb{R}^d \to \mathbb{R}$ such that each of the moments $\int \phi_j(\mathbf{x})\pi(\mathbf{x})\ \mathrm{d}\mathbf{x}$ of π can be analytically evaluated; without loss of generality we may suppose that each generalised moment of π is equal to 0 (since if not, we may simply redefine ϕ_j as $\phi_j - \int \phi_j(\mathbf{x})\pi(\mathbf{x})\ \mathrm{d}\mathbf{x}$). We have already seen examples of functions ϕ_j that can be used; we could take

$$\phi_j(\mathbf{x}) = \frac{\partial \log \pi(\mathbf{x})}{\partial x_j}$$

for $j = 1, \dots, d$. More generally, we can use the generator of any π-invariant Markov process to construct such functions; the details are deferred to Section 6.2. Assuming that the ϕ_j are linearly independent and appropriately normalised, we can construct a Hilbert space

$$\mathcal{H} = \left\{ h = \sum_{j=1}^{\infty} c_j \phi_j \ : \ \sum_{j=1}^{\infty} c_j^2 < \infty \right\}$$

whose elements are functions $h : \mathbb{R}^d \to \mathbb{R}$, equipped with an inner product $\langle h, h' \rangle_{\mathcal{H}} = \sum_{j=1}^{\infty} c_j c'_j$, where here $h = \sum_{j=1}^{\infty} c_j \phi_j$ and $h' = \sum_{j=1}^{\infty} c'_j \phi_j$. The induced norm is $\|h\|_{\mathcal{H}} := \langle h, h \rangle_{\mathcal{H}}^{1/2}$. By picking elements from this Hilbert space we can construct an infinitude of bias diagnostics, and the question is then which diagnostic to pick?

One solution is to adopt an *adversarial* perspective, where we select an element $h \in \mathcal{H}$ that maximally discriminates between π and the empirical approximation to π produced from MCMC. The bias diagnostic obtained in this way can be written as

$$\widetilde{B} := \sup_{\|h\|_{\mathcal{H}} \leq 1} \left| \frac{1}{n} \sum_{k=1}^{n} h(\mathbf{X}_k) \right|,$$

where the supremum is taken over the unit ball of $\mathcal{H}$, to ensure that the supremum is computed over a bounded set. Further re-writing in terms of the basis functions, we have

$$\widetilde{B} := \sup \left\{ \left| \frac{1}{n} \sum_{k=1}^{n} \sum_{j=1}^{\infty} c_j \phi_j(\mathbf{X}_k) \right| \ : \ \sum_{j=1}^{\infty} c_j^2 \leq 1 \right\}, \tag{6.5}$$

whence we see the maximisation is equivalent to finding the point $\mathbf{c}$ on the surface of the (infinite-dimensional) unit hypersphere that maximises the dot product with the (infinite-dimensional) vector $\mathbf{c}'$ with $c'_j = \frac{1}{n} \sum_{k=1}^{n} \phi_j(\mathbf{X}_k)$.

The solution of this maximisation problem is to align $\mathbf{c}$ to $\mathbf{c}'$, and upon properly normalising we obtain

$$c_j = \frac{\frac{1}{n}\sum_{k=1}^{n} \phi_j(\mathbf{X}_k)}{\sqrt{\sum_{j'=1}^{\infty} \left(\frac{1}{n}\sum_{k'=1}^{n} \phi_{j'}(\mathbf{X}_{k'})\right)^2}}.$$

Inserting this expression back into (6.5) and rearranging, we obtain the explicit bias diagnostic

$$\widetilde{B} = \sqrt{\frac{1}{n^2}\sum_{k=1}^{n}\sum_{k'=1}^{n}\left(\sum_{j=1}^{\infty} \phi_j(\mathbf{X}_k)\phi_j(\mathbf{X}_{k'})\right)}.$$

At this point one can raise a reasonable objection that evaluating $\widetilde{B}$ appears to require an infinite computational budget, due to the summation over the functions ϕ_j. Remarkably, there are situations where such infinite series admit closed-form analytic expressions

$$\mathsf{k}_\pi(\mathbf{x}, \mathbf{x}') := \sum_{j=1}^{\infty} \phi_j(\mathbf{x})\phi_j(\mathbf{x}'),$$

a situation known in machine learning as the *kernel trick*. See Section 1.5 for a primer on the kernel trick. Provided that we have access to a kernel trick, which of course depends on the precise choice we make for the functions ϕ_j to ensure that $\int \phi_j(\mathbf{x})\pi(\mathbf{x})\,\mathrm{d}\mathbf{x} = 0$, we can hope to obtain a closed-form expression for the bias diagnostic (6.5), namely

$$\widetilde{B} = \sqrt{\frac{1}{n^2}\sum_{k=1}^{n}\sum_{k'=1}^{n} \mathsf{k}_\pi(\mathbf{X}_k, \mathbf{X}_{k'})}. \tag{6.6}$$

One would hope that (6.6) converges to 0 as $n \to \infty$ if and only if the Markov chain is π-invariant. It turns out that such an idea can be made to work, as we will explain in Section 6.2.

To summarise, we have seen that both convergence diagnostics and conventional finite-dimensional bias diagnostics can provide a useful practical tool to detect the failure of MCMC, but even taken together they are insufficient to guarantee that output from the sampler provides an accurate approximation of the intended target distribution. In the next section, we turn our attention to the construction of infinite-dimensional bias diagnostics, of the form (6.6), which attempt to solve the problem of assessing MCMC output by establishing explicit upper bounds on an appropriate distance between the posterior and the approximation produced by MCMC in terms of diagnostics of the form (6.6).

6.2 Convergence Bounds for MCMC

In contrast to convergence diagnostics and conventional finite-dimensional bias diagnostics, which may fail to detect instances where MCMC has failed, the aim of this section is to seek explicit and computable *upper bounds* on the error between the MCMC output and the target distribution. This topic has received considerable recent interest following the pioneering work of Gorham and Mackey (2015). To set the scene, we first explain how the use of a suitable diffusion process enables explicit bounds on integral probability metrics, focusing on the Wasserstein-1 metric for clarity in Sections 6.2.1 and 6.2.2. However, the Wasserstein-1 metric is not favourable for computation in this context, and we instead consider integral probability metrics based on reproducing kernel Hilbert spaces in Section 6.2.3, noting that the associated *kernel Stein discrepancies* can also provide valid convergence bounds in Section 6.2.4. Lastly, in Section 6.2.5 we connect kernel Stein discrepancies to the stochastic gradient methods from Chapter 3.

6.2.1 Bounds on Integral Probability Metrics

Our aim here is to arrive at an explicit and computable upper bound on an appropriate metric between the target distribution π and the empirical distribution produced by MCMC. Let $\mathcal{P}(\mathbb{R}^d)$ denote the set of probability distributions on $\mathbb{R}^d$ and consider a metric $\mathsf{d} : \mathcal{P}(\mathbb{R}^d) \times \mathcal{P}(\mathbb{R}^d) \to [0, \infty]$. As a useful convention, we have extended the range of a metric to include the value ∞, to avoid the need to specify the subset of distributions on which the metric is defined. Let $\pi \in \mathcal{P}(\mathbb{R}^d)$ be the distributional target of MCMC. For our theoretical development, we will now introduce an auxiliary discrete time ergodic Markov chain, which need not be related to the Markov process(es) underpinning the MCMC method(s) being assessed. The role of this auxiliary chain is limited to being a theoretical device in what follows, and we denote its transition kernel as T_π, meaning that $T_\pi \nu$ is the distribution after one step of the auxiliary Markov chain initialised from $\mathbf{X}_0 \sim \nu$. This auxiliary chain is required to satisfy the *contraction* property

$$\mathsf{d}(T_\pi \pi, T_\pi \nu) \leq \rho \, \mathsf{d}(\pi, \nu) \tag{6.7}$$

for some $\rho \in [0, 1)$ and all $\nu \in \mathcal{P}(\mathbb{R}^d)$. From the triangle inequality, $\mathsf{d}(\pi, \nu) \leq \mathsf{d}(\pi, T_\pi \nu) + \mathsf{d}(T_\pi \nu, \nu)$, combined with the contraction $\mathsf{d}(\pi, T_\pi \nu) = \mathsf{d}(T_\pi \pi, T_\pi \nu) \leq \rho \, \mathsf{d}(\pi, \nu)$, it follows that $(1 - \rho)\mathsf{d}(\pi, \nu) \leq \mathsf{d}(T_\pi \nu, \nu)$. The

discrepancy

$$D_\pi(\nu) := \frac{1}{1-\rho}\,\mathsf{d}(T_\pi\nu, \nu)$$

therefore constitutes an upper bound on the metric $\mathsf{d}(\pi, \nu)$, which could in principle be used to quantify how well a given distribution ν approximates π in situations where we do not have direct access to π, but where the auxiliary Markov chain can be simulated. Further, since d is a metric and the Markov chain has a unique invariant distribution, $D_\pi(\nu) = 0$ if and only if $\nu = \pi$. An ideal scenario would be a fast mixing auxiliary Markov chain, so that $\rho \ll 1$, $T_\pi\nu \approx \pi$, and $D_\pi(\nu) \approx \mathsf{d}(\pi, \nu)$. On the other hand, if the auxiliary Markov chain mixes slowly then the values taken by the discrepancy could fail to provide a meaningful indication of whether or not ν is an accurate approximation to π. The utility of this upper bound therefore depends on the mixing properties of the auxiliary Markov chain on which it is based.

To move towards a computable bound, let us suppose that d is an *integral probability metric*, meaning that for any $\pi, \nu \in \mathcal{P}(\mathbb{R}^d)$

$$\mathsf{d}(\pi, \nu) = \sup_{g \in \mathcal{G}} \int g(\mathbf{x})\, \pi(\mathrm{d}\mathbf{x}) - \int g(\mathbf{x})\, \mathrm{d}\nu(\mathbf{x}) \tag{6.8}$$

for a suitable symmetric set[1] of test functions $\mathcal{G}$. Introducing the linear operator

$$(L_\pi g)(\cdot) = \int g(\mathbf{x}')T_\pi(\cdot, \mathrm{d}\mathbf{x}') - g(\cdot),$$

and observing that

$$\begin{aligned}\int (L_\pi g)(\mathbf{x})\, \mathrm{d}\nu(\mathbf{x}) &= \int g(\mathbf{x}')T_\pi(\mathbf{x}, \mathrm{d}\mathbf{x}')\, \mathrm{d}\nu(\mathbf{x}) - \int g(\mathbf{x})\, \mathrm{d}\nu(\mathbf{x}) \\ &= \int g(\mathbf{x})\, \mathrm{d}T_\pi\nu(\mathbf{x}) - \int g(\mathbf{x})\, \mathrm{d}\nu(\mathbf{x}),\end{aligned}$$

the discrepancy can be expressed as

$$D_\pi(\nu) = \frac{1}{1-\rho} \sup_{g \in \mathcal{G}} \int (L_\pi g)(\mathbf{x})\, \mathrm{d}\nu(\mathbf{x}). \tag{6.9}$$

However, to actually evaluate this discrepancy we are required to compute expressions involving L_π, which in effect requires simulating all possible realisations of one step of the auxiliary Markov chain, and is therefore intractable in general. To circumvent this issue, we will move from a

[1] The set $\mathcal{G}$ is symmetric if $-g \in \mathcal{G}$ whenever $g \in \mathcal{G}$; this allows us to avoid taking absolute values in the definition of the integral probability metric.

discrete-time auxiliary Markov chain to a continuous-time auxiliary Markov process, with time t transition kernel T_π^t and associated linear operator L_π^t and discrepancy D_π^t. The contraction property (6.7) in this case reads

$$\mathsf{d}(T_\pi^t \pi, T_\pi^t \nu) \leq \rho_t \, \mathsf{d}(\pi, \nu). \tag{6.10}$$

Considering the $t \downarrow 0$ limit we may, if the auxiliary Markov process mixes rapidly enough, obtain an expression for the discrepancy in terms of the generator $\mathcal{L}_\pi$ of the auxiliary Markov process, which we recall from Section 5.2.3 is defined through its action on suitably regular test functions $g : \mathbb{R}^d \to \mathbb{R}$ as

$$(\mathcal{L}_\pi g)(\cdot) := \lim_{t \downarrow 0} \frac{1}{t} L_\pi^t g(\cdot).$$

Indeed, assume that $\rho_t = e^{-ct}$ for some $c > 0$. Then, in a purely formal manipulation,

$$\begin{aligned} D_\pi(\nu) &:= \lim_{t \downarrow 0} D_\pi^t(\nu) \\ &= \sup_{g \in \mathcal{G}} \int \lim_{t \downarrow 0} \frac{1}{1 - \rho_t} (L_\pi^t g)(\mathbf{x}) \, \mathrm{d}\nu(\mathbf{x}) = \frac{1}{c} \sup_{g \in \mathcal{G}} \int (\mathcal{L}_\pi g)(\mathbf{x}) \, \mathrm{d}\nu(\mathbf{x}), \end{aligned} \tag{6.11}$$

where the final step uses the definition of the generator $\mathcal{L}_\pi$ and the fact that $e^{-ct} = 1 - ct + o(t)$ when t is small. Intriguingly, this form of discrepancy may be computable, up to the rate constant c, which will be unknown in general. The remaining challenges appear to be the selection of a suitable auxiliary Markov process, for which the contraction property (6.10) is satisfied, and the solution of the optimisation problem over $\mathcal{G}$. These issues are addressed, respectively, in Sections 6.2.2 and 6.2.3.

Remark (Stein discrepancies) The discrepancy that we have introduced in (6.11) is an instance of *Stein discrepancy*, as defined in the pioneering work of Gorham and Mackey (2015). A *Stein discrepancy* refers to any discrepancy of the form

$$\sup_{g \in \mathcal{G}'} \int (\mathcal{A}_\pi g)(\mathbf{x}) \, \mathrm{d}\nu \tag{6.12}$$

where the the *Stein operator* $\mathcal{A}_\pi$ and the *Stein class* $\mathcal{G}'$ are selected in such a way that (6.12) is zero if and only if π and ν are equal. The concept of a Stein discrepancy is more general than the discrepancy $D_\pi(\nu)$ we have constructed, since the Stein operator $\mathcal{A}_\pi$ need not arise from consideration of a continuous-time Markov process; see for example the review of Anastasiou et al. (2023).

6.2.2 Choice of Auxiliary Markov Process

To make this argument useful we require a metric d and an auxiliary continuous-time Markov process for which the contraction property (6.10) is satisfied. For the auxiliary Markov process, we will consider the overdamped Langevin diffusion from Section 1.4:

$$\mathrm{d}\mathbf{X}_t = \nabla \log \pi(\mathbf{X}_t)\, \mathrm{d}t + \sqrt{2}\, \mathrm{d}\mathbf{W}_t \tag{6.13}$$

whose infinitesimal generator is the second order differential operator $(\mathcal{L}_\pi g)(\mathbf{x}) = (\Delta g)(\mathbf{x}) + \langle (\nabla \log \pi)(\mathbf{x}), (\nabla g)(\mathbf{x}) \rangle$, where Δ denotes the Laplacian differential operator for $\mathbb{R}^d$. To simplify the presentation, we initially make the assumption that π is *strongly log-concave*, meaning that (3.18) holds for some $l > 0$, and we recall that a sufficient condition for strong log-concavity is that $-\nabla\nabla \log \pi(\mathbf{x}) > \epsilon \mathbf{I}_d$ for some $\epsilon > 0$ and all $\mathbf{x} \in \mathbb{R}^d$, where $\nabla\nabla$ denotes the Hessian differential operator and the notation $\mathbf{A} > \mathbf{B}$ is used to mean that $\mathbf{A} - \mathbf{B}$ is a symmetric positive definite matrix. This assumption will be relaxed in Section 6.2.4. Let $C^s(\mathbb{R}^d, \mathbb{R}^p)$ denote the set of functions $f : \mathbb{R}^d \to \mathbb{R}^p$ for which continuous derivatives exist of orders up to $s \in \{0, 1, \dots\} \cup \{\infty\}$. For $g \in C^0(\mathbb{R}^d, \mathbb{R}^p)$, let

$$M_1(g) := \sup_{\substack{\mathbf{x}, \mathbf{x}' \in \mathbb{R}^d \\ \mathbf{x} \neq \mathbf{x}'}} \frac{\|g(\mathbf{x}) - g(\mathbf{x}')\|}{\|\mathbf{x} - \mathbf{x}'\|} \tag{6.14}$$

denote its (possibly infinite) Lipschitz constant. Recall that a function g is called *Lipschitz* whenever $M_1(g) < \infty$. The following is a well-known contraction result for the overdamped Langevin diffusion, whose proof can be found in e.g. von Renesse and Sturm (2005), or see Remark 1 in Eberle (2016):

Theorem 6.1 (Contraction of the overdamped Langevin diffusion) *Let π be strongly log-concave and let $\nabla \log \pi$ be Lipschitz. Then the overdamped Langevin diffusion* (6.13) *satisfies the contraction property* (6.10) *in the* Wasserstein-*1 metric*

$$\mathsf{d}_{W_1}(\pi, \nu) := \sup_{\substack{g \in C^0(\mathbb{R}^d, \mathbb{R}) \\ M_1(g) \leq 1}} \int g(\mathbf{x})\, \mathrm{d}\pi(\mathbf{x}) - \int g(\mathbf{x})\, \mathrm{d}\nu(\mathbf{x}) \tag{6.15}$$

with $\rho_t = e^{-ct}$ for some $c > 0$ and all $t \in [0, \infty)$.

The infinitesimal generator $\mathcal{L}_\pi$ of the overdamped Langevin diffusion requires ∇g and Δg to exist, but the Wasserstein-1 integral probability metric contains non-differentiable functions in the test function set $\mathcal{G}$. This appears to prevent us from running the formal argument in (6.11). However, it turns

out that we may, without loss of generality, impose additional smoothness on the Wasserstein-1 test function set:

Lemma 6.2 (Smoother test functions for Wasserstein-1) *For $\pi, \nu \in \mathcal{P}(\mathbb{R}^d)$,*

$$\mathsf{d}_{W_1}(\pi, \nu) = \sup_{\substack{g \in C^\infty(\mathbb{R}^d, \mathbb{R}) \\ M_1(g) \leq 1}} \int g(\mathbf{x}) \, \mathrm{d}\pi(\mathbf{x}) - \int g(\mathbf{x}) \, \mathrm{d}\nu(\mathbf{x}). \tag{6.16}$$

Proof Since the supremum is being computed over a subset of the test functions in (6.15), it is immediate that the right-hand side of (6.16) is upper-bounded by $\mathsf{d}_{W_1}(\pi, \nu)$. To prove the corresponding lower bound, let $\epsilon \in (0, 1)$. From the definition of d_{W_1} in (6.15), there exists g_ϵ with $M_1(g_\epsilon) \leq 1$ such that $\int g_\epsilon(\mathbf{x}) \mathrm{d}\pi(\mathbf{x}) - \int g_\epsilon(\mathbf{x}) \mathrm{d}\nu(\mathbf{x}) > \mathsf{d}_{W_1}(\pi, \nu) - \epsilon$. Let $\delta > 0$ and $\mathbf{Z} \sim \mathsf{N}(\mathbf{0}, \mathbf{I}_d)$. Set $g_{\epsilon,\delta}(\mathbf{x}) = \mathbb{E}[g_\epsilon(\mathbf{x} + \delta\mathbf{Z})]$. Then $g_{\epsilon,\delta} \in C^\infty(\mathbb{R}^d, \mathbb{R})$ and the Lipschitz constant of $g_{\epsilon,\delta}$ is not greater than that of g_ϵ, since for all $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^d$,

$$\begin{aligned} |g_{\epsilon,\delta}(\mathbf{x}) - g_{\epsilon,\delta}(\mathbf{x}')| &= |\mathbb{E}[g_\epsilon(\mathbf{x} + \delta\mathbf{Z}) - g_\epsilon(\mathbf{x}' + \delta\mathbf{Z})]| \\ &\leq M_1(g_\epsilon)\|\mathbf{x} - \mathbf{x}'\|. \end{aligned}$$

Thus $g_{\epsilon,\delta}$ is an element of the test set over which the supremum is computed on the right hand side of (6.16). From

$$|g_{\epsilon,\delta}(\mathbf{x}) - g_\epsilon(\mathbf{x})| = |\mathbb{E}[g_\epsilon(\mathbf{x} + \delta\mathbf{Z}) - g_\epsilon(\mathbf{x})]| \leq \delta\mathbb{E}[\|\mathbf{Z}\|]M_1(g_\epsilon), \tag{6.17}$$

it follows that $g_{\epsilon,\delta}$ distinguishes between π and ν almost as well as g_ϵ, in the sense that

$$\begin{aligned} &\int g_{\epsilon,\delta}(\mathbf{x}) \, \mathrm{d}\pi(\mathbf{x}) - \int g_{\epsilon,\delta}(\mathbf{x}) \, \mathrm{d}\nu(\mathbf{x}) \\ &\geq \int g_\epsilon(\mathbf{x}) \, \mathrm{d}\pi(\mathbf{x}) - \int g_\epsilon(\mathbf{x}) \, \mathrm{d}\nu(\mathbf{x}) - 2\delta\mathbb{E}[\|\mathbf{Z}\|]M_1(g_\epsilon) \\ &> \{\mathsf{d}_{W_1}(\pi, \nu) - \epsilon - 2\delta\mathbb{E}[\|\mathbf{Z}\|]\}, \end{aligned}$$

which can be made arbitrarily close to $\mathsf{d}_{W_1}(\pi, \nu)$ by taking $\epsilon, \delta \to 0$. Thus the supremum in (6.16) coincides with $\mathsf{d}_{W_1}(\pi, \nu)$, as claimed. □

To summarise, our formal argument has led to a bound

$$D_\pi(\nu) = \frac{1}{c} \sup_{\substack{g \in C^\infty(\mathbb{R}^d, \mathbb{R}) \\ M_1(g) \leq 1}} \int \Delta g(\mathbf{x}) + \langle \nabla \log \pi(\mathbf{x}), \nabla g(\mathbf{x}) \rangle \, \mathrm{d}\nu(\mathbf{x}) \tag{6.18}$$

on the Wasserstein-1 distance between π and ν that holds in the strongly log-concave setting of Theorem 6.1. The route to obtaining this bound

is instructive, and the lessons that we learned will be exploited in the subsequent sections, but unfortunately, the evaluation of this discrepancy requires a challenging optimisation problem to be solved. In the case where ν has finite support, the objective function depends on g only through its derivatives at the nodes in the support. This observation enabled Gorham and Mackey (2015) to cast a closely related optimisation problem as a collection of linear programmes, which then can be numerically solved. The interested reader is referred to Gorham and Mackey (2015) for further detail. However, the reliance on numerical methods to evaluate (6.18) limits the utility of (6.18). Instead, we will proceed in Section 6.2.3 to consider alternative sets of test functions for which the corresponding optimisation problem can be *analytically* solved.

6.2.3 Kernel Stein Discrepancy

The aim of this section is to consider alternatives to the Wasserstein-1 distance, corresponding to alternative sets $\mathcal{G}$ of test functions defining the integral probability metric (6.8), for which the optimisation problem in (6.9) can be explicitly solved using the kernel trick advertised in Section 6.1.3. However, the use of alternative metrics leads us to depart from the argument of Section 6.2.1, which was based on the Wasserstein-1 contraction result of Theorem 6.1, raising the question of whether the resulting discrepancy is still a meaningful convergence bound. This question will be answered positively in Section 6.2.4.

To simplify the discussion, we start by considering vector fields as test functions, allowing us to reduce the order of the differential operators involved. Thus, in the general notation of (6.12), we consider

$$(\mathcal{A}_\pi \mathbf{g})(\mathbf{x}) = (\nabla \cdot \mathbf{g})(\mathbf{x}) + \langle (\nabla \log \pi)(\mathbf{x}), \mathbf{g}(\mathbf{x}) \rangle, \tag{6.19}$$

which is a *first* order differential operator and the elements $\mathbf{g}$ are now vector fields $\mathbf{g} : \mathbb{R}^d \to \mathbb{R}^d$. The discussion in Section 6.2.2 corresponds to $\mathbf{g}(\mathbf{x}) = (\nabla g)(\mathbf{x})$ for twice-differentiable $g : \mathbb{R}^d \to \mathbb{R}$. Here, and in the sequel, for ease of presentation, we have subsumed the constant factor $1/c$ into the definition of the vector fields $\mathbf{g}$. Now, if we are to consider alternative test functions $\mathbf{g}$, the minimum requirement on $\mathbf{g}$ is that $\mathcal{A}_\pi \mathbf{g}$ integrates to 0 with respect to π, to ensure that the discrepancy we construct vanishes when π and ν are equal. To this end, we have the following result:

Lemma 6.3 *Let* $\mathbf{g} : \mathbb{R}^d \to \mathbb{R}^d$ *satisfy* $\mathbf{g} \in \mathcal{L}^1(\pi)$ *and* $\mathcal{A}_\pi \mathbf{g} \in \mathcal{L}^1(\pi)$, *where* $\mathcal{A}_\pi$ *is defined in* (6.19). *Then* $\int (\mathcal{A}_\pi \mathbf{g})(\mathbf{x}) \, \mathrm{d}\pi(\mathbf{x}) = 0$.

Proof First notice that

$$\int (\mathcal{A}_\pi \mathbf{g})(\mathbf{x})\, \mathrm{d}\pi(\mathbf{x}) = \int \frac{1}{\pi(\mathbf{x})}(\nabla \cdot (\pi\mathbf{g}))(\mathbf{x})\, \mathrm{d}\pi(\mathbf{x}) = \int (\nabla \cdot (\pi\mathbf{g}))(\mathbf{x})\, \mathrm{d}\mathbf{x},$$

which suggests using the divergence theorem to calculate this integral. To avoid the explicit calculation of surface integrals, which would otherwise be required when using the divergence theorem, we will first approximate the vector field $\pi\mathbf{g}$ using another vector field with compact support. Let $\varphi_m : \mathbb{R}^d \to \mathbb{R}$ denote the mth term in a sequence of compactly supported functions with $\varphi_m(\mathbf{x}) = 1$ for $\|\mathbf{x}\| \leq m$, $\sup_{\mathbf{x}} \|\nabla\varphi_m(\mathbf{x})\| < m^{-1}$ for each $m \in \mathbb{N}$, and $\varphi_m(\mathbf{x}) \uparrow 1$ for each $\mathbf{x} \in \mathbb{R}^d$. From the divergence theorem applied to a vector field with compact support,

$$\begin{aligned} 0 &= \int (\nabla \cdot (\varphi_m \pi\mathbf{g}))(\mathbf{x})\, \mathrm{d}\mathbf{x} \\ &= \int \langle \nabla\varphi_m(\mathbf{x}), (\pi\mathbf{g})(\mathbf{x})\rangle\, \mathrm{d}\mathbf{x} + \int \varphi_m(\mathbf{x})(\nabla \cdot (\pi\mathbf{g}))(\mathbf{x})\, \mathrm{d}\mathbf{x}. \end{aligned}$$

Since $\varphi_m \uparrow 1$ pointwise and $\nabla \cdot (\pi\mathbf{g}) \in \mathcal{L}^1(\mathbb{R}^d)$, from the dominated convergence theorem

$$\int \varphi_m(\mathbf{x})(\nabla \cdot (\pi\mathbf{g}))(\mathbf{x})\, \mathrm{d}\mathbf{x} \to \int (\nabla \cdot (\pi\mathbf{g}))(\mathbf{x})\, \mathrm{d}\mathbf{x}.$$

On the other hand, using Cauchy–Schwarz and the assumption that $\pi\mathbf{g} \in \mathcal{L}^1(\mathbb{R}^d)$,

$$\left| \int \langle \nabla\varphi_m(\mathbf{x}), (\pi\mathbf{g})(\mathbf{x})\rangle\, \mathrm{d}\mathbf{x} \right| \leq \left(\sup_{\mathbf{x}} \|\nabla\varphi_m(\mathbf{x})\| \right) \int \|(\pi\mathbf{g})(\mathbf{x})\|\, \mathrm{d}\mathbf{x} \to 0.$$

Thus we have shown that

$$\int (\mathcal{A}_\pi \mathbf{g})(\mathbf{x})\, \mathrm{d}\pi(\mathbf{x}) = \int (\nabla \cdot (\pi\mathbf{g}))(\mathbf{x})\, \mathrm{d}\mathbf{x} = 0,$$

as claimed. □

Our attention now turns to selecting a set of vector fields $\mathbf{g}$ for which Lemma 6.3 holds and for which the optimisation problem in (6.18) can be explicitly solved. One approach to this task is to use a *matrix-valued reproducing kernel*, meaning a function $\mathsf{K} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^{d\times d}$ that is

1. *transpose-symmetric*; $\mathsf{K}(\mathbf{x}, \mathbf{x}') = \mathsf{K}(\mathbf{x}', \mathbf{x})^\top$ for all $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^d$
2. *positive semi-definite*;

$$\sum_{k=1}^{n} \sum_{k'=1}^{n} \langle \mathbf{c}_k, \mathsf{K}(\mathbf{x}_k, \mathbf{x}_{k'})\mathbf{c}_{k'} \rangle \geq 0$$

for all $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$, all $\mathbf{c}_1, \ldots, \mathbf{c}_n \in \mathbb{R}^d$, and all $n \in \mathbb{N}$.

For clarity, we emphasise that $\langle \mathbf{c}, \mathbf{c}' \rangle = \mathbf{c}^\top \mathbf{c}'$ is the usual Euclidean inner product on $\mathbb{R}^n$; in the sequel we will use subscripts to distinguish other inner products as they are introduced. Let $\mathsf{K}_\mathbf{x} = \mathsf{K}(\cdot, \mathbf{x})$, so that $\mathsf{K}_\mathbf{x} : \mathbb{R}^d \to \mathbb{R}^{d \times d}$ is matrix-valued. For vector-valued functions $\mathbf{g} = \sum_{k=1}^n \mathsf{K}_{\mathbf{x}_k} \mathbf{c}_k$ and $\mathbf{g}' = \sum_{l=1}^m \mathsf{K}_{\mathbf{x}'_l} \mathbf{c}'_l$, define an inner product

$$\langle \mathbf{g}, \mathbf{g}' \rangle_{\mathcal{H}(\mathsf{K})} = \sum_{k=1}^n \sum_{l=1}^m \langle \mathbf{c}_k, \mathsf{K}(\mathbf{x}_k, \mathbf{x}'_l) \mathbf{c}'_l \rangle. \tag{6.20}$$

There is a unique Hilbert space reproduced by K, denoted $\mathcal{H}(\mathsf{K})$; see Proposition 2.1 of Carmeli et al. (2006). This space is characterised as

$$\mathcal{H}(\mathsf{K}) = \overline{\text{span}}\{\mathsf{K}_\mathbf{x} \mathbf{c} : \mathbf{x}, \mathbf{c} \in \mathbb{R}^d\}$$

where here the closure is taken with respect to the inner product in (6.20). The resulting Hilbert space satisfies the *reproducing property*

$$\langle \mathbf{g}, \mathsf{K}_\mathbf{x} \mathbf{c} \rangle_{\mathcal{H}(\mathsf{K})} = \langle \mathbf{g}(\mathbf{x}), \mathbf{c} \rangle$$

for all $\mathbf{g} \in \mathcal{H}(\mathsf{K})$ and $\mathbf{x}, \mathbf{c} \in \mathbb{R}^d$, which is a particular instance of the kernel trick discussed in Section 6.1.3. In what follows, it is convenient to overload notation, such that the reproducing property becomes $\langle \mathbf{g}, \mathsf{K}_\mathbf{x} \rangle_{\mathcal{H}(\mathsf{K})} = \mathbf{g}(\mathbf{x})$ in an informal shorthand.

Assuming sufficient regularity that

$$\begin{aligned} F_\nu : \mathcal{H}(\mathsf{K}) &\to \mathbb{R} \\ \mathbf{g} &\mapsto \int (\mathcal{A}_\pi \mathbf{g})(\mathbf{x}) \, \mathrm{d}\nu(\mathbf{x}) \end{aligned}$$

is a bounded linear functional, the Riesz representation theorem tells us that there is a unique element μ_ν such that $F_\nu(\cdot) = \langle \mu_\nu, \cdot \rangle_{\mathcal{H}(\mathsf{K})}$. Using our reproducing property shorthand,

$$\mu_\nu(\mathbf{x}') = \langle \mu_\nu, \mathsf{K}_{\mathbf{x}'} \rangle_{\mathcal{H}(\mathsf{K})} = F_\nu(\mathsf{K}_{\mathbf{x}'}) = \int \mathcal{A}_\pi^\mathbf{x} \mathsf{K}(\mathbf{x}, \mathbf{x}') \mathrm{d}\nu(\mathbf{x}),$$

where the superscript in $\mathcal{A}_\pi^\mathbf{x}$ indicates the action of $\mathcal{A}_\pi$ on the $\mathbf{x}$ argument, collapsing the matrix-valued function $\mathsf{K}_\mathbf{x}$ into the vector-valued function $\mathcal{A}_\pi^\mathbf{x} \mathsf{K}_\mathbf{x}$. It follows that, if we consider the collection of vector fields $\mathbf{g}$ within the unit ball of $\mathcal{H}(\mathsf{K})$, our optimisation problem can be explicitly solved:

$$\sup_{\|\mathbf{g}\|_{\mathcal{H}(\mathsf{K})} \leq 1} \int (\mathcal{A}_\pi \mathbf{g})(\mathbf{x}) \, \mathrm{d}\nu(\mathbf{x}) = \sup_{\|\mathbf{g}\|_{\mathcal{H}(\mathsf{K})} \leq 1} \langle \mathbf{g}, \mu_\nu \rangle_{\mathcal{H}(\mathsf{K})} = \|\mu_\nu\|_{\mathcal{H}(\mathsf{K})}, \tag{6.21}$$

where, again from the reproducing property and the assumption that F_ν is a bounded linear functional,

$$\begin{aligned}\|\mu_\nu\|^2_{\mathcal{H}(\mathsf{K})} &= \left\langle \int \mathcal{A}^{\mathbf{x}}_\pi \mathsf{K}_{\mathbf{x}} \, \mathrm{d}\nu(\mathbf{x}), \int \mathcal{A}^{\mathbf{x}'}_\pi \mathsf{K}_{\mathbf{x}'} \, \mathrm{d}\nu(\mathbf{x}') \right\rangle_{\mathcal{H}(\mathsf{K})} \\ &= \iint \mathcal{A}^{\mathbf{x}}_\pi \mathcal{A}^{\mathbf{x}'}_\pi \langle \mathsf{K}_{\mathbf{x}}, \mathsf{K}_{\mathbf{x}'} \rangle_{\mathcal{H}(\mathsf{K})} \, \mathrm{d}\nu(\mathbf{x}) \mathrm{d}\nu(\mathbf{x}') \\ &= \iint \mathcal{A}^{\mathbf{x}}_\pi \mathcal{A}^{\mathbf{x}'}_\pi \mathsf{K}(\mathbf{x}, \mathbf{x}') \, \mathrm{d}\nu(\mathbf{x}) \mathrm{d}\nu(\mathbf{x}').\end{aligned}$$

It is convenient to introduce the shorthand $k_\pi(\mathbf{x}, \mathbf{x}') := \mathcal{A}^{\mathbf{x}}_\pi \mathcal{A}^{\mathbf{x}'}_\pi \mathsf{K}(\mathbf{x}, \mathbf{x}')$, whence we obtain the discrepancy

$$\mathcal{D}_{k_\pi}(\nu) := \sqrt{\iint k_\pi(\mathbf{x}, \mathbf{x}') \, \mathrm{d}\nu(\mathbf{x}) \mathrm{d}\nu(\mathbf{x}')}, \tag{6.22}$$

which is exactly of the form we sought in (6.6). This was termed a *kernel Stein discrepancy* in Chwialkowski et al. (2016); Liu et al. (2016), due to its dependence on a reproducing kernel and its characterisation as a Stein discrepancy (6.1). A second consequence of (6.21) is that we can view the kernel Stein discrepancy as a generalised moment

$$\mathcal{D}_{k_\pi}(\nu_n) = \left\| \frac{1}{n} \sum_{k=1}^{n} \mathcal{A}^{\mathbf{x}}_\pi \mathsf{K}_{\mathbf{x}} \big|_{\mathbf{x}=\mathbf{x}_k} \, \mathrm{d}\nu(\mathbf{x}) \right\|_{\mathcal{H}(\mathsf{K})},$$

which takes a similar form to (6.4) from Section 6.1.2, albeit the generalised moment can now be infinite-dimensional by virtue of taking values in $\mathcal{H}(\mathsf{K})$. The function $\mathcal{A}^{\mathbf{x}}_\pi \mathsf{K}_{\mathbf{x}}$ is indeed a member of $\mathcal{H}(\mathsf{K})$ due to the *differential reproducing property* (see Barp et al., 2022, Appendix C6). The kernel Stein discrepancy has the potential to be a useful bias diagnostic, but first we need to establish that it has our basic desired functionality, such as being equal to 0 if and only if π and ν are identical. Clearly, then the choice of kernel K will be important, so we address this point next.

One of the simplest forms of matrix-valued reproducing kernel is $\mathsf{K}(\mathbf{x}, \mathbf{x}') = \mathsf{k}(\mathbf{x}, \mathbf{x}')\mathbf{I}_d$, where k is a scalar-valued reproducing kernel. This choice leads to the explicit formula, due to Oates et al. (2017):

$$\begin{aligned}\mathsf{k}_\pi(\mathbf{x}, \mathbf{x}') = \nabla_{\mathbf{x}} \cdot \nabla_{\mathbf{x}'} \mathsf{k}(\mathbf{x}, \mathbf{x}') &+ \langle \nabla_{\mathbf{x}} \log \pi(\mathbf{x}), \nabla_{\mathbf{x}'} \mathsf{k}(\mathbf{x}, \mathbf{x}') \rangle \\ &+ \langle \nabla_{\mathbf{x}'} \log \pi(\mathbf{x}'), \nabla_{\mathbf{x}} \mathsf{k}(\mathbf{x}, \mathbf{x}') \rangle \\ &+ \langle \nabla_{\mathbf{x}} \log \pi(\mathbf{x}), \nabla_{\mathbf{x}'} \log \pi(\mathbf{x}') \rangle \mathsf{k}(\mathbf{x}, \mathbf{x}')\end{aligned} \tag{6.23}$$

The function k_π is automatically a scalar-valued reproducing kernel (see

Barp et al., 2022, Theorem 2.6), and $\mathsf{k}_\pi(\mathbf{x}, \mathbf{x}') = \mathcal{A}_\pi^{\mathbf{x}} \mathbf{g}_{\mathbf{x}'}(\mathbf{x})$ where $\mathbf{g}_{\mathbf{x}'}(\mathbf{x}) := \mathcal{A}_\pi^{\mathbf{x}'} \mathsf{K}(\mathbf{x}, \mathbf{x}') \in \mathcal{H}(\mathsf{K})$. Thus, *if* the matrix-valued reproducing kernel K is selected such that the conditions $\mathbf{g} \in \mathcal{L}^1(\pi)$ and $\mathcal{A}_\pi \mathbf{g} \in \mathcal{L}^1(\pi)$ of Lemma 6.3 are satisfied for each $\mathbf{g} \in \mathcal{H}(\mathsf{K})$, it follows that $\int \mathsf{k}_\pi(\mathbf{x}, \mathbf{x}') \mathrm{d}\pi(\mathbf{x}) = \int \mathcal{A}_\pi^{\mathbf{x}} \mathbf{g}_{\mathbf{x}'}(\mathbf{x}) \, \mathrm{d}\pi(\mathbf{x}) = 0$ for all $\mathbf{x}' \in \mathbb{R}^d$. Sufficient conditions for satisfying the preconditions of Lemma 6.3 will shortly be discussed.

In the case where $\nu = \sum_{k=1}^n w_k \delta_{\mathbf{x}_k}$ is a discrete distribution, (6.22) reduces to the double sum

$$\mathcal{D}_{\mathsf{k}_\pi}(\nu) = \sqrt{\sum_{k=1}^n \sum_{k'=1}^n w_k w_{k'} \mathsf{k}_\pi(\mathbf{x}_k, \mathbf{x}_{k'})}. \tag{6.24}$$

For the degenerate reproducing kernel with $\mathsf{k}(\mathbf{x}, \mathbf{x}') = 1$ for all $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^d$ and uniform weights $w_k = n^{-1}$, the kernel Stein discrepancy in (6.24) reduces to the simple form

$$\mathcal{D}_{\mathsf{k}_\pi}(\nu) = \left\| \frac{1}{n} \sum_{k=1}^n \nabla_{\mathbf{x}} \log \pi(\mathbf{x}_k) \right\|,$$

which is precisely the bias diagnostic from (6.4). In this case, $\mathcal{H}(\mathsf{K})$ is the Hilbert space of constant vector fields on $\mathbb{R}^d$ with norm $\|\mathbf{g}\|_{\mathcal{H}(\mathsf{K})} = \|\boldsymbol{\beta}\|$ where $\mathbf{g}(\mathbf{x}) = \boldsymbol{\beta}$ for all $\mathbf{x} \in \mathbb{R}^d$, which is insufficiently rich to determine whether or not π and ν are close or equal. However, with a suitable choice of reproducing kernel the kernel Stein discrepancy can distinguish between different distributions and indeed provide a form of convergence control, as we explain in Section 6.2.4.

First, however, we must ensure the conditions of Lemma 6.3 are satisfied, so that $\mathcal{D}_{\mathsf{k}_\pi}(\nu) = 0$ when π and ν are equal. This can be achieved using the following result:

Lemma 6.4 *If* $\mathsf{K}(\mathbf{x}, \mathbf{x}') = \mathsf{k}(\mathbf{x}, \mathbf{x}') \mathbf{I}_d$ *with* $\mathsf{k}(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x} - \mathbf{x}')$ *for some* $\phi \in C^2(\mathbb{R}^d, \mathbb{R})$, *then* $\nabla \log \pi \in \mathcal{L}^1(\pi)$ *implies* $\mathbf{g} \in \mathcal{L}^1(\pi)$ *and* $\mathcal{A}_\pi \mathbf{g} \in \mathcal{L}^1(\pi)$ *for all* $\mathbf{g} \in \mathcal{H}(\mathsf{K})$.

Proof The reproducing property, followed by Cauchy–Schwarz, gives

$$\begin{aligned} \int \|\mathbf{g}(\mathbf{x})\| \, \mathrm{d}\pi(\mathbf{x}) &= \int \|\langle \mathbf{g}, \mathsf{K}_{\mathbf{x}} \rangle_{\mathcal{H}(\mathsf{K})}\| \, \mathrm{d}\pi(\mathbf{x}) \\ &\leq \|\mathbf{g}\|_{\mathcal{H}(\mathsf{K})} \int \sqrt{\mathrm{tr}(\mathsf{K}(\mathbf{x}, \mathbf{x}))} \, \mathrm{d}\pi(\mathbf{x}) = \|\mathbf{g}\|_{\mathcal{H}(\mathsf{K})} \sqrt{d\phi(\mathbf{0})}, \end{aligned}$$

which is finite for all $\mathbf{g} \in \mathcal{H}(\mathsf{K})$. Similarly,

$$\int |(\mathcal{A}_\pi \mathbf{g})(\mathbf{x})| \, \mathrm{d}\pi(\mathbf{x}) = \int |\langle \mathbf{g}, \mathcal{A}_\pi^{\mathbf{x}} \mathsf{K}_{\mathbf{x}} \rangle_{\mathcal{H}(\mathsf{K})}| \, \mathrm{d}\pi(\mathbf{x}) \leq \|\mathbf{g}\|_{\mathcal{H}(\mathsf{K})} \int \sqrt{\mathcal{A}_\pi^{\mathbf{x}} \mathcal{A}_\pi^{\mathbf{x}'} \mathsf{K}(\mathbf{x}, \mathbf{x}')|_{\mathbf{x}'=\mathbf{x}}} \, \mathrm{d}\pi(\mathbf{x}).$$

Here the assumption $\phi \in C^2(\mathbb{R}^d, \mathbb{R})$ ensures the application of $\mathcal{A}_\pi^{\mathbf{x}} \mathcal{A}_\pi^{\mathbf{x}'}$ to $\mathsf{K}(\mathbf{x}, \mathbf{x}')$ is well-defined. Specialising to the translation-invariant reproducing kernel in the statement, we have

$$\mathcal{A}_\pi^{\mathbf{x}} \mathcal{A}_\pi^{\mathbf{x}'} \mathsf{K}(\mathbf{x}, \mathbf{x}')|_{\mathbf{x}'=\mathbf{x}} = -(\Delta\phi)(\mathbf{0}) + \phi(\mathbf{0}) \|(\nabla \log \pi)(\mathbf{x})\|^2 \tag{6.25}$$

which shows that $\mathcal{A}_\pi \mathbf{g} \in \mathcal{L}^1(\pi)$ whenever $\nabla \log \pi \in \mathcal{L}^1(\pi)$, and completes the argument. □

The kernel Stein discrepancies we have just constructed are well-defined and computable, but we have not yet addressed the question of if and how the values of the discrepancy $\mathcal{D}_{\mathsf{k}_\pi}(\nu)$ are related to the closeness of π and ν. Indeed, since we have used alternative test functions compared to (6.18), we cannot expect $\mathcal{D}_{\mathsf{k}_\pi}(\nu)$ to provide an upper bound on the Wasserstein-1 distance between π and ν. The next section explains to what extent kernel Stein discrepancies relate to the closeness of π and ν.

6.2.4 Convergence Control

The aim of this section is to establish whether kernel Stein discrepancies can provide control over integral probability metrics. At the same time, we will weaken the strong log-concavity assumption from Section 6.2.2 to an assumption that π is *distantly dissipative*, meaning that

$$\liminf_{r \to \infty} \inf_{\substack{\mathbf{x}, \mathbf{x}' \in \mathbb{R}^d \\ \|\mathbf{x} - \mathbf{x}'\| = r}} \left\{ -\frac{\langle (\nabla \log \pi)(\mathbf{x}) - (\nabla \log \pi)(\mathbf{x}'), \mathbf{x} - \mathbf{x}' \rangle}{\|\mathbf{x} - \mathbf{x}'\|^2} \right\} > 0,$$

for which Wasserstein-1 contraction of the overdamped Langevin diffusion (6.13) can still be established (see Lindvall and Rogers, 1986; Eberle, 2016). The next Lemma demonstrates that distant dissipativity is indeed a generalisation of strong log-concavity:

Lemma 6.5 *If $\nabla \log \pi$ is bounded on a compact set $S \subset \mathbb{R}^d$ and π is strongly log-concave on the boundary of and outside of the set S, then π is distantly dissipative.*

Proof Let $\mathbf{b}(\mathbf{x}) := (\nabla \log \pi)(\mathbf{x})$, let S be the compact set in the statement, and let $\mathrm{int}(S)$ denote the interior of S. From the strong log-concavity assumption, there exists $c > 0$ such that for all $\mathbf{x}, \mathbf{x}' \notin \mathrm{int}(S)$ we have

$$-\frac{\langle \mathbf{b}(\mathbf{x}) - \mathbf{b}(\mathbf{x}'), \mathbf{x} - \mathbf{x}' \rangle}{\|\mathbf{x} - \mathbf{x}'\|^2} > c. \tag{6.26}$$

Since $\mathbf{b}$ is bounded on S, we may pick $B > \sup_{\mathbf{x} \in S} \|\mathbf{b}(\mathbf{x})\|$. Since S is compact, S is contained in $\{\mathbf{x} : \|\mathbf{x}\| \leq r/2\}$ for some sufficiently large $r > 0$, and we may suppose that $r > 2B/c$, so $c' := c - (2B/r) > 0$.

Consider arbitrary $\mathbf{x}, \mathbf{x}'$ such that $\|\mathbf{x} - \mathbf{x}'\| > r$. If $\mathbf{x}, \mathbf{x}' \notin S$, condition (6.26) is satisfied. Thus consider the other case, where without loss of generality $\mathbf{x} \in S$ (and thus $\mathbf{x}' \notin S$). Let $\mathbf{x}''$ be the closest point to $\mathbf{x}$ that belongs to S and is colinear with $\mathbf{x}$ and $\mathbf{x}'$. Then

$$\begin{aligned}
-\langle \mathbf{b}(\mathbf{x}) - \mathbf{b}(\mathbf{x}'), \mathbf{x} - \mathbf{x}' \rangle &= -\langle \mathbf{b}(\mathbf{x}) - \mathbf{b}(\mathbf{x}''), \mathbf{x} - \mathbf{x}' \rangle - \langle \mathbf{b}(\mathbf{x}'') - \mathbf{b}(\mathbf{x}'), \mathbf{x} - \mathbf{x}' \rangle \\
&= -\|\mathbf{x} - \mathbf{x}'\| \left\langle \mathbf{b}(\mathbf{x}) - \mathbf{b}(\mathbf{x}''), \frac{\mathbf{x} - \mathbf{x}''}{\|\mathbf{x} - \mathbf{x}''\|} \right\rangle \\
&\quad - \frac{\|\mathbf{x} - \mathbf{x}'\|}{\|\mathbf{x}'' - \mathbf{x}'\|} \langle \mathbf{b}(\mathbf{x}'') - \mathbf{b}(\mathbf{x}'), \mathbf{x}'' - \mathbf{x}' \rangle \\
&> -\|\mathbf{x} - \mathbf{x}'\| \cdot 2B \; + \; 1 \cdot c \|\mathbf{x}'' - \mathbf{x}'\|^2
\end{aligned}$$

where in the final line the Cauchy–Schwarz and triangle inequalities were used to bound the first term, and for the second term (6.26) was applied to $\mathbf{x}', \mathbf{x}'' \notin \mathrm{int}(S)$. Thus

$$-\frac{\langle \mathbf{b}(\mathbf{x}) - \mathbf{b}(\mathbf{x}'), \mathbf{x} - \mathbf{x}' \rangle}{\|\mathbf{x} - \mathbf{x}'\|^2} > -\frac{2B}{\|\mathbf{x} - \mathbf{x}'\|} + c > -\frac{2B}{r} + c = c'.$$

Combining these two results, we have shown that for all $\mathbf{x}, \mathbf{x}'$ with $\|\mathbf{x} - \mathbf{x}'\| > r$,

$$-\frac{\langle \mathbf{b}(\mathbf{x}) - \mathbf{b}(\mathbf{x}'), \mathbf{x} - \mathbf{x}' \rangle}{\|\mathbf{x} - \mathbf{x}'\|^2} > \min(c, c') > 0$$

and thus the distant dissipativity of π is established. □

The most commonly used kernel Stein discrepancies do not offer control of the Wasserstein-1 distance, though it is possible, through a careful choice of kernel, to obtain Wasserstein-1 control; we return to this point at the end of the present section. Rather, the most common kernel Stein discrepancies offer control of the (weaker) *Dudley metric*

$$\mathsf{d}_D(\pi, \nu) := \sup_{\substack{g \in C^0(\mathbb{R}^d, \mathbb{R}) \\ M_0(g) + M_1(g) \leq 1}} \int g(\mathbf{x}) \, \mathrm{d}\pi(\mathbf{x}) - \int g(\mathbf{x}) \, \mathrm{d}\nu(\mathbf{x})$$

on $\mathcal{P}(\mathbb{R}^d)$, at least in certain scenarios where π is distantly dissipative and the reproducing kernel k is carefully selected. For a bounded function $g : \mathbb{R}^d \to \mathbb{R}^p$, let $M_0(g) = \sup_{\mathbf{x}\in\mathbb{R}^d} \|g(\mathbf{x})\|$ and recall that $M_1(g)$ denotes the Lipschitz constant, from (6.14).

Theorem 6.6 (Weak convergence control; Theorem 8 of Gorham and Mackey, 2017) *Let π be distantly dissipative and $\nabla \log \pi$ be Lipschitz. Let $\mathsf{K}(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x} - \mathbf{x}')\mathbf{I}_d$ where $\phi(\mathbf{z}) = (1 + \|\mathbf{z}/\sigma\|^2)^{-\beta}$ for some $\sigma > 0$, $\beta \in (0, 1)$. Then $\mathcal{D}_{\mathsf{k}_\pi}(\nu_n) \to 0$ implies that $\mathsf{d}_D(\pi, \nu_n) \to 0$.*

Of course, if $\nabla \log \pi$ is Lipschitz, then $\nabla \log \pi$ is automatically bounded on any compact set. The set of test functions that define the Dudley metric is smaller than that for the Wasserstein-1 metric, and as a result the Dudley metric is weaker than the Wasserstein-1 metric. Indeed, the Dudley metric actually metrises the so-called *weak convergence* of distributions, meaning that ν_n converges weakly (or *in distribution*) to π if and only if $\mathsf{d}_D(\pi, \nu_n) \to 0$. The kernel Stein discrepancy itself does not provide an upper bound on d_D in this context, but an explicit nonlinear transformation of the kernel Stein discrepancy *does* still constitute an explicit upper bound. See Gorham and Mackey (2017) for full details.

The reproducing kernel appearing in Theorem 6.6 is called the *inverse multi-quadric* reproducing kernel, and its use was not accidental. The careful analysis in Theorem 6 of Gorham and Mackey (2017) demonstrates that reproducing kernels with lighter tails can fail to control weak convergence, at least in dimension $d \geq 3$. Moreover, the inverse multi-quadric reproducing kernel is computationally straightforward, and in fact the property of weak convergence control extends to the parametric family of inverse multi-quadric reproducing kernels of the form

$$\mathsf{K}_{\mathrm{IMQ}}(\mathbf{x}, \mathbf{x}') = \left(1 + \|\boldsymbol{\Sigma}^{-1/2}(\mathbf{x} - \mathbf{x}')\|^2\right)^{-\beta}\mathbf{I}_d, \qquad \boldsymbol{\Sigma} \succ 0,\ \beta \in (0, 1), \quad (6.27)$$

where $\boldsymbol{\Sigma}$ is a symmetric positive definite matrix, with the former case recovered when $\boldsymbol{\Sigma} = \mathbf{I}_d$; see Theorem 4 in Chen et al. (2019). For the extended family of inverse multi-quadric reproducing kernels in (6.27), the explicit form in (6.23) becomes

$$\begin{aligned}\mathsf{k}_\pi(\mathbf{x}, \mathbf{x}') = &-\frac{4\beta(\beta+1)\|\boldsymbol{\Sigma}^{-1}(\mathbf{x} - \mathbf{x}')\|^2}{(1 + \|\boldsymbol{\Sigma}^{-1/2}(\mathbf{x} - \mathbf{x}')\|^2)^{\beta+2}} \\ &+ 2\beta\left[\frac{\mathrm{tr}(\boldsymbol{\Sigma}^{-1}) + \langle(\nabla \log \pi)(\mathbf{x}) - (\nabla \log \pi)(\mathbf{x}'), \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \mathbf{x}')\rangle}{(1 + \|\boldsymbol{\Sigma}^{-1/2}(\mathbf{x} - \mathbf{x}')\|^2)^{\beta+1}}\right] \\ &+ \frac{\langle(\nabla \log \pi)(\mathbf{x}), (\nabla \log \pi)(\mathbf{x}')\rangle}{(1 + \|\boldsymbol{\Sigma}^{-1/2}(\mathbf{x} - \mathbf{x}')\|^2)^{\beta}},\end{aligned}$$

which can be readily computed provided that the gradient $\nabla \log \pi$ can be pointwise evaluated.

To illustrate the performance of kernel Stein discrepancies, consider again the Gaussian mixture distribution π from (6.3). This distribution is distantly dissipative[2] and has a log-density that is Lipschitz, meaning we are in the setting of Theorem 6.6. Proceeding with the inverse multiquadric reproducing kernel (6.27) with parameters $\Sigma = 1$, $\beta = 0.5$, we therefore have a guarantee that convergence of the kernel Stein discrepancy $\mathcal{D}_{\mathrm{k}_\pi}(\nu_n)$ to 0 implies the weak convergence of ν_n to π. In what follows we let $\nu_n = \frac{1}{n}\sum_{k=1}^{n} \delta_{X_k}$ be the empirical distribution associated to a Markov chain sample path $(X_k)_{1\leq k\leq n}$ and use kernel Stein discrepancy to determine whether or not ν_n converges to π. The left panel of Figure 6.3 displays typical realisations of the Markov chain sample path (top), while underneath the associated kernel Stein discrepancy (as a function of n) is displayed. In addition, the figure includes corresponding results for a Markov chain that leaves only the first component of π invariant (right), and thus does not provide a consistent approximation of π. Asymptotically, it can be seen that the kernel Stein discrepancy correctly distinguishes between the two scenarios, in which the Markov chain does and does not leave π invariant. However, focusing on the biased Markov chain, at small sample sizes the discrepancy does not detect that the chain has only explored one mixture component, and the discrepancy appears to decrease smoothly as more samples are collected. It is only once a sufficient number of samples have been collected that the failure of the Markov chain to explore the second mixture component is detected, and the discrepancy ceases decreasing to reflect that.

The small n behaviour of the kernel Stein discrepancy observed in Figure 6.3 has been termed *blindness to mixing proportions* in Wenliang and Kanagawa (2021), and provides an important note of caution that, when using kernel Stein discrepancies to assess MCMC output, the failure of the Markov chain to explore distant high-probability regions may only be detected if the length n of the Markov chain output is large enough. Our formal argument in Section 6.2.1 provides insight into this phenomenon; the Wasserstein-1 contraction rate constant of the overdamped Langevin diffusion, denoted c in (6.11), can be extremely small for distributions such as π for which the blindness phenomenon is encountered, since for this diffusion a move between the effective support of the distinct mixture components

[2] The class of distantly dissipative distributions includes all finite Gaussian mixtures with common covariance matrix; see Gorham et al. (2019).

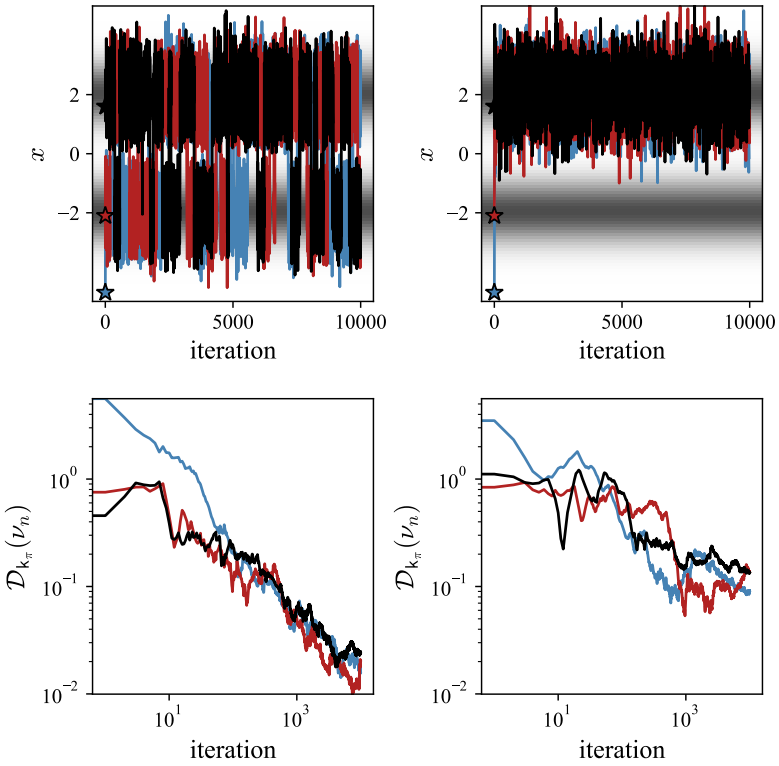

Figure 6.3 Performance of kernel Stein discrepancy. Three unbiased (left) and biased (right) Markov chains were simulated. In the unbiased case, the kernel Stein discrepancy $\mathcal{D}_{k_\pi}(\nu_n)$ correctly detects that the Markov chains are converging to the target. In the biased scenario, the kernel Stein discrepancy detects that the Markov chains have not converged to the correct target, but this becomes clear only after a sufficient number of iterations have been performed. [Stars indicate the initial state of each Markov chain. The density π is shaded.]

is a rare event. As a consequence, although kernel Stein discrepancy does provide convergence control, in unfavourable settings it may provide only a loose form of control.

In multi-dimensional settings, an appropriate choice of the matrix $\boldsymbol{\Sigma}$ appearing in the inverse multi-quadric reproducing kernel (6.27) can be

important. In situations where ν_n can be interpreted as an approximation of π, the present authors continue to recommended the use of $\mathbf{\Sigma} = \mathbf{I}_d$, following the application of a data-dependent transformation

$$(\mathbf{x}_i, \nabla \log \pi(\mathbf{x}_i)) \mapsto (\mathbf{\Gamma}_n^{-1}\mathbf{x}_i, \mathbf{\Gamma}_n \nabla \log \pi(\mathbf{x}_i)), \tag{6.28}$$

where $\mathbf{\Gamma}_n$ is the diagonal matrix whose diagonal entries are the mean absolute deviation of the corresponding coordinates of $(\mathbf{x}_k)_{1 \leq k \leq n}$, the states on which ν_n is supported. This transformation amounts to performing the change of variables $\mathbf{x} \mapsto \widetilde{\mathbf{x}} := \mathbf{\Gamma}_n^{-1}\mathbf{x}$ prior to computing the kernel Stein discrepancy with $\mathbf{\Sigma} = \mathbf{I}_d$. Indeed, denoting by $\widetilde{\pi}$ the transformed probability density function, the change-of-variables formula gives that

$$\begin{aligned}
\nabla_{\widetilde{\mathbf{x}}} \log \widetilde{\pi}(\widetilde{\mathbf{x}}) &= \nabla_{\widetilde{\mathbf{x}}} \log \left[\det(\mathbf{\Gamma}_n)\pi(\mathbf{x})\right] \\
&= \nabla_{\widetilde{\mathbf{x}}} \log \pi(\mathbf{x}) \\
&= \nabla_{\widetilde{\mathbf{x}}} \log \pi(\mathbf{\Gamma}_n\widetilde{\mathbf{x}}) = \mathbf{\Gamma}_n \nabla \log \pi(\mathbf{\Gamma}_n\widetilde{\mathbf{x}}) = \mathbf{\Gamma}_n \nabla \log \pi(\mathbf{x}).
\end{aligned}$$

In this recommendation, the mean absolute deviation is used to provide a robust estimate for the unknown scale of the standard deviation of each coordinate in π. One may be tempted to consider extending this recommendation to the more general class of invertible linear transformations, but the authors of Riabiz et al. (2022) cautioned that if considerable additional sample-based variability is introduced in estimating a general invertible linear transform, this can act as an undesirable confounding factor when the resulting discrepancies are to be interpreted for assessment of MCMC. In a similar spirit, so-called *sliced* kernel Stein discrepancies have recently been developed for high-dimensional applications (Gong et al., 2020); however, at the time of writing the convergence control of these discrepancies has yet to be established.

Aside from the specific limitations just discussed, there are a myriad of statistical applications where kernel Stein discrepancies can and have been successfully applied. Two distinct uses will be discussed in this chapter; optimal weighting of Markov chain output (Section 6.3), and optimal thinning of Markov chain output (Section 6.4). To close this section, we highlight that stronger modes of convergence can also be controlled by kernel Stein discrepancies. The following result, which is a special case of Kanagawa et al. (2024), indicates how a suitable *tilting* of the reproducing kernel enforces moment convergence control:

Theorem 6.7 (Moment convergence control; Corollary 3.4 in Kanagawa et al. (2024)) *Let π be distantly dissipative and $\nabla \log \pi$ be Lipschitz. Let $q \in \mathbb{N}$, $\mathbf{x}_0 \in \mathbb{R}^d$, and adopt the shorthand $w_r(\mathbf{x}) := (1 + \|\mathbf{x} - \mathbf{x}_0\|^2)^{(r-1)/2}$.*

Let

$$\mathsf{K}(\mathbf{x}, \mathbf{x}') = w_q(\mathbf{x}) w_q(\mathbf{x}') \mathsf{K}_{\mathrm{IMQ}}(\mathbf{x}, \mathbf{x}') + w_{q-1}(\mathbf{x}) w_{q-1}(\mathbf{x}') (1 + \langle \mathbf{x} - \mathbf{x}_0, \mathbf{x}' - \mathbf{x}_0 \rangle) \mathbf{I}_d$$

where $\mathsf{K}_{\mathrm{IMQ}}$ *is the inverse multi-quadric reproducing kernel from* (6.27)*. Let* $(\nu_n)_{n \in \mathbb{N}}$ *be a sequence of distributions whose moments up to order q exist. Then* $\mathcal{D}_{\mathsf{k}_\pi}(\nu_n) \to 0$ *implies that both* $\mathsf{d}_D(\pi, \nu_n) \to 0$ *and the moments of* ν_n *up to order q converge to those of* π*.*

In other words, the kernel Stein discrepancies constructed in this manner have control over the Wasserstein-q distance, which is equivalent to weak convergence plus the convergence of moments up to qth order. The principal requirement for making use of the kernel Stein discrepancies in Theorem 6.7 is to pick a location $\mathbf{x}_0 \in \mathbb{R}^d$. Theoretical guidance tells us that this kernel Stein discrepancy provides tightest control over moments when $\mathbf{x}_0$ is in a region of high probability for π, since otherwise the weightings w_q and w_{q-1} become approximately constant and we recover the standard kernel. The difficulty of finding such a location $\mathbf{x}_0$ will be context-dependent.

6.2.5 Stochastic Gradient Stein Discrepancy

The aim of this section is to discuss how kernel Stein discrepancies may be extended to the so-called *tall data* setting, where algorithms such as stochastic gradient MCMC from Chapter 3 are used. The principal challenge in this setting is that computation of the gradient $\nabla \log \pi$ is associated with a high computational cost $O(N)$ due to the form of the likelihood

$$L(\mathbf{x}; \mathcal{D}) = \prod_{j=1}^{N} L(\mathbf{x}; \mathbf{y}_j)$$

as a product of a large number N of terms that each need to be differentiated. Performing Bayesian inference with a prior $\pi_0(\mathbf{x})$, the posterior distribution takes the form

$$\pi(\mathbf{x}) \propto \prod_{j=1}^{N} \pi_j(\mathbf{x}), \qquad \pi_j(\mathbf{x}) \propto \pi_0(\mathbf{x})^{1/N} L(\mathbf{x}; \mathbf{y}_j),$$

where we assume each π_j can be properly normalised. Let $\nu_n = \frac{1}{n} \sum_{k=1}^{n} \delta_{\mathbf{x}_k}$ define a sequence $(\nu_n)_{n \in \mathbb{N}} \subset \mathcal{P}(\mathbb{R}^d)$ of discrete distributions in terms of a sequence $(\mathbf{x}_k)_{k \in \mathbb{N}} \subset \mathbb{R}^d$. Fix a *batch size* $m \ll N$ and, for each k, independently select a uniformly random subset $\mathcal{S}_m^{(k)}$ of size m from

$\{1, \dots, N\}$. Then

$$\widehat{\mathbf{b}}_k := \frac{N}{m} \sum_{j \in \mathcal{S}_m^{(k)}} \nabla \log \pi_j(\mathbf{x}_k)$$

is a stochastic approximation to the gradient $\mathbf{b}(\mathbf{x}_k) = \nabla \log \pi(\mathbf{x}_k)$ that can be computed at a relatively lower $O(m)$ cost. It is then tempting to replace the exact gradients $\mathbf{b}(\mathbf{x}_i)$ with their stochastic counterparts $\widehat{\mathbf{b}}_i$ within the construction of kernel Stein discrepancy. For reproducing kernels of the form $\mathsf{K}(\mathbf{x}, \mathbf{x}') = \mathsf{k}(\mathbf{x}, \mathbf{x}')\mathbf{I}_d$, this construction leads to the following stochastic approximation of (6.23):

$$\begin{aligned}\widehat{\mathsf{k}}_\pi(\mathbf{x}_i, \mathbf{x}_j) := \nabla_{\mathbf{x}} \cdot \nabla_{\mathbf{x}'} \mathsf{k}(\mathbf{x}, \mathbf{x}')|_{\mathbf{x}=\mathbf{x}_i, \mathbf{x}'=\mathbf{x}_j} + \left\langle \widehat{\mathbf{b}}_i, \nabla_{\mathbf{x}'} \mathsf{k}(\mathbf{x}_i, \mathbf{x}')|_{\mathbf{x}'=\mathbf{x}_j} \right\rangle \\ + \left\langle \widehat{\mathbf{b}}_j, \nabla_{\mathbf{x}} \mathsf{k}(\mathbf{x}, \mathbf{x}_j)\big|_{\mathbf{x}=\mathbf{x}_i} \right\rangle + \left\langle \widehat{\mathbf{b}}_i, \widehat{\mathbf{b}}_j \right\rangle \mathsf{k}(\mathbf{x}_i, \mathbf{x}_j)\end{aligned}$$

Gorham et al. (2020) defined the *stochastic kernel Stein discrepancy* in this context as

$$\mathcal{D}_{\hat{\mathsf{k}}_\pi}(\nu_n) = \sqrt{\frac{1}{n^2} \sum_{k=1}^{n} \sum_{k'=1}^{n} \hat{\mathsf{k}}_\pi(\mathbf{x}_k, \mathbf{x}_{k'})}.$$

An immediate question is whether or not the introduction of stochasticity into the gradients jeopardised the weak convergence control property established in the case of exact gradients in Theorem 6.6. It turns out that, provided each $\pi_1, \dots, \pi_N$ satisfies the conditions of Theorem 6.6, a form of weak convergence control continues to hold. Specifically, if each π_i is distantly dissipative, and each $\nabla \log \pi_i$ is Lipschitz, then with $\mathsf{k}(\mathbf{x}, \mathbf{x}') = (1 + \|(\mathbf{x} - \mathbf{x}')/\sigma\|^2)^{-\beta}$, $\sigma > 0$, $\beta \in (0, 1)$, Gorham et al. (2020, Theorem 4) shows that

$$\mathcal{D}_{\hat{\mathsf{k}}_\pi}(\nu_n) \to 0 \qquad \Longrightarrow \qquad \mathsf{d}_{\mathsf{D}}(\pi, \nu_n) \to 0$$

almost surely. This result justifies the use of stochastic kernel Stein discrepancies in their own right, not merely as approximations to kernel Stein discrepancy that becomes exact when $m = N$. Indeed, in principle, only a batch size of $m = 1$ is required.

Figure 6.4 displays stochastic kernel Stein discrepancies computed for the sequence of empirical distributions ν_n produced using stochastic gradient Langevin dynamics applied to the logistic regression example from Section 3.5.1. It can be seen that, even for a batch size $m = 100$, which is much less than the size $N = 10^4$ of the dataset, the stochastic kernel Stein discrepancy is capable of providing similar information on the performance

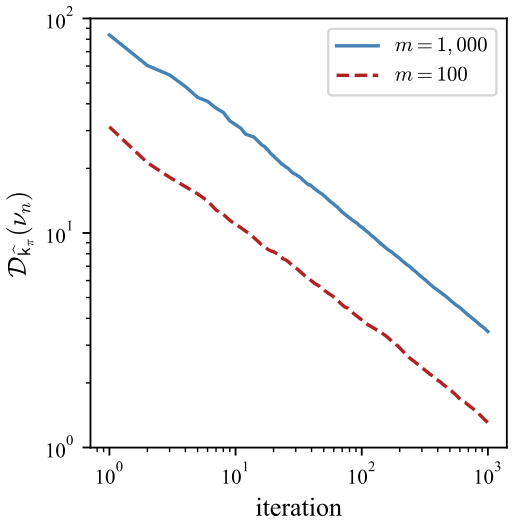

Figure 6.4 Stochastic gradient Stein discrepancies. Stochastic gradient Langevin dynamics was used to generate approximate samples from the posterior distribution π in the logistic regression example from Section 3.5.1, and stochastic gradient Stein discrepancies were used to measure the discrepancy between the empirical distribution ν_n of the approximate samples and π. Here m indicates the size of the data subsets that were used to approximate the gradient.

of the sampler compared to when larger batch sizes, such as $m = 10^3$ are used. The predictable decrease of the discrepancy indicates that the intrinsic bias of stochastic gradient Langevin dynamics is negligible relative to the error incurred by using only n of these samples to construct ν_n. Nevertheless, at the time of writing, there remains scope to improve these stochastic discrepancies, not least through the use of reduced-variance stochastic approximations to the gradient.

6.3 Optimal Weights for MCMC

At this point we have seen how computable discrepancies may be constructed and used to *passively* assess the performance of MCMC. Now we turn to how such discrepancies might be used to *actively* improve output

from MCMC. Specifically, in this section we explore how, given a realisation $(\mathbf{x}_k)_{1\leq k\leq n}$ of a Markov chain and a target distribution π, we may exploit the kernel Stein discrepancy to assign a weight w_k to each $\mathbf{x}_k$ in such a manner that the discrepancy between the weighted empirical distribution and π is minimised. This is loosely analogous to importance sampling (c.f. Section 1.1.5), except here the analogue of the importance distribution is the distribution of the MCMC sample path which, like the posterior itself, is implicitly defined. As such, the methods we will discuss were termed *Black Box Importance Sampling* in Liu and Lee (2017). Surprisingly, we will see that such retrospective re-weighting can be used to remove the bias of approximate sampling algorithms, such as stochastic gradient MCMC from Chapter 3.

Let $\mathsf{k}_\pi : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ be a scalar-valued reproducing kernel for which $\int \mathsf{k}_\pi(\mathbf{x}, \mathbf{y})\, \mathrm{d}\pi(\mathbf{x}) = 0$ for all $\mathbf{y} \in \mathbb{R}^d$; an example being (6.23). The weights that we consider are the solution of the following optimisation problem:

$$\mathbf{w}^\star := \begin{pmatrix} w_1^\star \\ \vdots \\ w_n^\star \end{pmatrix} \in \operatorname*{arg\,min}_{\substack{w_1,\ldots,w_n \geq 0 \\ w_1+\cdots+w_n=1}} \mathcal{D}_{\mathsf{k}_\pi}\left(\sum_{k=1}^n w_k \delta_{\mathbf{x}_k}\right) \tag{6.29}$$

Let ν_n be the general weighted empirical distribution appearing on the right-hand side of (6.29). From (6.22) we have $\mathcal{D}_{\mathsf{k}_\pi}(\nu_n)^2 = \langle \mathbf{w}, \mathsf{K}_\pi \mathbf{w} \rangle$, where K_π is the $n \times n$ matrix with entries $[\mathsf{K}_\pi]_{i,j} = \mathsf{k}_\pi(\mathbf{x}_i, \mathbf{x}_j)$. If the matrix K_π is positive definite then $\mathbf{w}^\star$ is unique and, although not available in closed form, $\mathbf{w}^\star$ can be computed by solving a linearly-constrained quadratic optimisation problem over the positive orthant of $\mathbb{R}^n$. The optimally weighted distribution will be denoted $\nu_n^\star$ in the sequel.

A natural first question is whether the weighted approximation to π, obtained by retrospectively assigning weights to output from MCMC, is consistent. This turns out to be true, under appropriate assumptions, and moreover, optimal weights can provide bias correction in settings where the Markov chain is not π-invariant. Indeed, the recent work of Riabiz et al. (2022) established that, in the case of a μ-invariant, time-homogeneous, ergodic Markov chain $(\mathbf{X}_k)_{k\in\mathbb{N}} \subset \mathbb{R}^d$, then the existence of certain moments of the ratio π/μ can be used to deduce that

$$\mathcal{D}_{\mathsf{k}_\pi}\left(\sum_{k=1}^n w_k^\star \delta_{\mathbf{X}_k}\right) \to 0$$

almost surely as $n \to \infty$. Importantly, the biased target μ of the Markov

chain $(\mathbf{X}_k)_{k\in\mathbb{N}}$ does not need to be known to perform Black Box Importance Sampling in (6.29).

It can sometimes be convenient to relax the non-negativity constraint, to consider

$$\widetilde{\mathbf{w}}^\star := \begin{pmatrix} \widetilde{w}_1^\star \\ \vdots \\ \widetilde{w}_n^\star \end{pmatrix} \in \underset{\substack{\mathbf{w}\in\mathbb{R}^n \\ w_1+\cdots+w_n=1}}{\arg\min}\ \mathcal{D}_{\mathsf{k}_\pi}\left(\sum_{k=1}^n w_k \delta_{\mathbf{x}_k}\right). \tag{6.30}$$

The weights $\widetilde{\mathbf{w}}^\star$ may be negative, and thus the associated $\tilde{\nu}_n^\star$ is a *signed measure* in general. Signed measures may not pose a problem if the goal of computation is to approximate posterior expectations of interest, but if the goal is to approximate π itself then a proper probability distribution may be preferred. The main advantage of the formulation in (6.30) is that, provided $\mathsf{K}_\pi \succ 0$, the relaxed optimisation problem has a unique and explicit solution

$$\widetilde{\mathbf{w}}^\star = \frac{\mathsf{K}_\pi^{-1}\mathbf{1}}{\mathbf{1}^\top \mathsf{K}_\pi^{-1}\mathbf{1}}. \tag{6.31}$$

This formulation is closely related to *kernel cubature* (also known as *Bayesian quadrature*), and specifically coincides with the *normalised* kernel cubature of Karvonen et al. (2018). From (6.31), we deduce that the computational complexity of obtaining optimal weights is in general $O(n^3)$. This can preclude the use of optimal weights on desktop computational hardware when n is larger than a few thousand. However, we will see in Section 6.4 how accurate sparse approximations to the optimally weighted distribution can be efficiently constructed.

To demonstrate the effect of re-weighting, consider the following *Rosenbrock* target

$$\pi(x, y) \propto \exp(-(x-a)^2 - b(y-x^2)^2)$$

where here we take $a = 0$, $b = 3$. The distribution π might be referred to as a *horseshoe* distribution, due to the curved shape of its level sets. To represent output from a biased sampler, we generate sequence $(\mathbf{X}_k)_{k\in\mathbb{N}}$ of independent samples from $\nu = \mathsf{N}(\mathbf{0}, \mathbf{I}_2)$ and assign a weight w_i to each state $\mathbf{X}_i$ according to either (6.29) or (6.30). For these experiments we took $\boldsymbol{\Sigma} = \mathbf{I}_2$, $\beta = 0.5$, and the transformation in (6.28) was applied. Figure 6.5 displays the qualitative properties of the weights $\mathbf{w}^\star$ defined by (6.29) (left) and the weights $\widetilde{\mathbf{w}}^\star$ defined by (6.30) (right). In both cases, it can be seen that states $\mathbf{X}_k$ for which the probability under π is low are typically assigned a small weight. On the other hand, optimal weights are not independent, and the over-representation of states in a local region due to Monte Carlo

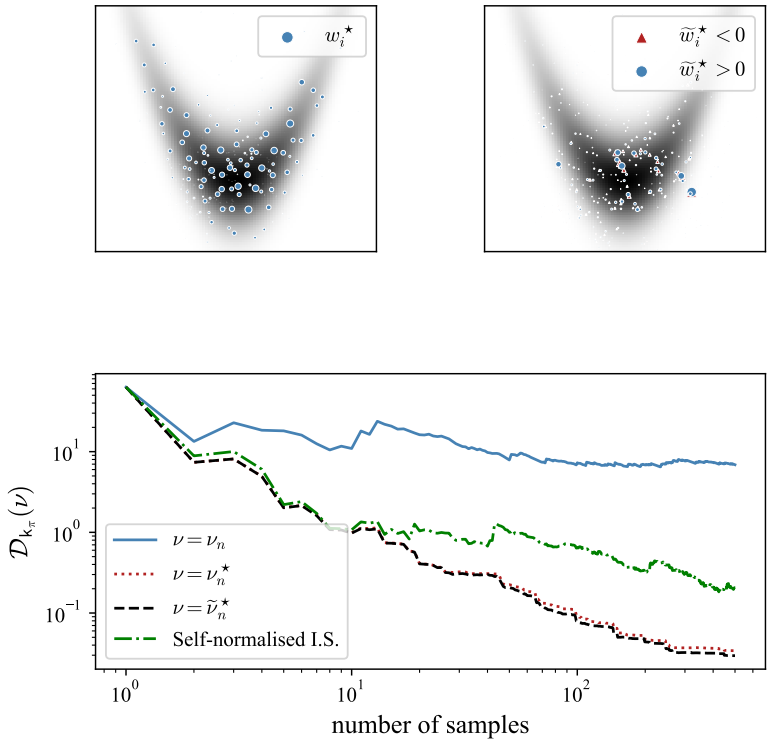

Figure 6.5 Optimal weights for MCMC. Samples from a standard Gaussian distribution were assigned either the optimal sign-constrained weights $\mathbf{w}^\star$ (left) or the optimal unconstrained weights $\widetilde{\mathbf{w}}^\star$ (right), to obtain consistent approximations $\nu_n^\star$ and $\widetilde{\nu}_n^\star$ respectively of the horseshoe distribution π (shaded). These approximations each demonstrate convergence in the sense of kernel Stein discrepancy as the number of samples is increased, while the distribution ν of the original samples does not provide a consistent approximation of π.

sampling variability is partially mitigated. Comparison of the kernel Stein discrepancies $\mathcal{D}_{\mathsf{k}_\pi}(\nu_n^\star)$ and $D_{\mathsf{k}_\pi}(\widetilde{\nu}_n^\star)$ indicates that the non-negative weights $\mathbf{w}^\star$ perform nearly as well as the signed weights $\widetilde{\mathbf{w}}^\star$, while both sets of weights lead to a substantial decrease in kernel Stein discrepancy compared to the use of (inconsistent) uniform weights in this experiment.

Of course, in this toy example, one has access to the sampling density of the $\mathbf{X}_k$ and self-normalised importance sampling could trivially be used. That is, to each sample $\mathbf{X}_k$ we assign weights proportional to $\pi(\mathbf{X}_k)/\nu(\mathbf{X}_k)$, and we normalise these weights to sum to 1. The performance of self-normalised importance sampling is displayed in the lower panel of Figure 6.5, where it is seen to be inferior to the discrepancy-based methods which we have discussed. Where has this performance gap come from? Well, in addition to bias correction, the discrepancy-based methods additionally perform variance reduction, in the sense that the random vectors $\mathbf{w}^\star$ and $\widetilde{\mathbf{w}}^\star$ each contain components that are strongly inter-dependent. This means that if a region is over-represented with samples, then the overall weight of these samples can be collectively reduced to better approximate the target π. In contrast, self-normalised importance sampling has to rely on the long-run frequency of independent sampling to ensure that different regions of the domain are assigned an equal amount of probability mass, and this negatively affects its finite sample performance.

The satisfactory performance of optimal weights is observed in settings such as Figure 6.5, where pathological behaviours of kernel Stein discrepancy (such as blindness to mixing proportions; see Section 6.2.4) are not encountered. However, outside this setting the use of optimal weights can fail. Further, if the kernel Stein discrepancy has weak convergence control but not moment control, then there is no guarantee that moments computed using the weighted empirical approximations $\nu_n^\star$ or $\widetilde{\nu}_n^\star$ will be convergent in the $n \to \infty$ limit. These remarks emphasise that a certain degree of caution is needed when optimal weights are employed.

6.4 Optimal Thinning for MCMC

The output from a sampling algorithm is often used for subsequent computation, for example, to approximate the posterior expectation of a quantity of interest. In scenarios where this subsequent computation incurs a non-trivial computational cost, it is usually desirable to work with as small a number n of samples as possible, provided that these continue to provide an accurate approximation to the posterior target. Standard practice for MCMC is to retain the subset of states visited along the sample path whose indices are $(\sigma(i))_{1\leq i\leq m}$, where $\sigma(i) = b + ci$, b is the duration of a burn-in period and c is the *thinning* period. However, this does not directly attempt to arrive at a compressed representation of the posterior target. The aim of this section is to discuss how one might select indices $(\sigma(i))_{1\leq i\leq m}$ to optimally approximate the target. In doing so, we will also arrive at a convenient sparse

approximation to the optimally weighted distributions studied in Section 6.3.

Given output $(\mathbf{X}_k)_{1 \leq k \leq n}$ from a π-invariant MCMC algorithm, we aim to construct an approximation $\nu_{n,m} = \frac{1}{m} \sum_{k=1}^{m} \delta_{\mathbf{X}_{\sigma(k)}}$ to π, which we require is *sparse*, meaning that $m \ll n$. For concreteness, we consider the setting where the index sequence σ is greedily determined according to

$$\sigma(j) \in \operatorname*{arg\,min}_{k=1,\ldots,n} \mathcal{D}_{\mathsf{k}_\pi} \left(\frac{1}{j} \delta_{\mathbf{X}_k} + \frac{1}{j} \sum_{j'=1}^{j-1} \delta_{\mathbf{X}_{\sigma(j')}} \right)$$

for each $j \in \mathbb{N}$. Using (6.22), and ignoring terms that do not depend on $\mathbf{X}_k$, the greedy algorithm is equivalent to

$$\sigma(j) \in \operatorname*{arg\,min}_{k=1,\ldots,n} \frac{\mathsf{k}_\pi(\mathbf{X}_k, \mathbf{X}_k)}{2} + \sum_{j'=1}^{j-1} \mathsf{k}_\pi(\mathbf{X}_k, \mathbf{X}_{\pi(j')})$$

where, in the event of a tie, it does not matter how a state is selected. The approximation $\nu_{n,m}$, under appropriate regularity conditions, converges to $\nu_n^\star$ in the $m \to \infty$ limit; see Theorem 1 of Riabiz et al. (2022). Thus, a finite run of this greedy algorithm could in principle be used as a sparse alternative to optimal weighting of states from Section 6.3. Further, the computational complexity of this greedy algorithm is $O(nm^2)$, which would improve on the $O(n^3)$ of the optimal weights from Section 6.3 when $m \ll n$. But how large should m be for $\nu_{n,m}$ to be a sufficiently accurate approximation of $\nu_n^\star$ to be useful? This question was answered with a theoretical argument in Riabiz et al. (2022), who established that

$$\mathcal{D}_{\mathsf{k}_\pi}(\nu_{n,m}) - \mathcal{D}_{\mathsf{k}_\pi}(\nu_n^\star) \to 0$$

almost surely as $m, n \to \infty$, under conditions that include requiring m to increase at least as fast as $(\log n)^{2/\beta}$ for some $\beta \in (0, 1)$. This is a relatively mild constraint on m, and in this sense the relative size of m compared to n can be small.

To perform an empirical comparison of optimal weighting and the greedy algorithm just described, we return to the Rosenbrock example from Section 6.3. Using the same MCMC output, we contrast the approximations produced using the weights $\mathbf{w}^\star$ with the approximation produced using the greedy algorithm just described. Figure 6.6 demonstrates the convergence of the sparse approximation to the optimally weighted approximation as m is increased. This convergence occurs reasonably quickly, suggesting that a faster, sparse approximation may often be preferred compared to the weighted approximations from Section 6.3. For these experiments, we took

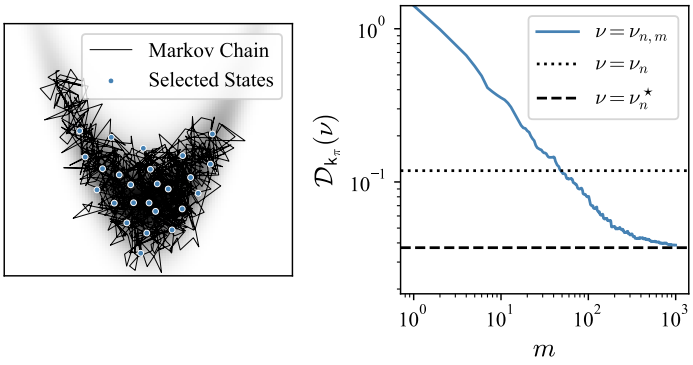

Figure 6.6 Optimal thinning for MCMC. A subset of size m was selected from the sample path $(\mathbf{X}_k)_{1 \leq k \leq n}$ of a π-invariant Markov chain in such a way that the kernel Stein discrepancy between the associated empirical distribution (circles) and π (shaded) was greedily minimised. The selected states are shown on the left panel, while on the right panel the kernel Stein discrepancy of the resulting empirical distribution $\nu_{n,m}$ is seen to converge to that of the optimally weighted empirical distribution $\nu_n^\star$ that uses the full Markov chain output. Here $n = 10^3$.

$\boldsymbol{\Sigma} = \mathbf{I}_2$, $\beta = 0.5$, and the transformation in (6.28) was applied. Although in this toy example, sampling was not a computational bottleneck, in more challenging examples the use of these greedy algorithms can provide an automatic method to both identify and remove an initial burn-in period, and to compress the sampler output.

6.5 Chapter Notes

The development of sophisticated sampling algorithms, including those described in this book, should be guided by a qualitative assessment of their empirical performance over a variety of realistic distributional targets. The purpose of this Chapter was to demonstrate how one can construct explicit upper bounds on the "closeness" of the sampler output to the target. In particular, kernel Stein discrepancies emerged as a computationally convenient performance measure, which can be computed provided that the gradient of the target log-density can be evaluated pointwise. Except for scenarios where the posterior contains distant high-probability regions,

the kernel Stein discrepancy can provide an accurate indication of sampler performance. Further, we described two different scenarios in which output from MCMC can be actively improved using these techniques; optimal weighting and optimal thinning of MCMC output.

The literature on diagnostic checks for MCMC is almost as old as the literature on MCMC. Our brief discussion in Section 6.1 barely scratched the surface of this topic, and we refer the interested reader to more detailed treatments such as Cowles and Carlin (1996). The convergence diagnostics we presented are due to Gelman and Rubin (1992); Brooks and Gelman (1998); Gelman et al. (2014). The somewhat arbitrary choices of $\delta = 0.1$ and $\delta = 0.01$ were used, respectively, in Gelman et al. (2014); Vats and Knudson (2021) and Vehtari et al. (2021). Generalisations of these convergence diagnostics to the case of a multivariate target, and other improvements, can be found in e.g. Brooks and Gelman (1998); Vats and Knudson (2021); Vehtari et al. (2021).

The construction of computable convergence bounds has received limited historical attention, from authors that include Meyn et al. (1994); Rosenthal (1995); Roberts and Tweedie (1999); Jones and Hobert (2001). The convergence bound we presented in Section 6.2 was somewhat novel, in the sense that earlier work has tended to motivate and derive such bounds as a consequence of *Stein's method*. Introduced in Stein (1972), this technique from applied probability has been extensively used to study various instances of approximation among random variables. However, the last decade has seen an explosion of research investigating the *computational* uses of Stein's method, sparked by the formalisation of the Stein discrepancy in Gorham and Mackey (2015). A myriad of computational applications of Stein discrepancies have now been explored, and a recent overview is provided in Anastasiou et al. (2023).

The use of reproducing kernels led us to a discrepancy that could be explicitly computed. However, there are some technical challenges associated with the use of the resulting kernel Stein discrepancies. First, the computational complexity of evaluating the kernel Stein discrepancy between π and a distribution ν_n supported on n discrete states is $O(n^2)$; c.f. (6.24). However, this complexity can in fact be reduced to near-linear using the *random features* approach developed in Huggins and Mackey (2018), whose discussion was beyond the scope of this book. Second, one must ensure that the required properties of the discrepancy hold in the relevant applied context. Our discussion focused on weak convergence control, but other relevant properties include *separation*, meaning that $D_\pi(\nu) = 0$ if and only if $\pi = \nu$, and *convergence detection*, meaning that $D_\pi(\nu_n) \to 0$

whenever ν_n converges to π in an appropriate sense to be specified. A discrepancy for which both convergence control and convergence detection are satisfied may be used to compare and select between competing sampling algorithms, as investigated in Gorham and Mackey (2015, 2017). To this end, a rigorous technical presentation of kernel Stein discrepancies and their theoretical properties can be found in Barp et al. (2022).

The use of isotropic reproducing kernels can lead to a *curse of dimension*, meaning that differences between probability distributions become more difficult to detect as the dimension d of the state space is increased. A generalisation that replaces the overdamped Langevin in (6.13) with a more general class of π-invariant diffusion processes on $\mathbb{R}^d$ was studied in Gorham et al. (2019), and in the case of kernel Stein discrepancy, this is equivalent to the use of certain non-isotropic reproducing kernels, however, the selection of a suitable diffusion to address the curse of dimension has not been explored. In a constructive attempt to improve the performance of Stein discrepancy in the high-dimensional context, Grathwohl et al. (2020) proposed to substitute the reproducing kernel Hilbert space in the integral probability metric (6.8) with the set of test functions spanned by an appropriately differentiable parametric neural network. Such an approach trades the potentially better detection properties of the discrepancy with both a lack of theoretical guarantees and the additional computational complexity involved in the adversarial training of a neural network. Further research will be required to understand this trade-off in detail.

To limit the scope, we discussed only algorithms for optimal weighting and optimal thinning, in each case for probability distributions defined on $\mathbb{R}^d$. Optimal weighting was introduced in Liu and Lee (2017) and its consistency was first established in Hodgkinson et al. (2020). Optimal thinning was introduced and analysed in Riabiz et al. (2022), and mini-batching strategies were proposed and studied to further reduce the $O(nm^2)$ cost in Teymur et al. (2021). Both algorithms can be generalised to non-Euclidean domains $\mathcal{X}$ through the identification of a suitable Markov process on $\mathcal{X}$ with an explicit generator $\mathcal{L}_\pi$; some Markov processes suitable for discrete domains $\mathcal{X}$ are described in e.g. Shi et al. (2022). In related work, Fisher and Oates (2024) demonstrated how consistent approximation using optimal weights and optimal thinning can even be achieved *without* access to gradients of the target, provided that gradients of a suitable approximating distribution can be obtained.

References

Ahn, Sungjin, Korattikara, Anoop, Liu, Nathan, Rajan, Suju, and Welling, Max. 2015. Large-scale distributed Bayesian matrix factorization using stochastic gradient MCMC. Pages 9–18 of: *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*. ACM.

Aicher, Christopher, Ma, Yi-An, Foti, Nicholas J, and Fox, Emily B. 2019. Stochastic gradient MCMC for state space models. *SIAM Journal on Mathematics of Data Science*, **1**(3), 555–587.

Aicher, Christopher, Putcha, Srshti, Nemeth, Christopher, Fearnhead, Paul, and Fox, Emily. 2023. Stochastic gradient MCMC for nonlinear state space models. *Bayesian Analysis*, **1**(1), 1–23.

Anastasiou, Andreas, Barp, Alessandro, Briol, François-Xavier, Ebner, Bruno, Gaunt, Robert E, Ghaderinezhad, Fatemeh, Gorham, Jackson, Gretton, Arthur, Ley, Christophe, Liu, Qiang, Mackey, Lester, Oates, Chris J., Reinert, Gesine, and Swan, Yvik. 2023. Stein's method meets computational statistics: A review of some recent developments. *Statistical Science*, **38**(1), 120–139.

Andrieu, Christophe, Durmus, Alain, Nüsken, Nikolas, and Roussel, Julien. 2021. Hypocoercivity of piecewise deterministic Markov process-Monte Carlo. *The Annals of Applied Probability*, **31**(5), 2478–2517.

Baker, Jack, Fearnhead, Paul, Fox, Emily, and Nemeth, Christopher. 2018. Large-Scale Stochastic Sampling from the Probability Simplex. Pages 6721–6731 of: *Advances in Neural Information Processing Systems*.

Baker, Jack, Fearnhead, Paul, Fox, Emily B, and Nemeth, Christopher. 2019. Control variates for stochastic gradient MCMC. *Statistics and Computing*, **29**(3), 599–615.

Bardenet, Rémi, Doucet, Arnaud, and Holmes, Chris. 2014. Towards scaling up Markov chain Monte Carlo: an adaptive subsampling approach. Pages 405–413 of: *International Conference on Machine Learning (ICML)*.

Barp, Alessandro, Simon-Gabriel, Carl-Johann, Girolami, Mark, and Mackey, Lester. 2022. Targeted separation and convergence with kernel discrepancies. In: *NeurIPS 2022 Workshop on Score-Based Methods*.

Beck, Amir, and Teboulle, Marc. 2003. Mirror descent and nonlinear projected subgradient methods for convex optimization. *Operations Research Letters*, **31**(3), 167–175.

Bernardo, José M, and Smith, Adrian FM. 2009. *Bayesian Theory*. John Wiley & Sons.

Besag, Julian. 1994. Comments on "Representations of knowledge in complex systems" by U. Grenander and M.I. Miller. *Journal of the Royal Statistical Society Series B*, **56**, 591–592.

Beskos, Alex, Roberts, Gareth, and Stuart, Andrew. 2009. Optimal scalings for local Metropolis-Hastings chains on non-product targets in high dimensions. *Annals of Applied Probability*, **19**(3), 863–898.

Beskos, Alexandros, Pillai, Natesh, Roberts, Gareth, Sanz-Serna, Jesus-Maria, and Stuart, Andrew. 2013. Optimal tuning of the hybrid Monte Carlo algorithm. *Bernoulli*, **19**(5A), 1501–1534.

Bierkens, Joris. 2016. Non-reversible Metropolis-Hastings. *Statistics and Computing*, **26**(6), 1213–1228.

Bierkens, Joris, and Duncan, Andrew. 2017. Limit theorems for the zig-zag process. *Advances in Applied Probability*, **49**(3), 791–825.

Bierkens, Joris, and Roberts, Gareth. 2017. A piecewise deterministic scaling limit of lifted Metropolis–Hastings in the Curie–Weiss model. *The Annals of Applied Probability*, **27**, 846–882.

Bierkens, Joris, and Verduyn Lunel, Sjoerd M. 2022. Spectral analysis of the zigzag process. Pages 827–860 of: *Annales de l'Institut Henri Poincare (B) Probabilites et statistiques*, vol. 58. Institut Henri Poincaré.

Bierkens, Joris, Bouchard-Côté, Alexandre, Doucet, Arnaud, Duncan, Andrew B, Fearnhead, Paul, Lienart, Thibaut, Roberts, Gareth, and Vollmer, Sebastian J. 2018. Piecewise deterministic Markov processes for scalable Monte Carlo on restricted domains. *Statistics & Probability Letters*, **136**, 148–154.

Bierkens, Joris, Roberts, Gareth O, and Zitt, Pierre-André. 2019a. Ergodicity of the zigzag process. *The Annals of Applied Probability*, **29**(4), 2266–2301.

Bierkens, Joris, Fearnhead, Paul, and Roberts, Gareth O. 2019b. The Zig-Zag process and super-efficient sampling for Bayesian analysis of big data. *Annals of statistics*, **47**(3), 1288–1320.

Bierkens, Joris, Grazzi, Sebastiano, Kamatani, Kengo, and Roberts, Gareth. 2020. The boomerang sampler. Pages 908–918 of: *International Conference on Machine Learning*. PMLR.

Bierkens, Joris, Kamatani, Kengo, and Roberts, Gareth O. 2022. High-dimensional scaling limits of piecewise deterministic sampling algorithms. *The Annals of Applied Probability*, **32**(5), 3361–3407.

Bierkens, Joris, Kamatani, Kengo, and Roberts, Gareth O. 2023a. *Scaling of Piecewise Deterministic Monte Carlo for Anisotropic Targets*.

Bierkens, Joris, Grazzi, Sebastiano, Meulen, Frank van der, and Schauer, Moritz. 2023b. Sticky PDMP samplers for sparse and local inference problems. *Statistics and Computing*, **33**(1), 8.

Blei, David M, Ng, Andrew Y, and Jordan, Michael I. 2003. Latent dirichlet allocation. *Journal of machine Learning research*, **3**(Jan), 993–1022.

Bou-Rabee, Nawaf, and Sanz-Serna, Jesús María. 2017. RANDOMIZED HAMILTONIAN MONTE CARLO. *The Annals of Applied Probability*, **27**(4), 2159–2194.

Bouchard-Côté, Alexandre, Vollmer, Sebastian J, and Doucet, Arnaud. 2018. The bouncy particle sampler: A nonreversible rejection-free Markov chain Monte Carlo method. *Journal of the American Statistical Association*, **113**(522), 855–867.

Brooks, Stephen P, and Gelman, Andrew. 1998. General methods for monitoring convergence of iterative simulations. *Journal of computational and graphical statistics*, **7**(4), 434–455.

Brooks, Steve, Gelman, Andrew, Jones, Galin, and Meng, Xiao-Li. 2011. *Handbook of Markov chain Monte Carlo*. CRC press.

Brosse, Nicolas, Durmus, Alain, Moulines, Éric, and Pereyra, Marcelo. 2017. Sampling from a log-concave distribution with compact support with proximal Langevin Monte Carlo. Pages 319–342 of: *Conference on learning theory*. PMLR.

Brosse, Nicolas, Durmus, Alain, and Moulines, Éric. 2018. The promises and pitfalls of Stochastic Gradient Langevin Dynamics. Pages 8278–8288 of: *Advances in Neural Information Processing Systems*.

Bubeck, Sébastien, Eldan, Ronen, and Lehec, Joseph. 2018. Sampling from a log-concave distribution with Projected Langevin Monte Carlo. *Discrete & Computational Geometry*, **59**(4), 757–783.

Cabezas, Alberto, Corenflos, Adrien, Lao, Junpeng, and Louf, Rémi. 2024. BlackJAX: Composable Bayesian inference in JAX. *arXiv preprint arXiv:2402.10797*.

Caflisch, Russel E. 1998. Monte Carlo and Quasi-Monte Carlo Methods. *Acta Numerica*, **7**, 1–49.

Carmeli, Claudio, De Vito, Ernesto, and Toigo, Alessandro. 2006. Vector valued reproducing kernel Hilbert spaces of integrable functions and Mercer theorem. *Analysis and Applications*, **4**(04), 377–408.

Chatterji, Niladri, Flammarion, Nicolas, Ma, Yian, Bartlett, Peter, and Jordan, Michael. 2018. On the theory of variance reduction for stochastic gradient Monte Carlo. Pages 764–773 of: *International Conference on Machine Learning*. PMLR.

Chen, Fang, Lovász, László, and Pak, Igor. 1999. Lifting Markov chains to speed up mixing. Pages 275–281 of: *Proceedings of the thirty-first annual ACM symposium on Theory of computing*.

Chen, Tianqi, Fox, Emily, and Guestrin, Carlos. 2014. Stochastic gradient Hamiltonian Monte Carlo. Pages 1683–1691 of: *International Conference on Machine Learning*.

Chen, Wilson Ye, Barp, Alessandro, Briol, François-Xavier, Gorham, Jackson, Girolami, Mark, Mackey, Lester, and Oates, Chris. 2019. Stein point Markov chain Monte Carlo. Pages 1011–1021 of: *International Conference on Machine Learning*. PMLR.

Chevallier, Augustin, Power, Sam, Wang, Andi Q, and Fearnhead, Paul. 2021. *PDMP Monte Carlo methods for piecewise-smooth densities*. arXiv:2111.05859.

Chevallier, Augustin, Fearnhead, Paul, and Sutton, Matthew. 2023. Reversible jump PDMP samplers for variable selection. *Journal of the American Statistical Association*, **118**(544), 2915–2927.

Christensen, Ole F., Roberts, Gareth O., and Rosenthal, Jeffrey S. 2005. Scaling Limits for the Transient Phase of Local Metropolis-Hastings Algorithms. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, **67**(2), 253–268.

Chwialkowski, Kacper, Strathmann, Heiko, and Gretton, Arthur. 2016. A kernel test of goodness of fit. Pages 2606–2615 of: *International conference on machine learning*. PMLR.

Conway, John B. 2010. *A Course in Functional Analysis*. Second edn. Springer.

Corbella, Alice, Spencer, Simon EF, and Roberts, Gareth O. 2022. Automatic Zig-Zag sampling in practice. *Statistics and Computing*, **32**(6), 107.

Coullon, Jeremie, and Nemeth, Christopher. 2022. SGMCMCJax: a lightweight JAX library for stochastic gradient Markov chain Monte Carlo algorithms. *Journal of Open Source Software*, **7**(72), 4113.

Coullon, Jeremie, South, Leah, and Nemeth, Christopher. 2023. Efficient and generalizable tuning strategies for stochastic gradient MCMC. *Statistics and Computing*, **33**(3), 66.

Cowles, Mary Kathryn, and Carlin, Bradley P. 1996. Markov chain Monte Carlo convergence diagnostics: a comparative review. *Journal of the American Statistical Association*, **91**(434), 883–904.

Cox, John, Ingersoll Jr, Jonathan E, and Ross, Stephen A. 1985. A Theory of the Term Structure of Interest Rates. *Econometrica*, **53**(2), 385–408.

Creutz, Michael. 1988. Global Monte Carlo algorithms for many-fermion systems. *Phys. Rev. D*, **38**(Aug), 1228–1238.

Dalalyan, Arnak S, and Karagulyan, Avetik. 2019. User-friendly guarantees for the Langevin Monte Carlo with inaccurate gradient. *Stochastic Processes and their Applications*, **129**(12), 5278–5311.

Davis, Mark H A. 1984. Piecewise-deterministic Markov processes: A general class of non-diffusion stochastic models. *Journal of the Royal Statistical Society: Series B (Methodological)*, **46**(3), 353–376.

Deligiannidis, George, Bouchard-Côté, Alexandre, and Doucet, Arnaud. 2019. Exponential ergodicity of the bouncy particle sampler. *The Annals of Statistics*, **47**, 1268–1287.

Deligiannidis, George, Paulin, Daniel, Bouchard-Côté, Alexandre, and Doucet, Arnaud. 2021. Randomized Hamiltonian Monte Carlo as scaling limit of the bouncy particle sampler and dimension-free convergence rates. *The Annals of Applied Probability*, **31**(6), 2612–2662.

Diaconis, Persi, Holmes, Susan, and Neal, Radford M. 2000. Analysis of a nonreversible Markov chain sampler. *Annals of Applied Probability*, **10**(3), 726–752.

Doucet, Arnaud, Johansen, Adam M, et al. 2009. A tutorial on particle filtering and smoothing: Fifteen years later. *Handbook of nonlinear filtering*, **12**(656-704), 3.

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

Dubey, Kumar Avinava, Reddi, Sashank J, Williamson, Sinead A, Poczos, Barnabas, Smola, Alexander J, and Xing, Eric P. 2016. Variance reduction in stochastic gradient Langevin dynamics. Pages 1154–1162 of: *Advances in Neural Information Processing Systems*.

Eberle, Andreas. 2016. Reflection couplings and contraction rates for diffusions. *Probability theory and related fields*, **166**(3), 851–886.

Fearnhead, Paul, Bierkens, Joris, Pollock, Murray, Roberts, Gareth O, et al. 2018. Piecewise deterministic Markov processes for continuous-time Monte Carlo. *Statistical Science*, **33**(3), 386–412.

Fisher, Matthew, and Oates, Chris J. 2024. Gradient-free kernel Stein discrepancy. *Advances in Neural Information Processing Systems*, **36**.

Gamerman, Dani, and Lopes, Hedibert F. 2006. *Markov chain Monte Carlo: stochastic simulation for Bayesian inference*. CRC press.

Gelman, Andrew, and Rubin, Donald B. 1992. Inference from iterative simulation using multiple sequences. *Statistical science*, **7**(4), 457–472.

Gelman, Andrew, Carlin, John B, Stern, Hal S, Dunson, David B, Vehtari, Aki, and Rubin, Donald B. 2014. *Bayesian Data Analysis*. Vol. 2. CRC press.

Geyer, Charles J. 1992. Practical Markov chain Monte Carlo. *Statistical Science*, **7**(4), 473 – 483.

Girolami, Mark, and Calderhead, Ben. 2011. Riemann manifold Langevin and Hamiltonian Monte Carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **73**(2), 123–214.

Gong, Wenbo, Li, Yingzhen, and Hernández-Lobato, José Miguel. 2020. Sliced Kernelized Stein Discrepancy. In: *International Conference on Learning Representations*.

Gorham, Jackson, and Mackey, Lester. 2015. Measuring sample quality with Stein's method. Pages 226–234 of: *Advances in Neural Information Processing Systems*.

Gorham, Jackson, and Mackey, Lester. 2017. Measuring sample quality with kernels. Pages 1292–1301 of: *Proceedings of the 34th International Conference on Machine Learning*. PMLR.

Gorham, Jackson, Duncan, Andrew B, Vollmer, Sebastian J, and Mackey, Lester. 2019. Measuring sample quality with diffusions. *The Annals of Applied Probability*, **29**(5), 2884–2928.

Gorham, Jackson, Raj, Anant, and Mackey, Lester. 2020. Stochastic Stein discrepancies. *Advances in Neural Information Processing Systems*, **33**, 17931–17942.

Grathwohl, Will, Wang, Kuan-Chieh, Jacobsen, Jörn-Henrik, Duvenaud, David, and Zemel, Richard. 2020. Learning the stein discrepancy for training and evaluating energy-based models without sampling. Pages 3732–3747 of: *International Conference on Machine Learning*. PMLR.

Green, Peter J, and Mira, Antonietta. 2001. Delayed rejection in reversible jump Metropolis–Hastings. *Biometrika*, **88**(4), 1035–1053.

Grenander, Ulf, and Miller, Michael I. 1994. Representations of knowledge in complex systems. *Journal of the Royal Statistical Society: Series B (Methodological)*, **56**(4), 549–581.

Gustafson, Paul. 1998. A guided walk Metropolis algorithm. *Statistics and Computing*, **8**(4), 357–364.

Hastings, W Keith. 1970. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, **57**, 97–109.

Heidelberger, Philip, and Welch, Peter D. 1981. A Spectral Method for Confidence Interval Generation and Run Length Control in Simulations. *Communications of the ACM*, **24**(4), 233–245.

Hodgkinson, Liam, Salomone, Robert, and Roosta, Fred. 2020. The reproducing Stein kernel approach for post-hoc corrected sampling. *arXiv preprint arXiv:2001.09266*.

Hoffman, Matthew, Radul, Alexey, and Sountsov, Pavel. 2021. An Adaptive-MCMC Scheme for Setting Trajectory Lengths in Hamiltonian Monte Carlo. Pages 3907–3915 of: Banerjee, Arindam, and Fukumizu, Kenji (eds), *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*. Proceedings of Machine Learning Research, vol. 130. PMLR.

Hoffman, Matthew D, and Gelman, Andrew. 2014. The No-U-Turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research*, **15**(1), 1593–1623.

Horowitz, Alan M. 1991. A generalized guided Monte Carlo algorithm. *Physics Letters B*, **268**(2), 247–252.

Hsieh, Ya-Ping, Kavis, Ali, Rolland, Paul, and Cevher, Volkan. 2018. Mirrored Langevin Dynamics. Pages 2883–2892 of: *Advances in Neural Information Processing Systems*.

Huggins, Jonathan, and Mackey, Lester. 2018. Random feature Stein discrepancies. *Advances in Neural Information Processing Systems*, **31**.

Huggins, Jonathan, and Zou, James. 2017. Quantifying the accuracy of approximate diffusions and Markov chains. Pages 382–391 of: *Artificial Intelligence and Statistics*. PMLR.

Johndrow, James E, Pillai, Natesh S, and Smith, Aaron. 2020. *No free lunch for approximate MCMC*. arXiv:2010.12514.

Jones, Galin L, and Hobert, James P. 2001. Honest exploration of intractable probability distributions via Markov chain Monte Carlo. *Statistical Science*, **16**(4), 312–334.

Kamatani, Kengo. 2020. Random walk Metropolis algorithm in high dimension with non-Gaussian target distributions. *Stochastic Processes and their Applications*, **130**(1), 297–327.

Kanagawa, Heishiro, Barp, Alessandro, Simon-Gabriel, Carl-Johann, Gretton, Arthur, and Mackey, Lester. 2024. Controlling Moments with Kernel Stein Discrepancies. *arXiv preprint arXiv:2211.05408v4*.

Karvonen, Toni, Oates, Chris J, and Sarkka, Simo. 2018. A Bayes-Sard cubature method. *Advances in Neural Information Processing Systems*, **31**.

LeCam, Lucien. 1986. *Asymptotic methods in statistical decision theory*. Springer series in statistics. Springer.

Lewis, PA W, and Shedler, Gerald S. 1979. Simulation of nonhomogeneous Poisson processes by thinning. *Naval Research Logistics Quarterly*, **26**(3), 403–413.

Li, Wenzhe, Ahn, Sungjin, and Welling, Max. 2016. Scalable MCMC for mixed membership stochastic blockmodels. Pages 723–731 of: *Artificial Intelligence and Statistics*.

Lindvall, Torgny, and Rogers, L Cris G. 1986. Coupling of multidimensional diffusions by reflection. *The Annals of Probability*, 860–872.

Liu, Qiang, and Lee, Jason. 2017. Black-box importance sampling. Pages 952–961 of: *Artificial Intelligence and Statistics*. PMLR.

Liu, Qiang, Lee, Jason, and Jordan, Michael. 2016. A kernelized Stein discrepancy for goodness-of-fit tests. Pages 276–284 of: *International Conference on Machine Learning*. PMLR.

Livingstone, Samuel, and Zanella, Giacomo. 2022. The Barker Proposal: Combining Robustness and Efficiency in Gradient-Based MCMC. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, **84**(2), 496–523.

Ludkin, M, and Sherlock, C. 2022. Hug and hop: a discrete-time, nonreversible Markov chain Monte Carlo algorithm. *Biometrika*, **110**(2), 301–318.

L'Ecuyer, Pierre, and Lemieux, Christiane. 2002. Recent advances in randomized quasi-Monte Carlo methods. In: Dror, Moshe, L'Ecuyer, Pierre, and Szidarovszky, Ferenc (eds), *Modeling Uncertainty: An Examination of Stochastic Theory, Methods, and Applications*. Springer.

Ma, Yi-An, Chen, Tianqi, and Fox, Emily. 2015. A complete recipe for stochastic gradient MCMC. Pages 2917–2925 of: *Advances in Neural Information Processing Systems*.

Ma, Yi-An, Foti, Nicholas J, and Fox, Emily B. 2017. Stochastic gradient MCMC methods for hidden Markov models. Pages 2265–2274 of: *International Conference on Machine Learning*. PMLR.

Majka, Mateusz B, Mijatović, Aleksandar, and Szpruch, Łukasz. 2020. Non-asymptotic bounds for sampling algorithms without log-concavity. *The Annals of Applied Probability*, **30**(4), 1534–1581.

Metropolis, Nicholas, Rosenbluth, Arianna W, Rosenbluth, Marshall N, Teller, Augusta H, and Teller, Edward. 1953. Equation of state calculations by fast computing machines. *The journal of chemical physics*, **21**(6), 1087–1092.

Meyn, Sean P, and Tweedie, Richard L. 2012. *Markov Chains and Stochastic Stability*. Springer Science & Business Media.

Meyn, Sean P, Tweedie, Robert L, et al. 1994. Computable bounds for geometric convergence rates of Markov chains. *The Annals of Applied Probability*, **4**(4), 981–1011.

Michel, Manon, Kapfer, Sebastian C, and Krauth, Werner. 2014. Generalized event-chain Monte Carlo: Constructing rejection-free global-balance algorithms from infinitesimal steps. *The Journal of Chemical Physics*, **140**(5).

Michel, Manon, Durmus, Alain, and Sénécal, Stéphane. 2020. Forward event-chain Monte Carlo: Fast sampling by randomness control in irreversible Markov chains. *Journal of Computational and Graphical Statistics*, **29**(4), 689–702.

Nagapetyan, Tigran, Duncan, Andrew B, Hasenclever, Leonard, Vollmer, Sebastian J, Szpruch, Lukasz, and Zygalakis, Konstantinos. 2017. *The true cost of stochastic gradient Langevin dynamics*. arXiv:1706.02692.

Neal, Radford M. 2003. Slice sampling. *The Annals of Statistics*, **31**(3), 705–767.

Neal, Radford M. 2004. Improving asymptotic variance of MCMC estimators: Non-reversible chains are better. *arXiv preprint math/0407281*.

Neal, Radford M. 2011. MCMC using Hamiltonian dynamics. Pages 113–162 of: Brooks, Steve, Gelman, Andrew, Jones, Galin L, and Meng, Xiao-Li (eds), *Handbook of Markov chain Monte Carlo*. CRC Press.

Nemeth, Christopher, and Fearnhead, Paul. 2021. Stochastic gradient markov chain monte carlo. *Journal of the American Statistical Association*, **116**(533), 433–450.

Nemeth, Christopher, and Sherlock, Chris. 2018. Merging MCMC subposteriors through Gaussian-process approximations. *Bayesian Analysis*, **13**(2), 507–530.

Nemeth, Christopher, Fearnhead, Paul, and Mihaylova, Lyudmila. 2016. Particle approximations of the score and observed information matrix for parameter estimation in state–space models with linear computational cost. *Journal of Computational and Graphical Statistics*, **25**(4), 1138–1157.

Norris, James R. 1998. *Markov Chains*. Cambridge University Press.

Oates, Chris J, Girolami, Mark, and Chopin, Nicolas. 2017. Control functionals for Monte Carlo integration. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **79**(3), 695–718.

Oksendal, Bernt. 2013. *Stochastic Differential Equations: An Introduction with Applications*. Springer Science & Business Media.

Pagani, Filippo, Chevallier, Augustin, Power, Sam, House, Thomas, and Cotter, Simon. 2020. *NuZZ: numerical Zig-Zag sampling for general models*. arXiv:2003.03636.

Patterson, Sam, and Teh, Yee Whye. 2013. Stochastic gradient Riemannian Langevin dynamics on the probability simplex. Pages 3102–3110 of: *Advances in Neural Information Processing Systems*.

Peters, Elias A J F, and de With, G. 2012. Rejection-free Monte Carlo sampling for general potentials. *Physical Review E*, **85**(2), 026703.

Phillips, David B, and Smith, Adrian FM. 1996. Bayesian model comparison via jump diffusions. Pages 215–240 of: Gilks, Wally R, Richardson, Sylvia, and Spiegelhalter, David (eds), *Markov chain Monte Carlo in practice*. Chapman & Hall, CRC.

Pollock, Murray, Fearnhead, Paul, Johansen, Adam M, and Roberts, Gareth O. 2020. Quasi-stationary Monte Carlo and the ScaLE algorithm. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, **82**(5), 1167–1221.

Press, William H, Teukolsky, Saul A, Vetterling, William T, and Flannery, Brian P. 2007. *Numerical recipes in C++: The art of scientific computing*. Cambridge University Press.

Putcha, Srshti, Nemeth, Christopher, and Fearnhead, Paul. 2023. Preferential Subsampling for Stochastic Gradient Langevin Dynamics. Pages 8837–8856 of: *International Conference on Artificial Intelligence and Statistics*. PMLR.

Raginsky, Maxim, Rakhlin, Alexander, and Telgarsky, Matus. 2017. Non-convex learning via stochastic gradient langevin dynamics: a nonasymptotic analysis. Pages 1674–1703 of: *Conference on Learning Theory*. PMLR.

Rasmussen, Carl Edward, and Williams, Christopher K. I. 2005. *Gaussian Processes for Machine Learning*. The MIT Press.

Riabiz, Marina, Chen, Wilson Ye, Cockayne, Jon, Swietach, Pawel, Niederer, Steven A, Mackey, Lester, and Oates, Chris J. 2022. Optimal thinning of MCMC output. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, **84**(4), 1059–1081.

Riou-Durand, Lionel, and Vogrinc, Jure. 2023. *Metropolis Adjusted Langevin Trajectories: a robust alternative to Hamiltonian Monte Carlo*.

Ripley, Brian D. 2009. *Stochastic Simulation*. John Wiley & Sons.

Robbins, Herbert, and Monro, Sutton. 1951. A stochastic approximation method. *The annals of mathematical statistics*, 400–407.

Robert, Christian P. 2007. *The Bayesian Choice: from Decision-Theoretic Foundations to Computational Implementation*. Springer.

Robert, Christian P, and Casella, George. 1999. *Monte Carlo Statistical Methods*. Springer.

Roberts, Gareth O, and Rosenthal, Jeffrey S. 1998. Optimal scaling of discrete approximations to Langevin diffusions. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **60**(1), 255–268.

Roberts, Gareth O, and Rosenthal, Jeffrey S. 2001. Optimal scaling for various Metropolis-Hastings algorithms. *Statistical science*, **16**(4), 351–367.

Roberts, Gareth O, and Rosenthal, Jeffrey S. 2004. General state space Markov chains and MCMC algorithms. *Probability Surveys*, **1**, 20–71.

Roberts, Gareth O, and Tweedie, Richard L. 1996. Exponential convergence of Langevin distributions and their discrete approximations. *Bernoulli*, **2**(4), 341–363.

Roberts, Gareth O, and Tweedie, Richard L. 1999. Bounds on regeneration times and convergence rates for Markov chains. *Stochastic Processes and Their Applications*, **80**(2), 211–229.

Roberts, Gareth O., Gelman, Andrew, and Gilks, Walter R. 1997. Weak convergence and optimal scaling of random walk Metropolis algorithms. *The Annals of Applied Probability*, **7**, 110–120.

Rogers, Leonard CG, and Williams, David. 2000a. *Diffusions, Markov Processes, and Martingales: Volume 1, Foundations*. Cambridge University Press.

Rogers, Leonard CG, and Williams, David. 2000b. *Diffusions, Markov Processes, and Martingales: Volume 2, Ito Calculus*. Cambridge University Press.

Rosenthal, Jeffrey S. 1995. Minorization conditions and convergence rates for Markov chain Monte Carlo. *Journal of the American Statistical Association*, **90**(430), 558–566.

Rubinstein, R Y, and Kroese, D P. 2008. *Simulation and the Monte Carlo Method*. John Wiley & Sons.

Scott, Steven L, Blocker, Alexander W, Bonassi, Fernando V, Chipman, Hugh A, George, Edward I, and McCulloch, Robert E. 2016. Bayes and big data: The consensus Monte Carlo algorithm. *International Journal of Management Science and Engineering Management*, **11**(2), 78–88.

Sherlock, C., Thiery, A. H., Roberts, G. O., and Rosenthal, J. R. 2015. On the efficiency of pseudo-marginal random walk Metropolis algorithms. *Annals of Statistics*, **43**(1), 238–275.

Sherlock, Chris, and Roberts, Gareth. 2009. Optimal scaling of the random walk Metropolis on elliptically symmetric unimodal targets. *Bernoulli*, **15**(3), 774–798.

Sherlock, Chris, and Thiery, Alexandre H. 2022. A discrete bouncy particle sampler. *Biometrika*, **109**(2), 335–349.

Sherlock, Chris, Urbas, Szymon, and Ludkin, Matthew. 2023. The apogee to apogee path sampler. *Journal of Computational and Graphical Statistics*, **32**(4), 1436–1446.

Shi, Jiaxin, Zhou, Yuhao, Hwang, Jessica, Titsias, Michalis, and Mackey, Lester. 2022. Gradient estimation with discrete Stein operators. *Advances in neural information processing systems*, **35**, 25829–25841.

Sohl-Dickstein, Jascha, Mudigonda, Mayur, and DeWeese, Michael. 2014. Hamiltonian Monte Carlo without detailed balance. Pages 719–726 of: *International Conference on Machine Learning*. PMLR.

Stein, Charles. 1972. A bound for the error in the normal approximation to the distribution of a sum of dependent random variables. Pages 583–603 of: *Proceedings of the 6th Berkeley Symposium on Mathematical Statistics and Probability, Volume 2: Probability Theory*, vol. 6. University of California Press.

Stephens, Matthew. 2000. Bayesian analysis of mixture models with an unknown number of components-an alternative to reversible jump methods. *Annals of Statistics*, 40–74.

Sun, Hongwei. 2005. Mercer theorem for RKHS on noncompact sets. *Journal of Complexity*, **21**(3), 337–349.

Sun, Yi, Schmidhuber, Jürgen, and Gomez, Faustino. 2010. Improving the asymptotic performance of Markov chain Monte-Carlo by inserting vortices. Pages 2235–2243 of: Lafferty, J., Williams, C.K.I., Shawe-Taylor, J., Zemel, R.S., and Culotta, A. (eds), *Advances in Neural Information Processing Systems*, vol. 23.

Sutton, Matthew, and Fearnhead, Paul. 2023. Concave-convex PDMP-based sampling. *Journal of Computational and Graphical Statistics*, **32**(4), 1425–1435.

Suwa, Hidemaro, and Todo, Synge. 2010. Markov chain Monte Carlo method without detailed balance. *Physical Review Letters*, **105**(12), 120603.

Teh, Yee Whye, Thiery, Alexandre H, and Vollmer, Sebastian J. 2016. Consistency and fluctuations for stochastic gradient Langevin dynamics. *The Journal of Machine Learning Research*, **17**(1), 193–225.

Teymur, Onur, Gorham, Jackson, Riabiz, Marina, and Oates, Chris. 2021. Optimal quantisation of probability measures using maximum mean discrepancy. Pages 1027–1035 of: *International Conference on Artificial Intelligence and Statistics*. PMLR.

Turitsyn, Konstantin S, Chertkov, Michael, and Vucelja, Marija. 2011. Irreversible Monte Carlo algorithms for efficient sampling. *Physica D: Nonlinear Phenomena*, **240**(4-5), 410–414.

Vanetti, Paul, Bouchard-Côté, Alexandre, Deligiannidis, George, and Doucet, Arnaud. 2017. *Piecewise-deterministic Markov chain Monte Carlo*. arXiv:1707.05296.

Vats, Dootika, and Knudson, Christina. 2021. Revisiting the Gelman–Rubin diagnostic. *Statistical Science*, **36**(4), 518–529.

Vehtari, Aki, Gelman, Andrew, Simpson, Daniel, Carpenter, Bob, and Bürkner, Paul-Christian. 2021. Rank-normalization, folding, and localization: An improved $\hat{R}$ for assessing convergence of MCMC. *Bayesian Analysis*, **1**(1), 1–28.

von Renesse, Max-K, and Sturm, Karl-Theodor. 2005. Transport inequalities, gradient estimates, entropy and Ricci curvature. *Communications on pure and applied mathematics*, **58**(7), 923–940.

Vyner, Callum, Nemeth, Christopher, and Sherlock, Chris. 2023. SwISS: A scalable Markov chain Monte Carlo divide-and-conquer strategy. *Stat*, **12**(1), e523.

Welling, Max, and Teh, Yee W. 2011. Bayesian learning via stochastic gradient Langevin dynamics. Pages 681–688 of: *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*.

Wenliang, Li K, and Kanagawa, Heishiro. 2021. Blindness of score-based methods to isolated components and mixing proportions. In: *Proceedings of the NeurIPS Workshop "Your Model is Wrong: Robustness and Misspecification in Probabilistic Modeling"'*.

Wu, Changye, and Robert, Christian P. 2017. *Generalized bouncy particle sampler*. arXiv:1706.04781.

Wu, Changye, and Robert, Christian P. 2020. Coordinate sampler: a non-reversible Gibbs-like MCMC sampler. *Statistics and Computing*, **30**(3), 721–730.

Xifara, T., Sherlock, C., Livingstone, S., Byrne, S., and Girolami, M. 2014. Langevin diffusions and the Metropolis-adjusted Langevin algorithm. *Statistics & Probability Letters*, **91**, 14–19.

Index

acceptance probability, 43, 44, 48–50, 56–58, 60, 113–115, 120–122, 178
acceptance rate, 47, 49–51, 53, 56–58, 61–63, *see* acceptance probability
acceptance ratio, 46, 48, 49, 51, 55–58, 61, 62
aperiodicity, 20
auto-correlation, 21, 53, 62, 90, 91
auto-correlation time, *see* integrated auto-correlation time
autocorrelation, 114

batch, 216, 217
Bayesian matrix factorisation, 15, 16, 164
Bayesian neural network, 16, 17, 97, 98
bias diagnostic, 194
black-box importance sampling, 219
boomerang sampler, 187–190
bouncy particle sampler, 144–148, 150–152, 155–160, 164, 165, 178, 179, 182, 185–188, 191
BPS, *see* bouncy particle sampler
Brownian motion, 22
burn-in, 45, 47, 65, 73, 99, 107, 154, 193, 222, 224

Cauchy–Schwarz, 30, 206, 209, 211
Cauchy-Schwarz, 161, 172
central limit theorem, 6, 13, 61
component-wise, 48, 53, 54, 62
component-wise updates, 113, 114
concave-convex decomposition, 161, 164, 165
constant velocity dynamics, 138, 145, 186
contraction, 200, 202, 203, 205, 210, 213
control variate, 9–11, 72–75, 77–79, 83, 90, 94, 170, 172, 173
convergence control, 209, 210, 212, 214, 215, 217, 222, 225
convergence detection, 225
convergence diagnostic, 192–194, 196, 199, 225
coordinate sampler, 140–142, 147, 148, 152, 160, 178
curse of dimension, 9, 14, 226

delayed rejection, 119–121
detailed balance, 19, 42, 48, 60, 106, 112, 120, 121
diagnostic, 94
discontinuous target, 176, 178
discrepancy, 201, 202
discrete bouncy particle sampler, 121–124, 144, 191
discrete BPS, *see* discrete bouncy particle sampler
distantly dissipative, 82, 210, 212, 213, 215
divergence theorem, 206
dominated convergence theorem, 21, 206
Dudley metric, 211, 212

effective sample size, 21, 53, 155, 175
eigenfunction, 33, 34, 36, 39
eigenvalue, 33, 34, 36–39
ergodic theorem, 20
Euler–Maruyama, 54
Euler-Maruyama, 22, 23, 25, 66, 67, 82, 86, 87, 101
extended posterior distribution, 112

Gelman–Rubin diagnostic, 193–196
generalised moment, 196, 198, 208
generator, 25–27, 203
  of PDMP, 134–136
Gibbs move, 48, 49
Gibbs moves, 152
gradient, 15, 54–56, 64, 65, 69–72, 75, 76, 78, 79, 121, 123, 140, 172, 173, 175, 213, 216, 217
  control variate estimator, 170, 172, 173

subsampling estimator, 170
greedy algorithm, 223
guided random walk, 113–119, 124, 127, 128, 143
Gustafson's algorithm, *see* guided random walk

Hamiltonian Monte Carlo, 57, 59, 64, 85–91, 93, 111–113, 125, 152, 188
Hamiltonian Monte Carlo:(, 109
Hessian, 91, 160, 164, 171, 189
Hidden Markov model, 103, 104
Hilbert space, 37, 198
HMC, *see* Hamiltonian Monte Carlo, *see* Hamiltonian Monte Carlo

importance sampling, 7, 11, 13, 14, 40, 42, 43, 50, 219, 222
independence sampler, 42, 49
infinitesimal generator, *see* generator
inner product, 28–31, 34–39, 207
inner product space, 28, 35, 37
integral probability metric, 200, 201, 203, 210, 226
integrated auto-correlation time, 17, 21, 40, 53, 90, 159
invariant distribution, 135, 136, 138, 140, 141, 143, 145, 146, 148, 177, 178, 182, 185, 186
inverse multi-quadric, 212–214, 216
irreducibility, 20, 118, 122, 127, 141, 145, 152

Jacobian, 60, 61, 117, 186

kernel
  Markov transition, 130, 135, 137, 141, 143, 177–179
kernel (Gaussian), 35, 37–39
kernel (Markov transition), 200, 202
kernel (matrix-valued), 206, 208
kernel (scalar-valued), 208, 212, 214, 219
kernel (trace-class), 37, 38
kernel cubature, 220
kernel Stein discrepancy, 91, 208, 209, 211, 213, 215, 216, 219, 221
kernel Stein discrepancy (sliced), 215
kernel Stein discrepancy (stochastic), 217
kernel trick, 28, 35, 196, 199, 207

Langevin diffusion, 22, 25, 26, 53, 54, 56, 203, 210, 213, 226
  overdamped, 26, 27, 66, 67, 71, 85
  underdamped, 25–27, 85, 88
Langevin dynamics, 67, 100, 102
leapfrog dynamics, 58–62, 64, 111, 125
lifting schemes, 112–119, 125, 127
log-concave, 203, 204, 210
logistic regression, 14, 92, 93, 95, 164, 167, 171, 173, 179, 183, 189

MALA, *see* Metropolis–adjusted Langevin algorithm
Markov chain
  continuous-time, 129, 134
  discrete-time, 17, 43, 106, 107
Markov chain Monte Carlo, 4, 5, 14, 17, 42, 63, 65, 67, 71, 72, 81, 85, 89–91, 93–95, 104, 105, 111, 117, 125, 126, 130, 139–140, 159, 173, 191
mass matrix, 58–61, 87, 110
MCMC, *see* Markov chain Monte Carlo, *see* Markov chain Monte Carlo
Mercer's theorem, 36
Metropolis–adjusted Langevin algorithm, 43, 53, 67–72, 90, 91, 93
Metropolis–Hastings, 42–45, 47–49, 57, 59, 63, 67, 79, 90, 91, 113, 114, 125
MH, *see* Metropolis–Hastings
mirrored Langevin algorithm, 101, 102
mixing, 45, 56, 85, 87, 89, 94, 100, 101, 108, 114, 118, 119, 124, 139, 141, 146, 152, 155–159, 170, 173, 175, 178, 191, 201
Monte Carlo integration, 4–7, 9, 11–13

non-reversible MCMC, 106–109, 112, 113, 117, 121, 122, 124, 126–129, 139
numerical integration, 7, 163

Ornstein–Uhlenbeck process, 22–24
orthogonal, 30, 34, 118
orthonormal, 30, 31, 33–37

PDMP, *see* piecewise deterministic Markov process
period, 20, 62
piecewise deterministic Markov processes, 129–130, 134–136, 139–153, 190
  comparison of samplers, 155
  output, 154–155, 159
  reversible jump, 179–184
  simulation, 130–153, 159–165, 175
Poisson thinning, 132–133, 160, 163, 169, 170, 173, 175, 184
posterior distribution, 4, 12, 14, 15, 17, 42, 45, 65, 68, 71, 75, 78, 80, 81, 86,

89–94, 98, 101–105, 120, 139, 160, 170, 180, 182, 216
preconditioned, 50, 53–55, 58, 64, 89
proposal, 43, 47–51, 53–55, 57–59, 113, 117, 120–123, 152, 178
proposal distribution, 7, 13, 14

quadrature, 7, 8

random features, 225
random walk Metropolis, 42, 43, 48–50, 113, 114, 124
reducible, 20, 48, 117, 145, 151, 188
reflection, 122, 125, 144–146, 178, 179, 185, 187
refresh event/rate, 140, 141, 145–152, 157, 159, 187, 188, 191
reproducing kernel Hilbert space, 28, 37
reproducing kernel Hilbert space (vector-valued), 207
reproducing property, 207
reversibility, 17, 19, 106
reversible, 42, 44
Riesz representation theorem, 207
Rosenbrock, 220, 223

sample path, 129, 154, 194, 196
scaling, 46, 50–57, 61–64, 109, 157, 173, 175, 191
SDE, *see* stochastic differential equation
separation, 225
SGHMC, *see* stochastic gradient Hamiltonian Monte Carlo
SGLD, *see* stochastic gradient Langevin dynamics
SGMCMC, *see* stochastic gradient Markov chain Monte Carlo
SGRLD, *see* stochastic gradient Riemannian Langevin dynamics
stationary distribution, 4, 17–20, 24–26, 43, 53, 54, 58, 65–68, 84–87, 107, 112, 114, 134, 137, 146, 164, 176
Stein class, 202
Stein discrepancy, 202
Stein operator, 202
Stein's method, 91, 225
step size, 58, 61, 62, 71, 72, 77, 81, 82, 84, 89–92, 94, 100, 109, 113, 118, 124, 128
stochastic differential equation, 22, 23, 66, 85–89
stochastic gradient, 82, 85–88, 104, 194, 216, 217
stochastic gradient Hamiltonian Monte Carlo, 85, 87–90, 93, 94, 98, 99
stochastic gradient Langevin dynamics, 69–85, 87, 90–94, 98–101, 105, 170
stochastic gradient Markov chain Monte Carlo, 65, 85–87, 89–91, 93–95, 97–100, 103, 104
stochastic gradient Riemannian Langevin dynamics, 89, 102
strong law of large numbers, 6, 12, 20, 196
subsample, 77–79, 86, 90, 94, 104
subsampling, 170
superpositon, 133–134, 143, 147
symmetric set, 201

tall data, 216
thinning, 222

ULA, *see* unadjusted Langevin algorithm
unadjusted Langevin algorithm, 65–71, 79, 81, 91

vector space, 28–31, 36, 37

warm-up, 45
Wasserstein metric, 68, 81, 82, 90–92, 203, 204, 210–213
weak convergence, 212, 213, 216, 217, 222, 225
weight, 216, 219, 220, 222, 223

Zig–Zag sampler, 167
Zig-Zag sampler, 142–144, 147–150, 152, 155, 157, 159, 160, 165, 168–173, 175, 178, 179, 182, 184, 185