Piecewise-deterministic Markov Chain Monte Carlo

Paul Vanetti

St. Peter's College
University of Oxford

A thesis submitted for the degree of
Doctor of Philosophy

Michaelmas 2019

Abstract

Recent interest in a class of Markov chain Monte Carlo schemes based on continuous-time piecewise-deterministic Markov processes has led to several new and promising algorithmic developments. Prominent examples include the zig-zag process and the bouncy particle sampler. We explore this class of algorithms, proposing extensions and drawing connections to existing literature.

Two key aspects of a continuous-time piecewise-deterministic process are the *flow* and the *event kernel*. We discuss conditions on these sufficient to design a process targeting a distribution of interest. Based on these conditions, we show how the flow can be extended to Hamiltonian dynamics when exactly computable. For the event kernel, we demonstrate that a simple factorization of the invariant distribution subsumes all existing kernels and allows new kernels to be easily derived.

Next, we explore the idea of piecewise-deterministic processes in *discrete* time. These algorithms have many close connections with pre-existing algorithms. The simultaneous simulation of multiple event rates may be connected to the discrete-time *delayed acceptance* algorithm. We demonstrate that the flow and event kernels used in the bouncy particle sampler bear strong resemblance to the *reflective slice sampler*. A class of discrete-time event kernels which we call here *tunnelling* algorithms also prove to be a special case of this framework.

Applying this framework to the setting of large datasets with many observations, we describe how both the continuous-time and discrete-time algorithms can be adapted to this setting. We explore the use of approximations to bound the effects of individual observations. Methods of simulating from discrete distributions based on Poisson processes are used to efficiently subsample the data. Finally, we show how these methods can be used to extend and improve upon the existing firefly Monte Carlo algorithm.

A second application is found in the setting where interaction between parameters is limited and forms a sparse graph, for which the continuous-time *local* bouncy particle sampler algorithm has been developed. A discrete-time extension mimicking the local properties of this algorithm is proposed, unveiling a close connection to the random cluster algorithm of Swendsen-Wang.

Piecewise-deterministic Markov Chain Monte Carlo

Paul Vanetti

St. Peter's College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Michaelmas 2019

For my father.

Acknowledgements

I am indebted to my supervisor, Professor Arnaud Doucet, for his support throughout my graduate studies. His generosity with his time, unfailing patience, and encyclopedic knowledge of all things Monte Carlo have been immensely helpful throughout this process. It is only with his continued openness and encouragement that any of this has been possible.

I also wish to extend thanks to my collaborators Alexandre Bouchard-Côté, Rob Cornish, and George Deligiannidis for all their efforts in pushing forward our understanding.

I am grateful to Pierre Jacob for sponsoring a visit to Harvard which was a productive and expanding experience.

A special thanks to my CDT cohort: Andi, Beniamino, Ella, Guiseppe, Jack, Kaspar, Leon, Marcin, Nathan, Sherman, and Xenia for making our first year so memorable. Thanks also to the other members of the research group: Alex, Anthony, James, Jeremy, and Sebastian, for their camaraderie and many stimulating discussions. And to the first floor regulars: Dominic, Fadhel, and Marco, for the many stimulated discussions over tea and at the Oak.

My studies have been generously supported by the Clarendon Fund.

Abstract

Recent interest in a class of Markov chain Monte Carlo schemes based on continuous-time piecewise-deterministic Markov processes has led to several new and promising algorithmic developments. Prominent examples include the zig-zag process and the bouncy particle sampler. We explore this class of algorithms, proposing extensions and drawing connections to existing literature.

Two key aspects of a continuous-time piecewise-deterministic process are the *flow* and the *event kernel*. We discuss conditions on these sufficient to design a process targeting a distribution of interest. Based on these conditions, we show how the flow can be extended to Hamiltonian dynamics when exactly computable. For the event kernel, we demonstrate that a simple factorization of the invariant distribution subsumes all existing kernels and allows new kernels to be easily derived.

Next, we explore the idea of piecewise-deterministic processes in *discrete* time. These algorithms have many close connections with pre-existing algorithms. The simultaneous simulation of multiple event rates may be connected to the discrete-time *delayed acceptance* algorithm. We demonstrate that the flow and event kernels used in the bouncy particle sampler bear strong resemblance to the *reflective slice sampler*. A class of discrete-time event kernels which we call here *tunnelling* algorithms also prove to be a special case of this framework.

Applying this framework to the setting of large datasets with many observations, we describe how both the continuous-time and discrete-time algorithms can be adapted to this setting. We explore the use of approximations to bound the effects of individual observations. Methods of simulating from discrete distributions based on Poisson processes are used to efficiently subsample the data. Finally, we show how these methods can be used to extend and improve upon the existing firefly Monte Carlo algorithm.

A second application is found in the setting where interaction between parameters is limited and forms a sparse graph, for which the continuous-time *local* bouncy particle sampler algorithm has been developed. A discrete-time extension mimicking the local properties of this algorithm is proposed, unveiling a close connection to the random cluster algorithm of Swendsen-Wang.

Contents

List of Figures

1 Introduction

This thesis concerns recent developments in Markov chain Monte Carlo methods, a class of algorithms which provide numerical approximations to integrals. Any Monte Carlo method works on the basic premise that an expectation with respect to a probability distribution can be approximated using samples from that distribution:

$$\mathbb{E}_\pi[f(X)] \approx \frac{1}{N}\sum_{i=1}^{N} f(X_i) \quad X_i \sim \pi(\cdot).$$

Such integrals appear in a wide range of applications; we focus on the application to Bayesian statistics, discussed in the next section. While for the numerical experiments in this thesis we will focus on these Bayesian models, we emphasize that the frameworks and concepts used here are more general.

We do not give a thorough motivation or background for Markov chain Monte Carlo, but instead refer the reader to any of the many textbooks and reviews available. For a recent review see Brooks et al. (2011). In Section 1.2 we review several widely-used methods for constructing Markov chain Monte Carlo algorithms which we rely on heavily throughout the thesis. This review will serve to establish notation and terminology.

The structure of the thesis is as follows:

- In Chapter 2, we review *continuous-time Markov chain Monte Carlo* methods, a topic which has recently received substantial interest. We describe extensions of two main aspects of the algorithm: the flow, and the event kernel.

- In Chapter 3, we propose discrete-time versions of the algorithms discussed in Chapter 2. We show that these algorithms are closely connected to several algorithms in the literature. A generalization of *slice variables* is proposed to account for a family of existing algorithms. We show that the event-time simulation methods of continuous time can still be leveraged in discrete time. Finally, we review and provide a unifying framework called *tunnelling* which subsumes several pertinent discrete-time algorithms.

- In Chapter 4, we focus on the application of methods discussed in Chapters 2 and 3 to the problem of Bayesian inference in large datasets. We describe several extensions necessary to adapt to this setting. Some of this work has appeared as Cornish et al. (2019).

- In Chapter 5, we consider models where the interaction between parameters is limited. We demonstrate an extension to our discrete-time algorithms which connects these algorithms to the well-known family of *random-cluster* algorithms.

1.1 Numerical integration in Bayesian statistics

The main object of interest in Bayesian statistics is the *posterior distribution*, which is the distribution over some unknown model parameters which we wish to infer. A Bayesian model specifies a *likelihood* $\pi(y|x)$ which describes how the data y are generated from the parameters x; and a *prior distribution* $\pi(x)$, which ostensibly represents the modeller's prior beliefs about the distribution of the parameter x. The eponymous Bayes' theorem provides the relationship between the posterior $\pi(x|y)$, the likelihood, and the prior:

$$\pi(x|y) = \frac{\pi(y|x)\pi(x)}{\int \pi(y|x)\pi(x)\mathrm{d}x}.$$

Any quantities of interest related to the posterior can be expressed as *expectations*, where for some function f we compute

$$\mathbb{E}[f(X)] = \int f(x)\pi(x|y)\mathrm{d}x.$$

Typical functions are e.g. $f(x) = x$ yielding the mean of the posterior, $f(x) = x^2$ yielding the second moment, $f(x) = \mathbf{1}_A(x)$ indicating the probability that $x \in A$, etc.

When the posterior is in the same distribution family as the prior, the prior and likelihood are said to be *conjugate*. In such cases the posterior is typically a familiar distribution which can thus be easily interpreted, and expectations may often be exactly computable.

When the prior is *non-conjugate* then expectations over the distribution cannot be computed exactly. In this case, we turn to numerical solutions of the integral. Markov chain Monte Carlo is well-suited here, as we are only required to evaluate the density $\pi(x|y)$ up to a normalizing constant, so we can simply use $\pi(y|x)\pi(x)$.

We highlight two commonly used models which will serve as examples in our numerical experiments.

- Logistic regression, widely used for data with binary labels. For N observations we have for $n \in \{1, \ldots, N\}$ a set of covariates $\xi_n \in \mathbb{R}^d$ and a label $y_n \in \{-1, 1\}$. The labels are independent given the covariates and the regression parameters x, with likelihood

  $$p(y_n = 1|x) = \frac{1}{1 + \exp(-\langle \xi_n, x \rangle)}.$$

  There is no known conjugate prior for this likelihood, motivating the use of numerical methods to compute posteriors for this model. Often the normal distribution is used as a prior over x.

- Latent Gaussian fields with Poisson observations, widely used in spatial statistics. Here, the model of data generation first posits a randomly generated latent field

  $$x \sim \mathcal{N}(0, \Lambda^{-1})$$

where $\mathcal{N}$ is the normal distribution, and Λ is a sparse precision matrix which describes the relationships between the coordinates of x. The observations y are conditioned on the latent field with some likelihood; for the Poisson model this is

$$y_i \sim \text{Poisson}(\exp(x_i)).$$

Again, this model is not conjugate and requires some sort of numerical approximation.

The two models differ in a fundamental way: in the first, the data are independent given the parameters. In the second, the data are not independent, but the structure of Λ limits dependencies between spatially distant parts of the model.

1.2 Some Markov chain Monte Carlo concepts

This thesis will primarily be concerned with the development of novel Markov chain Monte Carlo algorithms. While some of these algorithms may seem complicated at first glance, they will often be understandable as instances of some well-known constructions. In this section we provide a cursory overview of some of the main ideas which will be used throughout this thesis.

1.2.1 The Metropolis-Hastings algorithm

The first appearance of a Markov chain Monte Carlo algorithm was the Metropolis algorithm (Metropolis et al. 1953). That work was concerned with the distribution of the positions of a number of particles in a two-dimensional square area, where the potential energy of the system is expressed as a sum of potentials corresponding to interactions between particles. The authors point out that a rejection sampling or importance sampling scheme is infeasible, and instead propose a scheme in which a perturbation of a single particle is proposed and then accepted or rejected according to the probability

$$\min\left\{1, \frac{\pi(x')}{\pi(x)}\right\}$$

where $\pi(x^*)$ is the density of the new state and $\pi(x)$ is the density of the incumbent state, and present an argument demonstrating that this yields the correct equilibrium distribution.

One advantage of this method is that we do not require access to a normalized density $\pi(x)$, but only require some $\gamma(x)$ such that $\pi(x) = \frac{1}{Z}\gamma(x)$, as the ratio $\pi(x^*)/\pi(x) = \gamma(x^*)/\gamma(x)$. We shall often use π in cases where we only need the unnormalized density γ, as in these cases it will be clear that the unnormalized density can be instead used.

Metropolis et al. (1953) considered only symmetric perturbations which we call *symmetric proposals*. Here, for current and proposed state x and x^*, if $q(x, x^*)$ is the density of the proposal, then proposing x from x^* has the same density, i.e. $q(x, x^*) = q(x^*, x)$. This was generalized by Hastings (1970) to general proposals by instead using an acceptance probability

$$\min\left\{1, \frac{\pi(x')q(x^*, x)}{\pi(x)q(x, x^*)}\right\}$$

which is now known as the Metropolis-Hastings algorithm. Other acceptance probabilities have been proposed, such as that of Barker (1965), however the Metropolis-Hastings acceptance probability maximizes the overall probability of accepting the proposed state which has the intuitive benefit of exploring the state space more quickly. It was proven by Peskun (1973) that this acceptance probability is optimal in the sense that it minimizes the asymptotic variance of estimates based on the resulting Markov chain; we point out that this result is only relevant to reversible Markov chains which we define in the next section.

Algorithm 1 A single step of Metropolis-Hastings.

Sample $x^* \sim q(x, \cdot)$.
with probability

$$\min\left\{1, \frac{\pi(x^*)q(x^*, x)}{\pi(x)q(x, x^*)}\right\}$$

    return x^*.
otherwise
    return x.

1.2.2 Global balance and detailed balance

In order that they target the correct distribution π, one property that all Markov chain Monte Carlo algorithms must satisfy is *global balance*. Taking the kernel (Nummelin 2004; section 1.1) of the Markov chain as $K(x, x')$ — distinguished from the proposal distribution $q(x, x^*)$ as it also includes the effect of the acceptance step — global balance is defined as

$$\int \pi(x)K(x, x')\mathrm{d}x = \pi(x'). \tag{1.1}$$

When this condition is satisfied, then if x is distributed according to π then x' will also be distributed according to π, thus the Markov chain is said to be *invariant* with respect to π (or π-invariant). Taken along with other conditions on aperiodicity and irreducibility then samples from the chain will converge to the target distribution π.

It will typically be impossible to directly verify the global balance equation for a given kernel as the integral on the left-hand-side cannot be computed. Instead, the sufficient condition of *detailed balance* is used instead, which stipulates that

$$\pi(x)K(x, x') = \pi(x')K(x', x). \tag{1.2}$$

Any kernel satisfying (1.2) will satisfy (1.1); this can be seen by taking the integral over x. For many Markov chain Monte Carlo algorithms, this latter condition can easily be proved. Kernels satisfying this condition are known as *reversible*. For many of the algorithms developed in this thesis we will primarily be concerned with establishing detailed balance in order to show that they sample from the correct distribution.

1.2.3 Acceptance probabilities for deterministic proposals

It is possible to use a deterministic transformation of the state x instead of a random proposal distribution (Tierney 1998); here we consider the case for real-valued state spaces. Define a one-to-one transformation $f : \mathbb{R}^d \mapsto \mathbb{R}^d$ where $f^{-1} = f$. The acceptance probability of such a transform is

$$\alpha(x, f(x)) = \min\left\{1, \frac{\pi(f(x))}{\pi(x)}|\mathbf{J}_f(x)|\right\} \tag{1.3}$$

where $|\mathbf{J}_f(x)|$ is the determinant of the Jacobian of f evaluated at x, i.e. the matrix of partial derivatives of the coordinates of f with respect to the coordinates of x:

$$\mathbf{J}_f(x) = \begin{bmatrix} \frac{\partial f_1(x)}{\partial x_1} & \frac{\partial f_1(x)}{\partial x_2} & \cdots & \frac{\partial f_1(x)}{\partial x_d} \\ \frac{\partial f_2(x)}{\partial x_1} & \frac{\partial f_2(x)}{\partial x_2} & \cdots & \frac{\partial f_2(x)}{\partial x_d} \\ \vdots & & \ddots & \vdots \\ \frac{\partial f_d(x)}{\partial x_1} & \frac{\partial f_d(x)}{\partial x_2} & \cdots & \frac{\partial f_d(x)}{\partial x_d} \end{bmatrix}.$$

Throughout this work, we will have cause to analyze many algorithms involving deterministic proposals, and will present correctness results for such deterministic proposals. It is fortunately possible to describe typical Metropolis-Hastings algorithms in terms of deterministic proposals; as such we argue the more general framework for establishing correctness of Markov chain Monte Carlo algorithms is that of deterministic proposals.

We will incorporate Metropolis-Hastings into this framework by taking an augmented density $\pi(x, x^*) = \pi(x)q(x, x^*)$ where $q(x, x^*)$ represents the density of the proposal distribution. The deterministic proposal we then consider is a *swap* of x and x^*. Thus we might write

$$f\left(\begin{bmatrix} x \\ x^* \end{bmatrix}\right) = \begin{bmatrix} 0 & I \\ I & 0 \end{bmatrix} \begin{bmatrix} x \\ x^* \end{bmatrix}.$$

The determinant of the Jacobian of f is one, so we are left with an acceptance probability using the ratio of $\pi(x^*, x)$ to $\pi(x, x^*)$ which is equivalent to the Metropolis-Hastings acceptance ratio for the same target and proposal:

$$\alpha((x, x^*), (x^*, x)) = \min\left\{1, \frac{\pi(x^*, x)}{\pi(x, x^*)}|\mathbf{J}_f(x)|\right\} = \min\left\{1, \frac{\pi(x^*)q(x^*, x)}{\pi(x)q(x, x^*)}\right\}.$$

Using this and similar constructions will allow us to develop algorithms using random proposals whenever we have deterministic ones. We will do so throughout this thesis, where our primary focus will be in developing proposal schemes and acceptance rates in a deterministic setting; for many of these we will also provide versions allowing randomized proposals by using a framework like the one here.

1.2.4 Composing kernels

It will prove useful to *compose* individual kernels. Taking kernels $K_1 K_2 \dots K_m$, each satisfying detailed balance as above, we consider two ways of combining them to form a composite kernel (Tierney 1994):

1. as a **cycle** taking each in turn, applying K_1, then K_2, etc., until K_m; or
2. as a **mixture**, where we have mixture probabilities $p_1, p_2, \dots, p_m$ such that $\sum_{i=1}^{m} p_i = 1$ and take the composite kernel

$$K_{\text{mix}} = \sum_{i=1}^{m} p_i K_i.$$

When composed as a cycle this is often called a *deterministic scan*; as a mixture, a *random scan*. Deterministic cycles of kernels generally exhibit *non-reversibility*, discussed in Section 1.2.8. We will employ both strategies extensively throughout this thesis.

1.2.5 Gibbs sampling

One powerful method known as *Gibbs sampling* relies on the capability of sampling the conditional of a coordinate. Say the state space is multidimensional, $x = (x_1, x_2, \dots, x_d) \in \mathcal{X}_1 \times \mathcal{X}_2 \times \cdots \times \mathcal{X}_d$. A Gibbs sampling kernel will update a single coordinate x_i from its conditional distribution

$$\pi(x_i | x_1, x_2, \dots, x_{i-1}, x_{i+1}, \dots, x_d).$$

The d kernels, each targeting a coordinate of x, will be composed as a cycle or as a mixture as described in Section 1.2.4. As this relies on the availability of methods to sample from the conditional distributions of the individual coordinates, it is not applicable in general. When feasible, Gibbs sampling is appealing as it often allows for samplers which quickly converge to the target distribution while avoiding the necessity of selecting or tuning a proposal distribution.

Some authors use the term Gibbs sampling to refer to Markov chain Monte Carlo algorithms which use a composition of kernels, each updating an individual

coordinate, irrespective of whether these kernels sample from the conditional distribution or not. This leads to e.g. the terminology of "Metropolis-within-Gibbs."

1.2.6 Auxiliary variable methods

Many very effective Markov chain Monte Carlo algorithms rely on *auxiliary variables*, which is rather generally and simply the idea of targeting some joint distribution $\pi(x, y)$ over x and the auxiliary variable y which yields the target as a marginal distribution:

$$\int \pi(x, y) \mathrm{d}y = \pi(x)$$

and thus sampling pairs of x and y according to $\pi(x, y)$ yields x samples distributed according to $\pi(x)$. We shall at times use this sort of notation where π represents both the marginal and augmented densities; the density in question will be disambiguated by the symbols used as arguments.

One main category of auxiliary variable algorithm is *data augmentation*, where typically an auxiliary variable is introduced for each data point. In many situations, this is done to yield conditional distributions of known form which can subsequently be sampled using a Gibbs scheme. A famous example is the algorithm of Albert and Chib (1993) applied to Bayesian probit regression, where the augmented variable is a univariate truncated normal distribution, and conditional on this, the regression parameters are normally distributed.

The constructions of many other auxiliary variable algorithms are highly intuitive and the auxiliary variables are often interpretable in that they admit particular algorithmic developments. We discuss some of these in subsequent sections.

1.2.7 Slice constructions

We refer to *slice constructions* or *slice variables* as a particular form of auxiliary variable wherein, for some density $\pi(x) \propto \gamma(x)$, we take the distribution

augmented by $u \in \mathbb{R}_+$:

$$\begin{aligned}\pi(x, u) &\propto \gamma(x, u) \\ &= \mathbb{I}[u < \gamma(x)]. \end{aligned} \tag{1.4}$$

This auxiliarization was used by Neal (2003) to construct a sampler based on the conditionals

$$\begin{aligned}\pi(u|x) &= \text{Uniform}(0, \gamma(x)) \\ \pi(x|u) &\propto \mathbf{1}_{\{x:\gamma(x)>u\}}(x).\end{aligned}$$

The uniformity of these conditionals can yield efficient updates. Such algorithms are collectively known as *slice sampling*.

We are less interested in these sorts of conditional updates but instead we shall rely on the augmentation (1.4) to facilitate construction of complex proposals, the validity of which can be more easily verified using slice variables.

Using slice variables to establish correctness is useful since in many cases an algorithm which samples a slice variable in each iteration is equivalent to one which was formulated without the use of slice variables. For example, combining the deterministic proposal construction of Section 1.2.3 with a slice variable, we can easily see that this is equivalent to the Metropolis-Hastings algorithm. However, the accept-reject step now becomes entirely deterministic; we have essentially incorporated all sources of randomness into the augmented distribution over the state, the proposal, and a slice variable with joint density

$$\pi(x, x^*, u) \propto \mathbb{I}[u < \gamma(x)q(x, x^*)] \tag{1.5}$$

There are several examples in the literature demonstrating cases where slice variables ameliorate the difficulty of establishing detailed balance. Some of these include extra-chance Hamiltonian Monte Carlo (Campos and Sanz-Serna 2015, Park and Atchadé 2019), the No-U-Turn sampler (Hoffman and Gelman 2014), the "crumbs" method of Neal (2003), and the Swendsen-Wang algorithm (Swendsen and Wang 1987;Edwards and Sokal 1988;[Section 7.2]Liu 2008).

Algorithm 2 A single step of Metropolis-Hastings, using a deterministic accept-reject decision.

Given unnormalized density $\gamma(x)$, proposal distribution $q(x, \cdot)$, and current state x. Returns the next state of the Markov chain.

Sample $x^* \sim q(x, \cdot)$.
Sample $u \sim \text{Uniform}(0, \gamma(x)q(x, x^*))$.
if $u < \gamma(x^*)q(x^*, x)$ **then**
  return x^*.
else
  return x.
end if

1.2.8 Non-reversibility, lifting, and skew balance

A *reversible* Markov chain is one in which

$$\pi(x)K(x, x') = \pi(x')K(x', x)$$

is satisfied; at stationarity the chain is indistinguishable from its time-reversal. We shall consider a class of *non-reversible* chains known as *lifted* chains (Diaconis et al. 2000). These are chains which use an auxiliary variable, here denoted $s \in S$, often interpreted as a sort of "direction." Here we assume that there exists a "reversal" involution $\mathcal{S} : S \mapsto S$, $\mathcal{S} = \mathcal{S}^{-1}$, which is also measure preserving, so $\pi(x, s) = \pi(x, \mathcal{S}(s))$. The algorithm employs a cycle of two reversible kernels:

1. a typical Markov chain Monte Carlo kernel e.g. sampling a proposal $q_s(x, x')\delta_{\mathcal{S}(s)}(s')$ (i.e. the proposal applies the reversal, $s' = \mathcal{S}(s)$) and accepting this with the Metropolis-Hastings acceptance probability. If this proposal is rejected the chain remains in the state (x, s) as is usual.

2. a reversal of s, i.e. a proposal $\delta_{\mathcal{S}(s)}(s')$ where such a proposal is always accepted as $\mathcal{S}$ is measure-preserving.

The cycle of these two kernels either results in the state (x', s), in the case that the Metropolis-Hastings proposal is accepted, or $(x, \mathcal{S}(s))$, if the Metropolis-Hastings proposal is rejected.

While the cycle described here can be seen as an application of various constructions described in this chapter, one can fairly simply derive conditions on a

kernel which allow one to describe the above as a single kernel. We use the prefix "skew" to describe those constructions, after Turitsyn et al. (2011). In Chapter 2 we discuss this at length and demonstrate the equivalence between deriving reversible and skew-reversible kernels.

1.2.9 Hamiltonian Monte Carlo

Hamiltonian Monte Carlo takes its inspiration from dynamical systems to construct a simulation of a physical system using Hamiltonian dynamics. This method applies only when $x \in \mathbb{R}^d$. A *momentum* variable $p \in \mathbb{R}^d$ is introduced, giving a joint density $\rho(x,p) = \pi(x)\psi(p) \propto \exp(-H(x,p))$ where $H(x,p)$ can be interpreted as the total energy in the system,

$$H(x,p) = U(x) + \frac{1}{2}p^T M^{-1} p$$

and $U(x) = -\log \pi(x)$ is the potential energy of the system. The Hamiltonian dynamics for such a system are

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}x &= \nabla_p H(x,p) = M^{-1}p \\ \frac{\mathrm{d}}{\mathrm{d}t}p &= -\nabla_x H(x,p) = -\nabla U(x).\end{aligned}$$

The idea is to construct Markov chain Monte Carlo proposals by simulating these dynamics for a fixed time, yielding a new point (x', p'). These dynamics preserve the energy $H(x,p)$, suggesting that a Markov chain Monte Carlo algorithm using exact simulations of these dynamics could be rejection free.

As the Hamiltonian dynamics can be solved only in specific situations, proposals are instead constructed by simulating a discretization of the dynamics. The most widely used discretization is the *leapfrog* scheme, where one step of the leapfrog takes

$$\begin{aligned}x_{t+1/2} &= x_t + \frac{\epsilon}{2}p_t \\ p_{t+1} &= p_t - \epsilon\nabla U(x_{t+1/2}) \\ x_{t+1} &= x_t + \frac{\epsilon}{2}p_{t+1}.\end{aligned}$$

Typically the proposal is constructed from running some number L steps of the leapfrog. In order that the proposal be a deterministic function which is its own inverse, we take for the proposal $x_L, -p_L$.

This method of incorporating the proposal as an auxiliary variable is particularly advantageous here, as it allows for a simple construction of an acceptance ratio. Noting that the transformation due to the leapfrog is its own inverse (including the final reversal of momentum) and also volume-preserving, we apply the acceptance probability for such a deterministic transform (1.3) to yield

$$\alpha((x,p),(x_L,-p_L)) = \exp\left(-[H(x_L,-p_L) - H(x,p)]_+\right).$$

2

Piecewise-deterministic Markov chain Monte Carlo in continuous time

The algorithms described in this chapter will simulate continuous-time *paths* which can be used to compute expectations, in an analogous way to the samples produced by a discrete-time Markov chain Monte Carlo algorithm. Here, a path $\{z_t \in \mathcal{Z}; 0 \leq t \leq T\}$ in some state space $\mathcal{Z}$ will be simulated to target a density ρ. Under ergodicity assumptions, the estimate

$$\frac{1}{T}\int_0^T f(z_t)\mathrm{d}t \approx \mathbb{E}_\rho[f(Z)] \tag{2.1}$$

is a consistent estimate of the expectation $\mathbb{E}_\rho[f(Z)]$.

While various types of processes might be suitable for this purpose, we focus on a particular class known as *piecewise-deterministic Markov processes*. First introduced by Davis (1984), these describe a càdlàg process which follows a deterministic trajectory between random jumps at random times. We will see that this class of processes can be adapted to simulate from a large class of target distributions in such a way that the evolution of the process can be simulated exactly.

A variety of Markov chain Monte Carlo algorithms based on piecewise-deterministic Markov processes have been proposed in the literature; we review these throughout this chapter. Our aim is to describe and extend the framework that subsumes these algorithms to allow for the development of novel methods.

2.1 Piecewise-deterministic Markov processes and Markov chain Monte Carlo

The following is an informal presentation of some key concepts relating to the simulation and properties of a continuous-time process. For a more formal treatment, see the work of Davis (1984).

We consider only real-valued state spaces of fixed dimension, so $\mathcal{Z} = \mathbb{R}^n$. A piecewise-deterministic Markov process is defined by three aspects:

1. an ordinary differential equation with differentiable drift $\phi : \mathcal{Z} \mapsto \mathcal{Z}$, i.e.,

$$\frac{\mathrm{d}z_t}{\mathrm{d}t} = \phi(z_t),$$

   which induces a deterministic flow

$$(t, z) \in \mathbb{R}_{\geq 0} \times \mathcal{Z} \mapsto \Phi_t(z) \in \mathcal{Z}$$

   satisfying the semi-group property $\Phi_s \circ \Phi_t = \Phi_{s+t}$ and such that $t \mapsto \Phi_t(z)$ is càdlàg;

2. an event rate $\lambda : \mathcal{Z} \to \mathbb{R}_{\geq 0}$, with $\lambda(z_t)\epsilon + o(\epsilon)$ being the probability of having an event in the time interval $[t, t + \epsilon]$; and

3. a Markov transition kernel Q from $\mathcal{Z}$ to $\mathcal{Z}$ where the state at event time t is given by $z_t \sim Q(z_{t^-}, \cdot)$, z_{t^-} being the state of the process just before the event.

In Algorithm 3 we give a general framework providing one approach to simulate exactly the path of a piecewise-deterministic Markov process.

Algorithm 3 Simulation of continuous-time piecewise-deterministic Markov process

$t_0 \leftarrow 0$.
Initialize z_0 arbitrarily on $\mathcal{Z}$.
for $k \in \{1, 2, \dots\}$ **do**
Sample inter-event time $\tau_k \in [0, \infty)$, where τ_k has distribution such that

$$\mathbb{P}(\tau_k \geq t) = \exp\left[-\int_{r=0}^{t} \lambda\left\{\Phi_r(z_{t_{k-1}})\right\} \mathrm{d}r\right]. \tag{2.2}$$

Set $z_{t_{k-1}+r} \leftarrow \Phi_r(z_{t_{k-1}})$ for $r \in (0, \tau_k)$.
$t_k \leftarrow t_{k-1} + \tau_k$.
Sample $z_{t_k} \sim Q(z_{t_k^-}, \cdot)$.
end for

To simulate a piecewise-deterministic Markov process, we need to be able to

1. simulate inter-event times τ_k from the distribution (2.2),

2. compute the flow Φ, and

3. simulate from the transition kernel Q.

In the next section we present conditions on the design choices λ, Φ and Q which are necessary for the piecewise-deterministic Markov process to leave invariant a chosen distribution ρ.

2.2 The invariant distribution of a piecewise-deterministic Markov process

Our eventual goal is to select the parameters of our piecewise-deterministic Markov process - namely, the event rate λ, the flow Φ, and the transition kernel Q - in such a way that the process produces samples from a *target distribution* ρ, defined on the Borel space $(\mathcal{Z}, \mathcal{B}(\mathcal{Z}))$. We will call such a piecewise-deterministic Markov process a *piecewise-deterministic Markov chain Monte Carlo algorithm*, as — much like a typical discrete-time Markov chain Monte Carlo algorithm — we have designed the process to compute expectations with respect to ρ.

If we want to design a piecewise-deterministic Markov process to sample this target distribution, we require that it leave the distribution ρ invariant. We provide

here sufficient conditions to ensure this is satisfied. If the piecewise-deterministic Markov process is also ergodic, this will allow us to consistently estimate expectations with respect to the invariant distribution as in (2.1).

A common method to establish the invariant distribution of a continuous-time process is to identify the *infinitesimal generator* $\mathcal{L}$ of the process and show that

$$\int \rho(\mathrm{d}z)\mathcal{L}f(z) = 0 \tag{2.3}$$

holds for all $f \in D$ where D is a core of the generator (Ethier and Kurtz 2005; Proposition 9.2).

The generator is defined by

$$\mathcal{L}f(z) = \lim_{t \downarrow 0_+} \frac{\mathbb{E}[f(Z_t)|Z_0 = z] - f(z)}{t}.$$

The generator of a piecewise-deterministic Markov process was established by Davis (1984; Theorem 26.14) as

$$\mathcal{L}f(z) = \langle \phi(z), \nabla f(z) \rangle + \lambda(z) \int [f(z') - f(z)]\, Q(z, \mathrm{d}z'), \tag{2.4}$$

where $\langle a, b \rangle$ denotes the scalar product between vectors a and b. The first term on the right hand side of (2.4) arises from the deterministic flow while the second term corresponds to the jump component of the process.

Replacing $\mathcal{L}$ in (2.3) with the expression for the generator of a piecewise-deterministic Markov process (2.4), we arrive at the condition

$$\int \rho(\mathrm{d}z)\langle \phi(z), \nabla f(z) \rangle + \int \rho(\mathrm{d}z)\lambda(z) \int Q(z, \mathrm{d}z')\, [f(z') - f(z)] = 0.$$

From now on, the target distribution will be assumed to have a strictly positive density $\rho(z)$ with respect to the Lebesgue measure $\mathrm{Leb}(\mathrm{d}z)$ (and $\mathcal{Z} = \mathbb{R}^d$). The density is related to the potential function $H(z)$ by

$$\rho(z) = \exp(-H(z)).$$

Using integration by parts, we may rewrite the first term using the gradient of H, giving

$$\int \rho(\mathrm{d}z)\langle \phi(z), \nabla f(z) \rangle = -\int \rho(\mathrm{d}z)\, \{\nabla \cdot \phi(z) - \langle \nabla H(z), \phi(z) \rangle\}\, f(z)$$

where $\nabla \cdot \phi(z) := \sum_{i=1}^{n} \partial_i \phi_i(z)$ is the divergence of the vector field ϕ. Thus we arrive at a sufficient condition to ensure ρ-invariance of a piecewise-deterministic Markov process in terms of $H(z)$:

$$\int \rho(\mathrm{d}z) \left[\lambda(z) \int Q(z, \mathrm{d}z') \left[f(z') - f(z) \right] - \left\{ \nabla \cdot \phi(z) - \langle \nabla H(z), \phi(z) \rangle \right\} f(z) \right] = 0. \tag{2.5}$$

We introduce some notation which will be useful for denoting probability measures under transformation. For a measure ν on $(\mathcal{Z}, \mathcal{B}(\mathcal{Z}))$ and a measurable mapping $g : \mathcal{Z} \mapsto \mathcal{Z}$, the *push-forward measure* of the ν under the mapping f, typically denoted $f_{\#}(\nu)$, is the measure $A \in \mathcal{B}(\mathcal{Z}) \mapsto \nu(g^{-1}(A))$. We will use here the notation $\nu(g^{-1}(\mathrm{d}z))$, where for any measurable $h : \mathcal{Z} \mapsto \mathbb{R}$, we have

$$\int_{\mathcal{Z}} h \circ g(z) \nu(\mathrm{d}z) = \int_{\mathcal{Z}} h(z) (g_{\#}(\nu))(\mathrm{d}z) = \int_{\mathcal{Z}} h(z) \nu(g^{-1}(\mathrm{d}z)).$$

From (2.5), we arrive at conditions on ρ, λ, and Q to ensure ρ-invariance of the associated piecewise-deterministic Markov process:

(**A1**) Conditions on ρ, λ, and Q:

1. there exists a ρ-preserving mapping $\mathcal{S} : \mathcal{Z} \to \mathcal{Z}$ that is measurable and satisfies $\rho(\mathcal{S}^{-1}(\mathrm{d}z)) = \rho(\mathrm{d}z)$.

2. The event rate λ satisfies

$$\lambda(\mathcal{S}(z)) - \lambda(z) = \nabla \cdot \phi(z) - \langle \nabla H(z), \phi(z) \rangle. \tag{2.6}$$

3. The kernel Q satisfies

$$\int \rho(\mathrm{d}z) \lambda(z) Q(z, \mathrm{d}z') = \rho(\mathcal{S}^{-1}(\mathrm{d}z')) \lambda(\mathcal{S}(z')). \tag{2.7}$$

Proposition 2.1. *Assume A1. Then the piecewise-deterministic Markov process admits ρ as an invariant distribution.*

Proof of Proposition 2.1. Using Assumption A1.3 then Assumption A1.1 we obtain

$$\begin{aligned}\iint \rho(\mathrm{d}z)\lambda(z)Q(z,\mathrm{d}z')f(z') &= \int \rho(\mathcal{S}^{-1}(\mathrm{d}z'))\lambda(\mathcal{S}(z'))f(z') \\ &= \int \rho(\mathrm{d}z')\lambda(\mathcal{S}(z'))f(z').\end{aligned}$$

Hence, (2.5) is equal to

$$\begin{aligned}&\int \rho(\mathrm{d}z)\left[\lambda(z)\int Q(z,\mathrm{d}z')[f(z')-f(z)] - \{\nabla\cdot\phi(z) - \langle\nabla H(z),\phi(z)\rangle\}f(z)\right] \\ =&\int \rho(\mathrm{d}z)\left[\{\lambda(\mathcal{S}(z))-\lambda(z)\} - \{\nabla\cdot\phi(z) - \langle\nabla H(z),\phi(z)\rangle\}\right]f(z) = 0\end{aligned}$$

using Assumption A1.2. This establishes the result. □

We also note that the rate for a divergence-free flow can also be written as

$$\langle\nabla H(z),\phi(z)\rangle = \left[\frac{\mathrm{d}}{\mathrm{d}r}H(\Phi_r(z))|_{r=0}\right], \tag{2.8}$$

providing an alternative interpretation which may be useful in some settings.

While the method is valid for any choice of $\mathcal{S}$ satisfying A1.1, we suggest that finding a λ satisfying A1.2 will only be tractable when $\mathcal{S}$ is an involution. Henceforth we make the assumption that this is the case, so $\mathcal{S}^{-1} = \mathcal{S}$.

We further note that, at equilibrium, the distribution proportional to $\rho(z)\lambda(z)$ is the distribution of states just before the occurrence of an event, and $\int_{z\in\mathcal{Z}}\rho(\mathrm{d}z)\lambda(z)Q(z,\mathrm{d}z')$ is the distribution just after the event, the so called "jump" chain (Costa 1990).

2.2.1 Invariance conditions for decomposable rates

Here we extend the piecewise-deterministic Markov process to allow for multiple event "types." That is, we define a family of event rates indexed by $\omega\in\Omega$ denoted λ_ω; for each define an associated kernel Q_ω. Additionally define a measure μ on (Ω,Ω). Instead of considering a simple event time $t\in\mathbb{R}_+$, we consider event times associated with an ω, so our event will be denoted $(t,\omega)\in(\mathbb{R}_+,\Omega)$. The distribution of $t\in\mathbb{R}_+$ is characterized by

$$\mathbb{P}[\tau > t] = \exp\left(-\int_{r=0}^{t}\int_{\omega\in\Omega}\lambda_\omega(\Phi_r(z))\mu(\mathrm{d}\omega)\mathrm{Leb}(\mathrm{d}r)\right), \tag{2.9}$$

similarly to (2.2).

Simulating (t,ω) may be achieved in several ways. Two possible approaches are

1. to simulate t from (2.9) and then simulate ω from the distribution proportional to $\lambda_\omega(\Phi_t(z))\mu(\mathrm{d}\omega)$, or to

2. simulate the first arrival in time of an inhomogenous Poisson process with rate $\lambda_\omega(\Phi_t(z))\mu(\mathrm{d}\omega)\mathrm{Leb}(\mathrm{d}t)$.

This construction is particularly useful in those cases where the right-hand-side of the rate condition (2.6) can be expressed as an integral; here we can write

$$\nabla \cdot \phi(z) - \langle \nabla H(z), \phi(z) \rangle = \int_{\omega \in \Omega} h_\omega(z) \mu(\mathrm{d}\omega). \tag{2.10}$$

This encompasses settings where the gradient of H can be estimated unbiasedly, and also any setting where $H(z)$ can be factorized, i.e. written as a sum of terms.

(**A2**) Conditions on ϕ, λ_ω, and Q_ω:

1. There exists a ρ-preserving mapping $\mathcal{S} : \mathcal{Z} \to \mathcal{Z}$.

2. The event rates $\{\lambda_\omega : \omega \in \Omega\}$ satisfy

$$\int \mu(\mathrm{d}\omega) \{\lambda_\omega(\mathcal{S}(z)) - \lambda_\omega(z)\} = \nabla \cdot \phi(z) - \langle \nabla H(z), \phi(z) \rangle. \tag{2.11}$$

3. For all $\omega \in \Omega$, the transition kernel Q_ω satisfies

$$\int \rho(\mathrm{d}z) \lambda_\omega(z) Q_\omega(z, \mathrm{d}z') = \rho(\mathcal{S}^{-1}(\mathrm{d}z')) \lambda_\omega(\mathcal{S}(z')). \tag{2.12}$$

Proposition 2.2. *Assume A2. Then the piecewise-deterministic Markov process admits ρ as an invariant distribution.*

The proof is similar to that for 2.1.

2.2.2 For stochastic potentials

A case of particular interest is the setting where $H(z)$ is naturally expressed as an integral. Here we can write

$$H(z) = \int_{\omega \in \Omega} H_\omega(z) \mu(\mathrm{d}\omega).$$

where μ is a measure on some space (Ω, Ω) as before. This encompasses settings where H can be estimated unbiasedly (e.g. when we can write μ as a probability measure, by simulating $\omega \sim \mu$ and taking the estimate $H_\omega(z)$), and several models where H_ω is the realization of a process.

If the functions $\{H_\omega : \omega \in \Omega\}$ are differentiable then A2.2 is satisfied for a divergence-free vector field, i.e. $\nabla \cdot \phi = 0$, if for all ω

$$\lambda_\omega(\mathcal{S}(z)) - \lambda_\omega(z) = -\langle \nabla H_\omega(z), \phi(z) \rangle. \tag{2.13}$$

2.2.3 For factorizable H

A further specialized case is where

$$H(z) = \sum_{i \in [N]} H_i(z)$$

which can be considered a special case of the stochastic potential above where $\mu(\mathrm{d}\omega)$ is a counting measure. We refer to this as a *factorization* of the potential; this case will form the basis for most of our algorithms of interest, including those for large datasets (Chapter 4) and for sparse factor graphs (Chapter 5).

2.2.4 Previous work

Simulation of continuous-time piecewise-deterministic Markov processes has been studied extensively for several decades; the earliest examples of which we are aware apply these methods to the simulation of rarefied gasses (Haviland 1965). Those simulations found in physics literature typically aim to study macroscopic properties of a system by simulating the well-understood microscopic dynamics of that system and observing the emergent behaviour.

The methodology of designing of a piecewise-deterministic Markov process to target an explicit density seems to be a much more recent development. Peters et al. (2012) provide perhaps the first instance of such an algorithm, derived as the limit of a series of Metropolis-Hastings steps. They derive the Fokker-Planck equation for a linear flow and reflective event kernel, and demonstrate correctness of their algorithm when applied to potentials which can be decomposed as sums.

Mesquita and Hespanha (2012) describe the design of a continuous-time piecewise-deterministic Markov process controller which leaves invariant a distribution. They discuss a limited set of event kernels.

Another derivation is found in Michel et al. (2014), where a "factorized Metropolis filter" is shown to converge to the event rates of Peters et al. (2012). Denote the target energy by the potential U and the state as x; this will correspond to notation used in subsequent chapters. The factorized Metropolis filter is essentially a conjunction of Metropolis-Hastings probabilities for each term of an additive energy $U = \sum_i U_i$, i.e. the probability of accepting a move from state x to x^* is $\prod_i \min\{1, \exp(U_i(x) - U_i(x^*))\}$. We will revisit such ideas in Chapter 3, where we describe discrete-time versions of these continuous-time algorithms

A special case of the conditions A1 were presented by Fearnhead et al. (2018), where they were derived for a linear flow. The authors make use of the general condition on Q to propose some alternatives to the typical reflection operator, discussed further in Section 2.5.

Pakman et al. (2017) prove a special case of A2 for the special case that an unbiased estimate of the gradient ∇U is used. While this unbiased estimate is used to derive exact bounces, in their application they are only able to provide approximate control on the bounce rate, yielding an inexact algorithm.

Our contributions are to extend the invariance conditions to allow for arbitrary flows, whereas previously this has been done only for linear flows. Additionally, whereas the above make use of a decomposition of the potential to a finite sum, we have extended this to arbitrary integrals.

2.3 Simulating event times

The majority of this chapter is dedicated to discussing choices for the flow Φ and the event kernel Q. As for simulating from the inter-event time distribution (2.2), this has been studied in a few contexts where exact computation is not possible, see e.g. Bierkens et al. (2019a), Bouchard-Côté et al. (2018). We highlight the main approaches here.

Henceforth we consider a state space which can be decomposed into state and velocity components, denoting these by $z = (x, v)$ where $x \in \mathbb{R}^d$ and $v \in \mathbb{R}^d$. The energy over this state will be denoted $H(z) = U(x) + K(v)$ where U represents the potential energy of the state x and K the kinetic energy of the velocity. For the sake of illustration we restrict ourselves to the case of linear flow, i.e. where $\Phi_t(x, v) = (x + vt, v)$.

1. **Convex potentials.** When $\lambda(x, v) = \langle \nabla U(x), v \rangle_+$ (letting a_+ denote $\max(0, a)$) and $U(x)$ is convex, we can efficiently solve for the inverse of the integral as described in Algorithm 4. The algorithm can be seen as correct using a similar argument as that for the typical inversion method of simulating from a cumulative distribution function.

Algorithm 4 Simulating event times for convex potentials.

Given state x, v, potential U, simulate an event time with rate $\langle \nabla U(x + vt), v \rangle_+$.

function SIMULATECONVEXEVENTTIME(U)
  $t^* \leftarrow \text{argmin}_{t \geq 0} U(x + vt)$.
  Simulate $E \sim \text{Exp}(1)$.
  Solve for $\tau \geq 0$ in

$$U(x + v\tau) = U(x + vt^*) + E. \tag{2.14}$$

  return τ.
end function

Due to the convexity of U, the equation (2.14) can be solved efficiently.

2. **Exact solutions.** Available in a few important cases, e.g. when the distribution is normal. In the normal case, this is a special case of the convex case, but where the equation (2.14) can be solved analytically and without resorting to numerical optimization. Due to its importance, we give the full expression for the normal case: assuming the potential is $U(x) = \frac{1}{2} x^T \Lambda x$, i.e. the precision matrix is given by Λ, the event time is given by

$$\tau = \frac{-x^T \Lambda v + \sqrt{\max\{0, v^T \Lambda x\}^2 + 2 v^T \Lambda v E}}{v^T \Lambda v} \tag{2.15}$$

where $E \sim \text{Exp}(1)$.

3. **Poisson process thinning.** Noting that for a bounding rate $\bar{\lambda}(x, v) \geq \lambda(x, v)$, a Poisson process with rate $\lambda(x, v)$ can be recovered by *thinning* from a Poisson process with rate $\bar{\lambda}(x, v)$ as in Algorithm 5.

   Interpreting the simulation of τ as the first arrival of a Poisson process, we can easily see that this is the first time simulated from the process with bounding rate $\bar{\lambda}(x, v)$ that is retained after thinning. We refer to those times simulated from $\bar{\lambda}$ as *candidate times* as they are candidates to become actual event times in the thinning process.

   The bounds on the rate will be specific to the application. The performance of the algorithm will directly be affected by quality (i.e. tightness) of the bounding rate, as the number of operations performed by the algorithm will scale with the number of candidate events.

   An example of a more general approach is suggested by Bierkens et al. (2017): if the Hessian of the potential can be dominated uniformly in x by some matrix Q in positive-definite ordering, this may be exploited to derive a rate which is linear in t.

Algorithm 5 Simulating event times using a bounding rate.

Given rate λ and bounding rate $\bar{\lambda}$, simulate event time from λ.

```
function SimulateBoundedRate(λ, λ̄)
    t_0 ← 0.
    repeat
        Simulate t as the first arrival of an inhomogenous Poisson process on
{t : t > t_0} with rate λ̄(x, v).
        Simulate u ∼ Uniform(0, 1).
        t_0 ← t_0 + t.
    until u < λ(x + vt, v)/λ̄(x + vt, v)
    return t.
end function
```

In Algorithm 5, we see that the rejection of a candidate time t precludes the possibility of further considering any times inferior to t. As such, no events can

occur during this time and one may advance the state to the point $(x, x + vt)$ regardless of whether the event is accepted or not. This may provide some simplifications in implementation. An algorithm based on this interpretation is explicitly presented in Algorithm 6. This version is amenable to allowing the use of rates which are updated after the rejection of a candidate time, for example when the bounding rate $\bar{\lambda}$ is adapted based on the current state. Adapting Algorithm 6 for the factorized case can also be useful; there we can simulate a candidate event for each factor and then thin these in order until a candidate is accepted, possibly avoiding the need to thin many of the candidates.

Algorithm 6 One step of a piecewise-deterministic Markov chain Monte Carlo with thinned rates.

Given x_k, v_k, event rate λ, and bounding rate $\bar{\lambda}$. Return x_{k+1}, v_{k+1} and deterministic piece duration t_k.

function PDMCMC-THINNED(x_k, v_k)
  Simulate t as the first arrival of a inhomogenous Poisson process on $\{t : t > 0\}$ with rate $\bar{\lambda}(x_k, v_k)$.
  with probability $\lambda(x_k + v_k t, v_k)/\bar{\lambda}(x_k + v_k t, v_k)$
    Sample $(x_{k+1}, v_{k+1}) \sim Q(x_k + v_k t, v_k)$.
    return x_{k+1}, v_{k+1}, t.
  otherwise
    return $x_k + v_k t$, v_k, t.
end function

2.4 Some existing piecewise-deterministic Markov chain Monte Carlo algorithms

While the conditions A1, A2 are made to be fairly general, all the existing algorithms we are aware of fall within a more specific setting where a linear flow is used. The following specifications are made:

1. The distribution of interest is defined on $\mathbb{R}^d$ and admits a density with respect to Lebesgue measure on $\mathcal{X} = \mathbb{R}^d$ equal to $\pi(x) = \exp(-U(x))$. The algorithm will actually leave invariant an *extended target*: defining the state as $z = (x, v)$,

the extended target distribution $\rho(\mathrm{d}z)$ on $\mathcal{Z} = \mathcal{X} \times \mathcal{V}$ is then defined as

$$\rho(\mathrm{d}z) = \pi(\mathrm{d}x)\psi(\mathrm{d}v).$$

The extended variable v can be interpreted as a *velocity* (or sometimes as a *momentum*). The density ψ is an auxiliary distribution on $\mathcal{V}$, and $\mathcal{V}$ can be for example either $\mathbb{R}^d$ or the unit hypersphere $\mathbb{S}^{d-1}$.

2. The following linear drift is used:

$$\phi(z) = \begin{pmatrix} v \\ \mathbf{0} \end{pmatrix},$$

so the resulting flow is analytically tractable and given by

$$\Phi_t(z) = (x + vt, v). \tag{2.16}$$

In this case, we have $\nabla \cdot \phi = 0$.

3. The operator $\mathcal{S}$ is a reversal of velocity; that is $\mathcal{S}(x, v) = (x, -v)$ which can alternatively be viewed as a time reversal.

4. The event kernel Q only modifies the velocity; the position x is unchanged.

With these choices, we can give more specific conditions to yield a ρ-invariant algorithm. The condition A1.2 becomes

$$\lambda(x, -v) - \lambda(x, v) = -\langle \nabla U(x), v \rangle. \tag{2.17}$$

As the event kernel now only modifies the velocity, we write this velocity-only kernel as Q_x, so $Q(x, v, \mathrm{d}x', \mathrm{d}v') = \delta_x(\mathrm{d}x')Q_x(v, \mathrm{d}v')$, and Q_x must satisfy

$$\int \psi(v)\lambda(x, v)Q_x(v, \mathrm{d}v') = \psi(-v')\lambda(x, -v). \tag{2.18}$$

2.4.1 The bouncy particle sampler

Originally proposed by Peters et al. (2012), the distinguishing feature of the bouncy particle sampler and its variants is the use of a deterministic reflection operator for the kernel Q_x. The reflection is in the plane normal to the potential gradient $\nabla U(x)$; specifically, the velocity takes on the new value $R(v; \nabla U(x))$ where

$$R(v; \nabla U(x)) := v - 2\frac{\langle \nabla U(x), v\rangle}{\|\nabla U(x)\|_2^2}\nabla U(x). \tag{2.19}$$

A complication of this particular kernel is that, alone, it does not guarantee that the process is ergodic; Bouchard-Côté et al. (2018) demonstrate that this pathology occurs for an isotropic Gaussian. To rectify this, a *refreshment* event is added. This refreshment event occurs with rate λ_{ref} independent of the state, and when it occurs the velocity is resampled independently of its current value.

The overall rate is thus defined

$$\lambda(x, v) = \lambda_{\text{ref}} + \langle \nabla U(x), v\rangle_+$$

and the kernel Q_x can be written

$$Q_x(v, \mathrm{d}v') = \frac{\lambda_{\text{ref}}}{\lambda}\psi(\mathrm{d}v') + \frac{\langle \nabla U(x), v\rangle_+}{\lambda}\delta_{R(v;\nabla U(x))}(\mathrm{d}v'). \tag{2.20}$$

This kernel is interpreted as a mixture of the refreshment and bounce kernels, and may be implemented by simulating both the refreshment and bounce times, then selecting the event which occurs first and simulating the corresponding event kernel.

2.4.1.1 The factorized bouncy particle sampler

The algorithm proposed in (Peters et al. 2012) additionally exploits a decomposition of the potential U, i.e. when the potential can be rewritten

$$U(x) = \sum_{i=1}^{m} U_i(x) \tag{2.21}$$

which as a factorization of the potential $U(x)$ corresponds to the rate of Section 2.2.3. An event rate is simulated for each factor, that is,

$$\lambda(x, v) = \lambda_{\text{ref}} + \sum_{i=1}^{m}\langle \nabla U_i(x), v\rangle_+ \tag{2.22}$$

representing $m+1$ independent events. The corresponding kernel is

$$Q_x(v, \mathrm{d}v') = \frac{\lambda_{\mathrm{ref}}}{\lambda(x,v)}\psi(\mathrm{d}v') + \sum_{i=1}^{m} \frac{\langle \nabla U_i(x), v\rangle_+}{\lambda(x,v)} \delta_{R(v;\nabla U_i(x))}(\mathrm{d}v'). \tag{2.23}$$

We can confirm that the algorithm satisfies conditions to leave ρ invariant, as this corresponds to the factorizable case given in Section 2.2.3, that is $\Omega = [m+1]$ and

$$\begin{aligned}
\langle \nabla H(z), \phi(z)\rangle &= \sum_{i=1}^{m} \langle U_i(x), v\rangle \\
\lambda_i(z) &= \begin{cases} \langle U_i(x), v\rangle_+ & i \in [m] \\ \lambda_{\mathrm{ref}} & i = m+1 \end{cases} \\
Q_i(x, v, \mathrm{d}x', \mathrm{d}v') &= \delta_x(\mathrm{d}x') \begin{cases} \delta_{R(v;\nabla U_i(x))}(\mathrm{d}v') & i \in [m] \\ \psi(\mathrm{d}v') & i = m+1 \end{cases}
\end{aligned}$$

For $m = 1$, this algorithm reduces to the standard bouncy particle sampler.

We call the combination of this rate (2.22) and kernel (2.23) the *factorized bouncy particle sampler*, which represents a generalization of that given in (2.20). As shown in Bouchard-Côté et al. (2018), where it was called the *local bouncy particle sampler*, this factorization is useful in settings where the factors $U_i(x)$ depend on only a few coordinates of x, for example, in time series or spatial models where the coordinates of x_i are organized in a lattice and interaction between variables is limited to neighbouring sites. In such settings, the velocity modification resulting from a bounce affects only a small neighbourhood of factors, and as such the computational cost of a bounce is small and scales more efficiently with dimension. We explore such models in detail in Chapter 5.

2.4.2 The zig-zag algorithm

The zig-zag algorithm proposed in Bierkens et al. (2019a; 2017) is distinguished by the use of an independent event rate for each coordinate, that is, the event rates for $i \in [d]$ are

$$\lambda_i(x, v) = \lambda_{\mathrm{ref},i} + [v_i \partial_i U(x)]_+,$$

which can be understood to be an instance of (2.10) where

$$\langle \nabla H(z), \phi(z)\rangle = \sum_{i=1}^{d} \underbrace{v_i \partial_i U(x)}_{h_\omega(z)}.$$

and thus $\Omega = [d]$. The corresponding event transition kernels amount to a negation of that coordinate's velocity:

$$Q_i(x, v, \mathrm{d}x', \mathrm{d}v') = \delta_x(\mathrm{d}x')\delta_{-v_i}(\mathrm{d}v_i') \prod_{j \neq i} \delta_{v_j}(\mathrm{d}v_j').$$

The overall event rate and event transition kernel are thus expressed as

$$\lambda(x, v) = \sum_{i=1}^{d} [\lambda_{\mathrm{ref},i} + [v_i \partial U_i(x)]_+]$$

$$Q(x, v, \mathrm{d}x', \mathrm{d}v') = \delta_x(\mathrm{d}x') \sum_{i=1}^{d} \frac{\lambda_{\mathrm{ref},i} + [v_i \partial U_i(x)]_+}{\lambda(x, v)} \delta_{-v_i}(\mathrm{d}v_i') \prod_{j \neq i} \delta_{v_j}(\mathrm{d}v_j').$$

Since the velocity is only ever modified by the reversal of a single coordinate, the zig-zag algorithm uses the uniform distribution on $\{-1, 1\}^d$ for ψ. In this scenario, $\rho(\mathrm{d}z)$ does not admit a density with respect to Lebesgue measure but the invariance conditions discussed previously still hold.

The refreshment rates $\lambda_{\mathrm{ref},i}$ may be chosen arbitrarily in $[0, \infty)$; however it can be shown that, unlike the bouncy particle sampler, the algorithm remains ergodic even if $\lambda_{\mathrm{ref},i} = 0$ for all i (Bierkens et al. 2019b).

Much like the bouncy particle sampler, it is possible to use factorizations of $U(x)$. This was exploited by Bierkens et al. (2019a) for the big data setting. See Chapter 4 for further discussion.

2.4.3 Billiard Hamiltonian Monte Carlo

Neal (2011) notes an interesting variant of Hamiltonian Monte Carlo which can be derived using the following heuristic argument. Taking $H(x, p) = U(x) + K(p)$ with

$$K(p) = \begin{cases} 0 & p \in B \\ \infty & p \notin B \end{cases}$$

where B is a ball of radius 1 centered at the origin, the time derivative of $\frac{\mathrm{d}}{\mathrm{d}t}x = \nabla_p H(x, p)$ is zero when p is inside the ball but infinite at the boundary. p moves at a rate of $v_p - \nabla_x H(x, p) = -\nabla U(x)$, when this reaches the boundary the "infinite speed" of x is manifested as a jump in x along the line $\{x + sp : s > 0\}$ to the first point for which $U(x) = U(x + sp)$.

Regardless of the formulation of these dynamics, we see that the resulting algorithm has the following properties:

Algorithm 7 A single iteration of Billiard Hamiltonian Monte Carlo

Given state x, momentum p such that $\|p\|_2 = 1$, and maximum simulation time $t_{\max}$. Returns a dwell time t, indicating how much time is spent in state x, and a new state x' and new momentum p'.

function BILLIARDHMC(x, p, $t_{\max}$)
  $v_p \leftarrow -\nabla U(x)$.
  Find dwell time $t > 0$ such that $\|p + tv_p\|_2 = 1$.
  if $t > t_{\max}$ **then**
    return $t_{\max}$, x, $p + tv_p$.
  end if
  $p' \leftarrow p + tv_p$.
  Find $s > 0$ such that $U(x + sp') = U(x)$.
  $x' \leftarrow x + sp'$.
  return t, x', p'.
end function

1. the distance p must cross to traverse the unit ball with velocity $-\nabla U(x)$ is $2\frac{\langle \nabla U(x), p_0 \rangle}{\|\nabla U(x)\|_2}$, where p_0 is the momentum on the unit sphere when first entering the state x. Thus the dwell time in state x while p traverses the ball is

$$2\frac{\langle \nabla U(x), p_0 \rangle}{\|\nabla U(x)\|_2^2}$$

2. when p again reaches the unit sphere, this will be at the point

$$p' = p_0 - 2\frac{\langle \nabla U(x), p_0 \rangle}{\|\nabla U(x)\|_2^2}\nabla U(x)$$

(which is simply the velocity times the duration) which we recognize as the reflection of the momentum off the plane tangent to $\nabla U(x)$.

The update from x to $x + sp$, where $s > 0$ and such that $U(x) = U(x + sp)$, leaves invariant the distribution proportional to $\pi(x)\langle \nabla U(x), p \rangle$; see Section 2.8 for this proof. Thus the evolution of the chain on (x, p) is the same as that of the bouncy particle sampler but restricted to the jump chain. To see that this is a valid sampling scheme, we show that the dwell times provide the correct weighting to recover samples from $\pi(x)$ given samples from $\pi(x)\langle \nabla U(x), v \rangle$.

For a given x, the invariant density multiplied by the weight (i.e. the dwell time) should recover the correctly weighted distribution $\pi(x)$:

$$\pi(x)\langle \nabla U(x), p \rangle \psi(p) \times 2\frac{\langle \nabla U(x), p \rangle}{\|\nabla U(x)\|_2^2} \propto \pi(x)\psi(p)\frac{\langle \nabla U(x), p \rangle^2}{\|\nabla U(x)\|_2^2}$$

and marginally in x

$$\pi(x) \int \psi(p) \frac{\langle \nabla U(x), p \rangle^2}{\|\nabla U(x)\|_2^2} \mathrm{d}p \propto \pi(x)$$

noting that $\psi(p)$ (the uniform density on the ball) is isotropic.

Thus we can interpret the billiard Hamiltonian Monte Carlo algorithm as an Markov chain Monte Carlo chain targeting the jump chain $\pi(x)\langle \nabla U(x), p \rangle_+$ corrected by an importance weighting scheme. We note that Algorithm 7 alone is not sufficient for ergodicity and as such Neal (2011) suggests combining the above with other types of transitions.

2.4.4 Billiards for Bayes point machines

(Herbrich et al. 2001; Section 3.1) describe a continuous-time algorithm which samples from a hypersphere subject to linear constraints. The trajectories follow the hypersphere and reflect off of the hyperplanes which define the constraints. In the sense that this algorithm implements a flow targeting a particular geometry and uses reflections to accommodate linear constraints it is similar to the exact Hamiltonian Monte Carlo algorithm of Pakman and Paninski (2014) which is discussed further in Section 2.7.3.

2.4.5 The reflective slice sampler

The reflective slice sampler was introduced in Neal (2003) as a means to sample from multivariate slices. There, it was motivated as the application of Hamiltonian dynamics to the conditional slice density (as described in Section 1.2.7)

$$\pi(x|u) \propto \mathbb{I}(\gamma(x) > u),$$

for which the gradient $\nabla_x \gamma(x|u)$ is zero, yielding Hamiltonian trajectories which are exactly linear. Such linear trajectories will eventually reach the boundary of the slice (where $\gamma(x) = u$); this is dealt with in the reflective slice sampler by introducing reflections identical to those of (2.19); the trajectory is reflected off the plane tangent to the boundary at the intersection point, i.e. with normal vector $\nabla U(x)$ where x is the intersection point. See Algorithm 8 for a description of one step of this algorithm.

Thus the reflective slice sampler is — at least superficially — very similar to the bouncy particle sampler. An important distinction is that the reflective slice sampler is not actually a continuous-time sampler; instead, the Hamiltonian dynamics are simulated for a fixed and finite time to yield a new point in a rejection-free way. Another distinction is that the reflective slice sampler operates on the slice conditional $\pi(x|u)$, whereas the bouncy particle sampler does not require a slice variable.

Nonetheless, the parallels between the bouncy particle sampler and the reflective slice sampler are many. In particular, the discrete-time version proposed by Neal (2003) will be relevant in Chapter 3 where we seek to develop discrete-time versions of the continuous-time algorithms of this chapter.

Algorithm 8 Reflective slice sampling

Given current state x and simulation time τ, return next state x'.

function ReflectiveSliceSampler(x, τ)
  Sample $u \sim \text{Uniform}(0, \gamma(x))$.
  Sample $p \sim \mathcal{N}(0, I)$.
  while $\tau > 0$ **do**
    Find next boundary intersection time, $t^* \leftarrow \inf\{t > 0 : \gamma(x + tp) = u\}$.
    if $t^* < \tau$ **then**
      $x \leftarrow x + t^* p$.
      $p \leftarrow R(p; \nabla U(x))$.
    else
      $x \leftarrow x + \tau p$.
    end if
    $\tau \leftarrow \tau - t^*$.
  end while
  return x.
end function

2.5 Generalizing the event kernel: randomized bounces

Thus far we have only considered deterministic event kernels. Several randomized alternatives have been suggested in the literature, which we review here. We describe connections to the rarefied gas literature in which randomized bounces

have been widely employed, and show how these bounces may all be understood within a unified framework.

These alternatives are applicable to the case where $z = (x, v) \in \mathbb{R}^d \times \mathbb{R}^d$ and where the distribution over v is either Gaussian or uniform on the sphere. All leave the position x unchanged, taking the form

$$Q(x, v, \mathrm{d}x', \mathrm{d}v') = \delta_x(\mathrm{d}x')Q_x(v, \mathrm{d}v') \tag{2.24}$$

as in several of the examples of Section 2.4. As previously described, these kernels must obey the invariance condition A1.3.

Here ψ will be the standard multivariate normal distribution. We consider the scenario where $\lambda(x, v) = \langle \nabla U(x), v \rangle_+$ as in the bouncy particle sampler. In this case, we find that the above condition has been well-studied in the context of rarefied gas simulation, where the above equation arises when considering the effect of a molecule colliding with a surface.

2.5.1 The event kernel as a lifted Markov chain

We note that the condition A1.3 resembles a global balance condition for a Markov chain; were it not for the presence of the involution operator $\mathcal{S}$ on the right-hand side this would indeed be a typical global balance condition.

Here we demonstrate that the transition kernel Q may be cast in the framework of *lifted* Markov chains, a well-known class of non-reversible algorithms which we presented in Section 1.2.8. This interpretation will be useful for the exposition of new algorithms and also for establishing their correctness.

Consider the typical bounce kernel, where one takes as a new velocity $v_{t_k} = R(v_{t_k^-}; \nabla U(x_{t_k^-}))$. This procedure is a deterministic move from state $(x_{t_k^-}, v_{t_k^-})$ to state $(x_{t_k}, v_{t_k}) = (x_{t_k^-}, R(v_{t_k^-}; \nabla U(x_{t_k^-}))$, which is clearly non-reversible as the bounce rate in the new state is zero. In the lifted framework we interpret this move as a composition of two reversible kernels:

1. the first kernel assigns $v' \leftarrow -R(v_{t_k^-}; \nabla U(x_{t_k^-}))$; and

2. the second kernel assigns $v_{t_k} \leftarrow -v'$.

Here the involution $\mathcal{S}$ is taken as $\mathcal{S}(x, v) = (x, -v)$. Note that the bouncy particle sampler bounce rate is unchanged by the first step, as $\langle \nabla U(x), v \rangle_+ = \langle \nabla U(x), -R(v; \nabla U(x)) \rangle_+$ whereas as the projection on $\nabla U(x)$ is reversed by the combination of the two steps.

We present the following simple proposition showing that the above interpretation as a composition of two reversible kernels holds in the general case. That is to say, for any Q satisfying the condition A1.3, we can describe an equivalent kernel Q' satisfying a standard global balance condition

$$\int \rho(\mathrm{d}z)\lambda(z)Q'(z, \mathrm{d}z') = \rho(\mathrm{d}z')\lambda(z').$$

See Proposition 2.3 below. We call the condition on Q the *skew global balance*, after the terminology of Turitsyn et al. (2011).

Proposition 2.3. *Skew global balance.*

Given some positive density $\gamma : \mathcal{Z} \mapsto \mathbb{R}_{\geq 0}$ and an involution $\mathcal{S} : \mathcal{Z} \mapsto \mathcal{Z}$, a kernel Q satisfies

$$\int_{z \in \mathcal{Z}} \gamma(\mathrm{d}z)Q(z, \mathrm{d}z') = \gamma(\mathcal{S}(\mathrm{d}z')) \tag{2.25}$$

iff $Q'(z, z') = \int_{z''} Q(z, \mathrm{d}z'')\delta_{\mathcal{S}(z'')}(\mathrm{d}z')$ satisfies

$$\int_{z \in \mathcal{Z}} \gamma(\mathrm{d}z)Q'(z, \mathrm{d}z') = \gamma(\mathrm{d}z'). \tag{2.26}$$

Proof. First, we establish that Q and Q' can be constructed by simulating from one then applying the involution $\mathcal{S}$ to the result. Q' is the application of $\mathcal{S}$ to the result of Q by definition, and Q is thus recovered by two applications of the involution:

$$\int_{z'' \in \mathcal{Z}} Q'(z, z'')\delta_{\mathcal{S}z''}(\mathrm{d}z') = \int_{z'' \in \mathcal{Z}} \int_{z''' \in \mathcal{Z}} Q(z, z''')\delta_{\mathcal{S}(z''')}(z'')\delta_{\mathcal{S}(z'')}(\mathrm{d}z') = Q(z, z').$$

Now we show that if Q satisfies (2.25) then Q' satisfies (2.26):

$$\begin{aligned}
\int_{z \in \mathcal{Z}} \gamma(\mathrm{d}z)Q'(z, \mathrm{d}z') &= \int_{z \in \mathcal{Z}} \gamma(\mathrm{d}z) \int_{z'' \in \mathcal{Z}} Q(z, \mathrm{d}z'')\delta_{\mathcal{S}(z'')}(\mathrm{d}z') \\
&= \int_{z'' \in \mathcal{Z}} \gamma(\mathcal{S}(z''))\delta_{\mathcal{S}(z'')}(\mathrm{d}z') \\
&= \gamma(z').
\end{aligned}$$

□

Proposition 2.4. *Skew detailed balance.*

If Q satisfies skew detailed balance*:*

$$\gamma(\mathrm{d}z)Q(z,\mathrm{d}z') = \gamma(\mathcal{S}(\mathrm{d}z'))Q(\mathcal{S}(z'),\mathcal{S}(\mathrm{d}z)) \tag{2.27}$$

then Q satisfies (2.25) *and Q' satisfies detailed balance for γ.*

Proof. If we have (2.27) then

$$\int_{z\in\mathcal{Z}} \gamma(\mathrm{d}z)Q(z,\mathrm{d}z') = \gamma(\mathcal{S}(\mathrm{d}z')) \int_{z\in\mathcal{Z}} Q(\mathcal{S}(z'),\mathcal{S}(\mathrm{d}z)) = \gamma(\mathcal{S}(\mathrm{d}z')).$$

showing that Q also satisfies (2.25). □

Proposition 2.4 demonstrates that deriving Q' can be as simple as applying any typical Markov chain Monte Carlo kernel, for example, a Metropolis-Hastings or Gibbs algorithm targeting γ, which in the case of the event kernel is $\rho(z)\lambda(z)$. Once Q' has been developed we simply apply the involution to the resulting state. When using Metropolis-Hastings-based updates, the rejection of a move leaves the state unchanged, yet the second kernel setting $z' \leftarrow \mathcal{S}(z)$ is nonetheless accepted. Thus the net effect of a rejection is the application of $\mathcal{S}(z)$. See Figure 2.1 for an illustration.

2.5.2 Some algorithms for bouncing

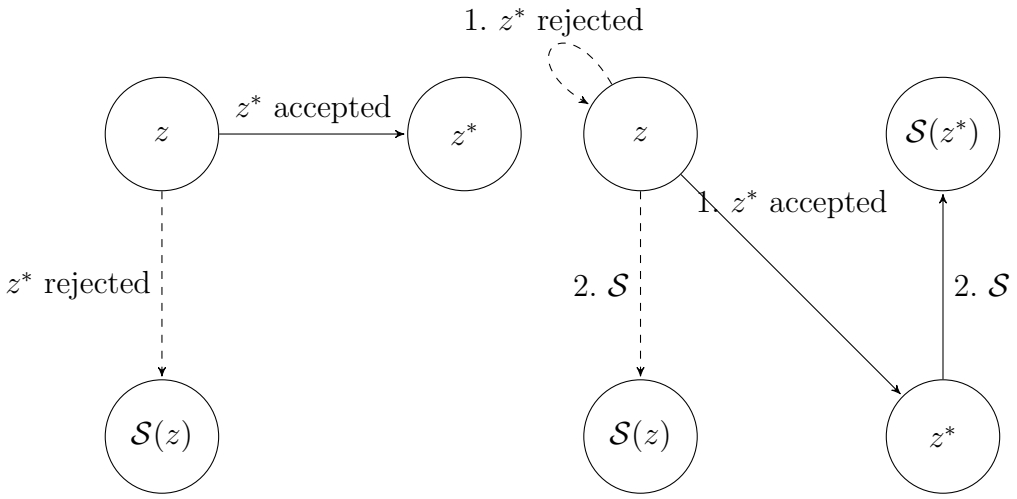
In this section we restrict ourselves to the setting where $z = (x, v) \in \mathbb{R}^d \times \mathbb{R}^d$; the v are independent of x, i.e. $\rho(z) = \pi(x)\psi(v)$; and the involution $\mathcal{S}$ is a reversal of velocity $\mathcal{S}(x, v) = (x, -v)$. We will take $\psi(v)$ to be multivariate Gaussian, however much of this is equally applicable to the case that $\psi(v)$ is uniform on the sphere $\mathbb{S}^{d-1}$. Additionally, we examine the case that $\lambda(x, v) = \langle \nabla U(x), v\rangle_+$, though these results also apply to factorized rates which take the form $\lambda_i(x, v) = \langle \nabla U_i(x), v\rangle_+$.

Restricting the kernel to the bounce form given in (2.24), we see that Q_x must satisfy

$$\int_{v\in\mathbb{R}^d} \psi(\mathrm{d}v)\langle \nabla U(x), v\rangle_+ Q_x(v,\mathrm{d}v') = \psi(\mathrm{d}v')\langle \nabla U(x), -v\rangle_+. \tag{2.28}$$

A decomposition of the velocity into the component parallel to the gradient and the component perpendicular to the gradient will prove useful. This decomposition was

Figure 2.1: An illustration of Metropolis-Hastings updates in a lifted chain, demonstrating the effect of a single kernel Q (left), versus the equivalent composition of two reversible kernels, Q' followed by a transformation $\mathcal{S}$ (right). In the figure on the left, a proposal $z^* \sim Q(z, \cdot)$ is either accepted, yielding the new state z^*; or rejected, yielding the new state $\mathcal{S}(z)$. In the figure on the right, there are two steps: 1. a proposal $z^* \sim Q'(z, \cdot)$ is either accepted, yielding the state z^*; or rejected, yielding the state z; and 2. the involution is then applied yielding $\mathcal{S}(z^*)$ or $\mathcal{S}(z)$.

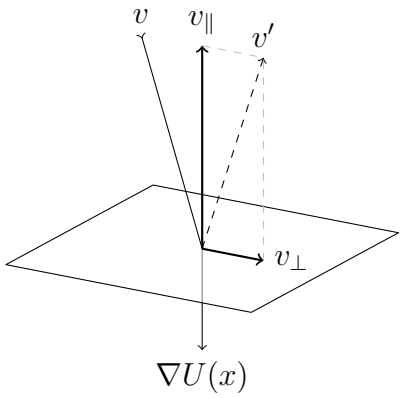


used in several proposals for alternate bounces (Michel et al. 2017, Wu and Robert 2017). The factorization of the jump distribution proportional to $\psi(\mathrm{d}v)\langle \nabla U(x), v\rangle_+$ using this decomposition was identified by Vanetti et al. (2017) and was used to describe the several proposed bounce alternatives that were used in the literature.

It turns out that this decomposition and factorization have been well-known in the rarefied gas literature for quite some time. The reflection of a gas particle bouncing off of a surface is analogous to our imagined "particle" bouncing off the plane normal to the gradient of the potential. Early descriptions of this behaviour include Maxwell (1867). Early examples of simulations include the work of Haviland (1965). Notions of detailed balance in this context are given by Kuščher (1971). See Shen (2006) for an explicit description of some of the reflection algorithms used in that literature.

Despite the presence of a significant literature pre-dating that used in statistics, it should be noted that those simulations were largely concerned with studying phenomena emerging from the simulation of microscopic dynamics. This is in

Figure 2.2: Illustration of the velocity decomposition (2.29).



contrast to the schemes presented here, where the dynamics are instead chosen to yield a particular invariant distribution. More relevant to the bounces studied here is that those simulations consider only three or fewer spatial dimensions.

The decomposition simply divides the velocity into two component vectors: one which is aligned with the gradient (cf. perpendicular to the wall) and another which is orthogonal to this. We use the notation

$$v = v_\perp + v_\parallel, \quad v_\parallel = a_\parallel \frac{\nabla U(x)}{\|\nabla U(x)\|_2^2}, \tag{2.29}$$

with $a_\parallel \in \mathbb{R}$ and where $\langle v_\perp, v_\parallel \rangle = 0$. We can construct these components from v by taking $a_\parallel = \langle v, \frac{\nabla U(x)}{\|\nabla U(x)\|_2^2} \rangle$ and $v_\perp = v - a_\parallel \nabla U(x)$. See the illustration of Figure 2.2.

With this notation we may write $\lambda(x, v) = \max\{0, \langle \nabla U(x), v \rangle\}$ which as a function of v is proportional to $\max\{0, a_\parallel\}$ and the invariance condition (2.28) becomes

$$\int \psi(\mathrm{d}v) \max\{0, a_\parallel\} Q_x(v, \mathrm{d}v') = \psi(\mathrm{d}v') \max\{0, -a'_\parallel\}.$$

The utility of the decomposition is demonstrated by noting that the components $v_\perp$ and $v_\parallel$ are *independent* under the skew-invariant distribution when these components are also independent under $\psi(v)$. We write

$$\psi(v) \max\{0, a_\parallel\} = \psi_\parallel(v_\parallel) \max\{0, a_\parallel\} \psi_\perp(v_\perp)$$

where $\psi_\parallel$ and $\psi_\perp$ are the independent components of ψ. This independence thus allows us to design Q_x to act on these components independently. Let

$$Q_x(v, \mathrm{d}v') = Q_{x\perp}(v_\perp, \mathrm{d}v'_\perp) Q_{x\parallel}(a_\parallel, \mathrm{d}a'_\parallel)$$

which implies the new invariance equations

$$\int \psi_\perp(\mathrm{d}v_\perp) Q_{x\perp}(v_\perp, \mathrm{d}v'_\perp) = \psi_\perp(-\mathrm{d}v') \tag{2.30}$$

$$\int \psi_\parallel(\mathrm{d}v_\parallel) \max\{0, a_\parallel\} Q_{x\parallel}(v_\parallel, \mathrm{d}v'_\parallel) = \psi(-\mathrm{d}v'_\parallel) \max\{0, a'_\parallel\}. \tag{2.31}$$

When ψ is the standard multivariate normal distribution, then $\psi_\perp$ is the $d-1$-dimensional normal distribution in the space perpendicular to $\nabla U(x)$, and $\psi_\parallel$ is the univariate normal distribution. We recognize that the condition on the parallel component can be understood as

$$\mathcal{N}(a_\parallel; 0, 1) \max\{0, a_\parallel\} \propto \chi(a_\parallel; 2),$$

that is, the χ-distribution with two degrees of freedom.

We list those kernels $Q_x(v, \mathrm{d}v')$ of which we are aware, all of which implement kernels satisfying (2.30) and (2.31).

1. **Standard reflections**. The typical reflection operator used in the bouncy particle sampler can be understood as a deterministic and skew-reversible kernel where the perpendicular component is unchanged and the parallel component is reversed:

$$Q_{x\perp}(v_\perp, \mathrm{d}v'_\perp) = \delta_{v_\perp}(\mathrm{d}v'_\perp)$$
$$Q_{x\parallel}(a_\parallel, \mathrm{d}a'_\parallel) = \delta_{-a_\parallel}(\mathrm{d}a'_\parallel).$$

2. **Independent sampling** (Fearnhead et al. 2018). Analogous to *diffuse reflection* in the rarefied gas literature (Shen 2006), the new velocity is independent of the old, taking:

$$Q_{x\perp}(v_\perp, \mathrm{d}v'_\perp) = \psi_\perp(\mathrm{d}v'_\perp)$$
$$Q_{x\parallel}(a_\parallel, \mathrm{d}a'_\parallel) = \chi(-\mathrm{d}a'_\parallel; 2).$$

When ψ is the multivariate normal, we can sample $v_\perp$ by first taking $v^* \sim \psi$ and then setting $v_\perp = v^* - \langle v^*, \frac{\nabla U(x)}{|\nabla U(x)|^2} \rangle$.

3. **Generalized BPS** (Wu and Robert 2017). The component parallel to the gradient is simply reversed, while the perpendicular component is sampled independently of its previous value:

$$Q_{x\perp}(v_\perp, \mathrm{d}v'_\perp) = \psi_\perp(\mathrm{d}v'_\perp)$$
$$Q_{x\parallel}(a_\parallel, \mathrm{d}a'_\parallel) = \delta_{-a_\parallel}(\mathrm{d}a'_\parallel).$$

4. **Forward-event chain** (Michel et al. 2017). The authors take ψ as the uniform distribution on $\mathbb{S}^{d-1}$, whereas we consider the scenario where ψ is the normal distribution. We suggest that an appropriate generalization to the Gaussian velocity would resample the parallel and perpendicular components independently, while resampling only the magnitude of the perpendicular component. To facilitate this, reparametrize $v_\perp$ into its magnitude and direction: $v_\perp = a_\perp u_\perp$ where $a_\perp \in \mathbb{R}^+$, $u_\perp \in \mathbb{R}^d$ and $\|u_\perp\|_2 = 1$, and note that $a_\perp \sim \chi(d-1)$ under $\psi_\perp$. The kernels can thus be written

$$Q_{x\perp}(a_\perp, u_\perp, \mathrm{d}a'_\perp, \mathrm{d}u'_\perp) = \chi(\mathrm{d}a'_\perp; d-1)\delta_{u_\perp}(\mathrm{d}u'_\perp)$$
$$Q_{x\parallel}(a_\parallel, \mathrm{d}a'_\parallel) = \chi(\mathrm{d}a'_\parallel; 2).$$

This is equivalent to the procedure for spherical ψ given by Michel et al. (2017) in the sense that it yields the same outgoing angle from the plane. To see this, we examine the distribution of $\sin\theta$ where θ is the angle formed by the new vector and the plane of reflection. In our case this is $\sin\theta = a'_\perp / \sqrt{a'^2_\perp + a'^2_\parallel}$ which by standard identities yields $\sin^2\theta \sim \mathrm{Beta}(1, \frac{d-1}{2})$, which is simulated by Michel et al. (2017) by taking $\sin\theta = \sqrt{1 - U^{2/(d-1)}}$ with $U \sim \mathrm{Uniform}(0,1)$.

If this bounce is used without refreshment, the resulting chain will not be ergodic. To address this, the authors include an additional step to follow the bounce: two random orthogonal unit vectors are chosen and v is modified such that the projections on these vectors are "swapped". This second step

Table 2.1: Error in variance estimates for several bounce algorithms.

Algorithm	$d = 16$	$d = 64$	$d = 256$	$d = 1024$
Autoregressive	.0047	.0060	.0079	.040
Forward-event chain	.0057	.013	.037	.22
Generalized	.0072	.014	.026	.53
Independent	.0043	.0059	.0093	.061
Reflection	.0091	.033	.23	.66

is performed only with some probability p_s, selected by the user. See Michel et al. (2017) for full details.

5. **Autoregressive bounce**. We propose a scheme similar to the above but with a different mechanism of controlling the randomness. Take $a'_{\|} \sim \chi(2)$ with probability p_b and $a'_{\|} = -a_{\|}$ otherwise, and update $v_{\perp}$ autoregressively:

$$Q_{x\perp}(v_{\perp}, \mathrm{d}v'_{\perp}) = \mathcal{N}(\mathrm{d}v_{\perp}; \rho v_{\perp}, (1-\rho^2)I)$$
$$Q_{x\|}(a_{\|}, \mathrm{d}a'_{\|}) = p_b \chi(\mathrm{d}a'_{\|}; 2) + (1-p_b)\delta_{a_{\|}}(\mathrm{d}a'_{\|}).$$

Yet another proposal is that of Terenin and Thorngren (2018), who suggest that a randomized bounce can be achieve by "swapping" the angle and radius, that is, by taking a reparameterization of the incoming velocity in terms of its angle to the gradient and its magnitude, they allow the outgoing angle to be a (random) function of the incoming magnitude, and the outgoing magnitude to be a function of the incoming angle. Their algorithm is not exact (except in the limit of increasing dimension) due to the dependence between these two variables. We suggest that the framework above is instructive for deriving valid bounces.

The properties of these randomized bounces are not yet well understood. In the next section, we compare them experimentally.

2.5.3 Numerical comparison of bounce algorithms

We used the various bounce algorithms to simulate an anisotropic Gaussian distribution with a diagonal covariance matrix of entries $\Sigma_{ii} = 10^{3(i-1)/(d-1)}$. Table 2.1

shows the mean absolute error of the estimate for Σ_{11} after 90000 iterations (9000 iterations for $d = 1024$).

No independent refreshments were used (i.e. $\lambda_{\text{ref}} = 0$). For the autoregressive algorithm, we chose $\rho = 0.5$ which performed well across all dimensions. Similarly, we chose the probability of "switching" for the forward-event chain algorithm to be $p_s = 1.0$.

We see that in higher dimensions the autoregressive algorithm performs well; we found these results to be fairly stable across values of ρ. The independent scheme also seems to scale well. Increasing λ_{ref} improves performance for the poorly-performing algorithms while it decreases performance for those performing well.

2.6 Discontinuous potentials

The rates and event kernels described up to this point all operate on continuous potentials, with the gradient of the potential ∇U playing a role in both the event rate and the event kernels. Here we present an extension to deal with potentials which are discontinuous, assuming that we can determine the times at which our continuous-time path would encounter a discontinuity. This case also accounts for boundaries or constraints on the state space, for instance by considering the potential beyond the boundary to be infinite.

We present a simple scheme applied at the point of the discontinuity which determines whether the trajectory is reflected off the boundary or continues unperturbed through the discontinuity. This is detailed in Algorithm 9, where we abuse notation and use $U(x_{k+1} + v_k\epsilon) - U(x_{k+1} - v_k\epsilon)$ to represent the instantaneous change in potential encountered when crossing the discontinuity, emphasizing that the discontinuity only has an effect in one direction; and we use $\nabla U(x_{k+1})$ at this point to represent the line normal to the discontinuity.

We make an heuristic argument for correctness of this algorithm as a limit of a continuous change in potential occurring over a decreasing interval. Take a boundary which is a straight line defined by $\langle g, x\rangle + b = 0$ with normal vector g. Smooth the discontinuity such that the potential around the line is continuous, that the total

change in potential is fixed and equal to that of the discontinuity, and that the gradient of the potential in this region is g. Applying the standard continuous-time rate $\langle \nabla U(x), v \rangle_+$ to this potential thus has probability $\min\{1, \exp(U(x - v\epsilon) - U(x + v\epsilon))\}$ of traversing the discontinuity without an event occurring. Furthermore, any event that does occur would be a reflection using the normal vector g.

This limiting argument bears a resemblance to that of Neal et al. (2011) who proposes an extension to Hamiltonian Monte Carlo to deal with constraints. We leave a rigorous derivation of this argument, including any technical conditions on the discontinuities, to future work.

Algorithm 9 Single step of a piecewise-deterministic Markov chain Monte Carlo algorithm for a target with discontinuous potential.

Given x_k, v_k, return x_{k+1}, v_{k+1}, piece duration t_k.
function DISCONTINUOUS-PDMCMC(x_k, v_k)
  Compute first time to encounter discontinuity τ_Δ, i.e. the discontinuity will be encountered at state $x + v\tau_\Delta$. If no discontinuity will be encountered on the current path, set $\tau_\Delta = \infty$.
  Simulate the first event time τ corresponding to the rate $\lambda(x + v\tau, v)$.
  if $\tau_\Delta < \tau$ **then**
    Set $x_{k+1} = x_k + v_k \tau_\Delta$.
    with probability $\min\{1, \exp(U(x_{k+1} - v_k\epsilon) - U(x_{k+1} + v_k\epsilon))\}$
      $v_{k+1} \leftarrow v_k$.
    otherwise
      Set $v_{k+1} = v_k - 2\frac{\langle \nabla U(x_{k+1}), v_k \rangle}{\|\nabla U(x_{k+1})\|_2^2} \nabla U(x_{k+1})$.
    return $x_{k+1}, v_{k+1}, \tau_\Delta$.
  else
    Sample $x_{k+1}, v_{k+1} \sim Q(x_k + v_k\tau, v_k)$.
    return x_{k+1}, v_{k+1}, τ.
  end if
end function

2.6.1 Previous work

Neal (2003) considers a similar situation in the reflective slice sampler, where when reaching the the boundary of the slice performs a reflection, which may be seen as a special case of the above algorithm.

Afshar and Domke (2015) propose an extension to the leapfrog scheme of Hamiltonian Monte Carlo which allows for *refractions* whenever a discontinuity

in U is encountered. They do so by modifying the configurational update of the leapfrog integrator, and where if the update $x + \epsilon p$ would cross such a discontinuity, they instead consider updating the momentum at the crossing point by the rule

$$p' = \begin{cases} \sqrt{p^2 - 2\Delta U} & \text{if } p^2 > 2\Delta U \\ -p & \text{otherwise} \end{cases}$$

where ΔU is the magnitude of the potential discontinuity at the point. It is not clear if such algorithms can be made to work in continuous-time; we suggest this as an avenue for future investigation.

Yi and Doshi-Velez (2017) arrive at a solution by replacing the discontinuous boundaries with a continuous approximation. They argue, as we do above, that this continuous approximation becomes a reflection in a limit where the continuous approximation is made to more closely match the discontinuity. However, they advocate the continuous approximation gives a computational speedup over reflecting at the boundaries, as it avoids having to compute exact intersection times; we suggest that this trade-off is perhaps implementation-specific as a continuous approximation must still be included for each discontinuity. Additionally, they demonstrate that as the faithfulness of the approximation is increased, the step-size of the leapfrog integrator must be commensurately decreased to accurately integrate the dynamics in these regions of sudden potential change.

Nishimura et al. (2017) describe an algorithm, based on the Hamiltonian dynamics of a Laplace-distributed momentum, which is perhaps better suited to discontinuous potentials. In their algorithm, the position is updated for each coordinate and reflected if rejected in an accept-reject step; they remark that their novel leapfrog scheme is related to the zig-zag sampler. Our algorithm may be seen as an analogous scheme in which all coordinates are updated at the same time.

2.7 Generalizing the flow: the Hamiltonian bouncy particle sampler

As mentioned, all previously proposed piecewise-deterministic Markov chain Monte Carlo methods utilize a linear flow (2.16). Nonetheless, the framework presented

in Section 2.2 allows for a more general choice of flow. We note that for a flow to be appropriate, we must be able to compute event rates $\langle \nabla H(z), \phi(z) \rangle_+$ and also simulate the result of the flow exactly for any time interval. (There may be exotic situations where we can both simulate the piecewise-deterministic Markov process and compute expectations without being able to compute this flow, however we do not explore this.)

We propose to use Hamiltonian dynamics for this flow; specifically, the Hamiltonian dynamics arising from a quadratic potential and quadratic kinetic energy. In this case, these dynamics can be exactly computed.

We thus suggest to select a quadratic potential which in some way is adapted to the problem, design the flow to target this potential, and correct for discrepancies between the target and the quadratic approximation through jump events. The quadratic potential could be selected in various ways, we highlight two:

1. If the distribution of interest is well-approximated by a Gaussian, then we suggest using this approximation.

2. When the model uses a Gaussian prior and the likelihood is weakly informative, the potential can be chosen as the prior. This mimics the setup of algorithms such as elliptical slice sampling Murray et al. (2010).

Similarly to before, we consider targets augmented by a velocity, however we use here the *momentum* p which is related to the velocity by $p = Mv$; this will aid in drawing connections to Hamiltonian mechanics. So we will target $\rho(x, p) = \pi(x)\psi(p)$ with $\pi(x) \propto \exp(-U(x))$ being the density of interest on $\mathcal{X} = \mathbb{R}^d$. ψ will be a multivariate normal distribution on $\mathcal{V} = \mathbb{R}^d$; we discuss a choice for its covariance matrix later in this section.

The Hamiltonian from which we derive our flow is based on an approximation to our potential, we denote this approximating Hamiltonian $\tilde{H}(x, p)$. The Hamiltonian can be decomposed into its potential and momentum terms,

$$\tilde{H}(x, p) = \tilde{U}(x) + K(p), \tag{2.32}$$

where $K(p) = \frac{1}{2}p^T M^{-1} p$ and $\tilde{U}(x) = \frac{1}{2}x^T M x$ is the approximating quadratic potential described above. Note that, while in principle a mass matrix may be chosen for K which is not related to the potential, we make this choice explicitly as this yields a flow which is computable. We can then rewrite the target as $\rho(x,p) = \exp(-H(x,p))$ where

$$H(x,p) = D(x) + \tilde{U}(x) + K(p),$$

where $D(x) := U(x) - \tilde{U}(x)$.

The Hamiltonian flow Φ is induced by the ordinary differential equation of drift

$$\phi(z) = \begin{bmatrix} \nabla_p \tilde{H}(x,p) \\ -\nabla_x \tilde{H}(x,p) \end{bmatrix}.$$

As the potential and kinetic energy are both quadratic we can write the flow in a linear form

$$\phi\left(\begin{bmatrix} x \\ p \end{bmatrix}\right) = \begin{bmatrix} 0 & M^{-1} \\ -M & 0 \end{bmatrix} \begin{bmatrix} x \\ p \end{bmatrix}$$

which has a solution

$$\Phi_t\left(\begin{bmatrix} x \\ p \end{bmatrix}\right) = \exp\left(\begin{bmatrix} 0 & M^{-1} \\ -M & 0 \end{bmatrix} t\right) \begin{bmatrix} x \\ p \end{bmatrix}. \tag{2.33}$$

Due to our choice of mass matrix, the exponential series simplifies as

$$\exp\left(\begin{bmatrix} 0 & M^{-1} \\ -M & 0 \end{bmatrix} t\right) = I\cos(t) + \begin{bmatrix} 0 & M^{-1} \\ -M & 0 \end{bmatrix} \sin(t).$$

This yields the flow

$$\Phi_t(z) = \begin{bmatrix} \cos(t)x + \sin(t)M^{-1}p \\ \cos(t)p - \sin(t)Mx \end{bmatrix}$$

which is computable. Note that in terms of velocity $v = M^{-1}p$ this can be expressed as

$$\begin{bmatrix} x_t \\ v_t \end{bmatrix} = \begin{bmatrix} \cos(t)x + \sin(t)v \\ \cos(t)v - \sin(t)x \end{bmatrix}.$$

2.7.1 Bounce rates

To establish a valid bounce rate, we examine condition A1.2 which reduces to

$$\begin{aligned}\lambda(x,-p)-\lambda(x,p) &= \nabla\cdot\phi(x,p)-\langle\nabla H(x,p),\phi(x,p)\rangle\\ &= -\langle\nabla_x D(x),M^{-1}p\rangle-\langle\nabla_x\tilde{U}(x),M^{-1}p\rangle+\langle\nabla_p K(p),Mx\rangle\\ &= -\langle\nabla_x D(x),M^{-1}p\rangle\end{aligned}$$

as the flow is divergence-free. Thus it is sufficient for the rate to be based on the gradient of the residual $D(x)$. Indeed we can see that A1.2 will be satisfied for the event rate

$$\lambda(x,p)=\lambda_{\text{ref}}+\langle\nabla D(x),M^{-1}p\rangle_+.$$

With this, condition A1.3 now becomes, for a bounce kernel $Q(x,p,\mathrm{d}x',\mathrm{d}p')=\delta_x(\mathrm{d}x')Q_x(p,\mathrm{d}p')$,

$$\int\psi(p)\langle\nabla D(x),M^{-1}p\rangle_+Q_x(p,\mathrm{d}p')=\psi(-\mathrm{d}p')\langle\nabla D(x),-M^{-1}p'\rangle_+. \tag{2.34}$$

The bounces proposed in Section 2.5 do not account for the selection of an arbitrary mass matrix as the kinetic energy $p^TM^{-1}p$ is not in general invariant to the bounce operator $R(p;g)$. However, we see that for the reflection vector g, taking as our reflection

$$p'=p-2\frac{g^TM^{-1}p}{g^TM^{-1}g}g$$

does leave the kinetic energy invariant; denote this mass-corrected reflection $R_M(p;g)$. This is a generalization of the previously considered bounce operator $R(p;g)=R_I(p;g)$. To see that this is invariant we note that

$$\left(p-2\frac{g^TM^{-1}p}{g^TM^{-1}g}g\right)^TM^{-1}\left(p-2\frac{g^TM^{-1}p}{g^TM^{-1}g}g\right)=p^TM^{-1}p$$

thus for any g the potential is invariant. Now to confirm that (2.34) is satisfied we check that this operator leaves invariant the rate terms, indeed we see that for $p'=R_M(p;\nabla D(x))$ we have

$$\begin{aligned}\langle\nabla D(x),-M^{-1}p'\rangle_+ &= \left\{\langle\nabla D(x),-M^{-1}p\rangle-2\frac{\nabla D(x)^TM^{-1}p}{\nabla D(x)^TM^{-1}\nabla D(x)}\langle\nabla D(x),-M^{-1}\nabla D(x)\rangle\right\}_+\\ &= \langle\nabla D(x),M^{-1}p\rangle_+.\end{aligned}$$

Table 2.2: Operations for velocity and momentum.

	Velocity	Momentum
Kinetic Energy	$\frac{1}{2}v^T M v$	$\frac{1}{2}p^T M^{-1} p$
Covariance matrix	M^{-1}	M
Hamiltonian flow	$\begin{bmatrix} x_t \\ v_t \end{bmatrix} = \begin{bmatrix} \cos(t)x + \sin(t)v \\ \cos(t)v - \sin(t)x \end{bmatrix}$	$\begin{bmatrix} x_t \\ p_t \end{bmatrix} = \begin{bmatrix} \cos(t)x + \sin(t)M^{-1}p \\ \cos(t)p - \sin(t)Mx \end{bmatrix}$
Bounce	$v - 2\frac{g^T v}{g^T M^{-1} g} M^{-1} g$	$p - \frac{g^T M^{-1} p}{g^T M^{-1} g} g$

Thus as both ψ and the rate are invariant to the bounce, the entire jump distribution is invariant.

It is less clear how one might modify the alternative bounces discussed in Section 2.5 to accommodate a non-identity mass matrix. We leave this to future work.

2.7.2 Velocity and momentum

We established our Hamiltonian in terms of the momentum where $K(p) = \frac{1}{2}p^T M^{-1} p$. To connect our algorithm to related algorithms in the literature, we present an alternative formulation in terms of a velocity variable v, related to the momentum by $v = M^{-1}p$.

See Table 2.2, we give the equivalent operations when using either velocity or momentum. While the Hamiltonian is derived in terms of momentum, in applications we typically prefer to deal with the velocity directly as this yields a simpler equation for the flow.

To sample v, this is typically first simulating some d-dimensional vector u from the standard multivariate Gaussian distribution, then multiplied by a matrix which is the Cholesky decomposition of the covariance of the target distribution. Specifically, if the Cholesky decomposition can be written $LL^T = M^{-1}$, then $v = Lu$. Alternatively, we can solve for v in the equation $L^{-1}v = u$. This latter case is useful when M is sparse, a common case in time-series or spatial models where the nonzero elements of M indicate a relationship between variables which are adjacent in time or in space. Since $(L^T)^{-1}L^{-1} = M$, L^{-1} is sparse when M is and when the bandwidth of M is limited; see Rue and Held (2005) for details.

2.7.3 Previous work

The Hamiltonian bouncy particle sampler is closely connected to several antecedents in the literature. A large body of work is dedicated to Hamiltonian Monte Carlo itself; see for example the review of Neal et al. (2011). A key difference between our algorithm and the standard Hamiltonian Monte Carlo algorithm is that the leapfrog discretization used in the discrete-time algorithm may target Hamiltonians using arbitrary potential functions (so long as they are pointwise evaluable and differentiable). Our insistence on a computable flow Φ_t for any $t \geq 0$ restricts the continuous-time variant to use flows based on a much smaller class of Hamiltonians. We therefore focus our review on those algorithms using computable continuous-time flows.

None of the following examples actually describe continuous-time piecewise-deterministic Markov chain Monte Carlo algorithms, in the sense that they do not output a continuous-time path. Nonetheless, many of them do simulate continuous paths to arrive at a new sample, and do so in a way that is rejection-free. Our framework above allows many of these to be trivially converted into continuous-time algorithms by simply retaining the simulated paths.

The reflective slice sampler algorithm (Neal 2003), discussed previously, is an exact algorithm wherein the Hamiltonian dynamics target a slice-conditional distribution. For a target density $\pi(x)$, introducing and conditioning on a slice variable u yields the conditional density $\pi(x|u) \propto \mathbb{I}(\pi(x) > u)$. The gradient of this density is zero within the slice, and the "Hamiltonian dynamics" reduce to linear trajectories; a reflection operator is used at event times (when $\pi(x) = u$), reflecting off the boundary of the slice which is $\nabla U(x)$. Downs et al. (2000) introduce the nonnegative Boltzmann machine and use this algorithm to sample from it.

Murray et al. (2010) describe the *elliptical slice sampler* which considers points $\cos(t)x + \sin(t)v$ for $t \in [0, 2\pi)$, and with $v \sim \mathcal{N}(0, \Sigma)$. They suggest that Σ be taken as the covariance of the prior distribution on the target. We see that this set of points corresponds exactly to those reached by our Hamiltonian flow for the same velocity. The algorithms differ in that Murray et al. (2010) correct for the

discrepancy between the target U and the Hamiltonian potential $\tilde{U}$ using a slice sampling mechanism, whereas we use bounces. Bloem-Reddy and Cunningham (2016) recognize the connection between Hamiltonian simulation and the elliptical slice sampler, and extend the reflective slice sampler to more arbitrary Hamiltonian flows in their Hamiltonian slice sampler.

Pakman and Paninski (2014) describe the *exact Hamiltonian Monte Carlo* algorithm, which targets distributions of the form $\pi(x) \propto \mathcal{N}(0, \Sigma)\mathbb{I}(x \in A)$, that is, truncated normal distributions. This may be easily understood in our framework, as a Hamiltonian flow targeting $\mathcal{N}(0, \Sigma)$ yields a residual which is simply the restriction of the state to the set A. Similarly to the reflective slice sampler, reflections are performed with respect to the boundary.

2.8 Generalizing the event kernel: tunnelling

In Section 2.5 we discussed how the event transition kernel may be randomized and demonstrated how these bounces may be understood as Markov chain Monte Carlo transitions. Those "bounces" all assumed a transition kernel where the position is unchanged, and the transition kernel takes the form $Q(x, v, \mathrm{d}x', \mathrm{d}v') = \delta_x(\mathrm{d}x')Q_x(v, \mathrm{d}v')$.

We introduce here a new transition scheme which we call *tunnelling*, inspired by a family of discrete-time transitions which we discuss in more depth in Section 3.8. Upon reaching an event time, these transitions will select a new state by continuing to simulate the flow and selecting a future point along the flow. We imagine that such transitions can be viewed as "tunnelling" through regions of low probability and emerging in regions of higher probability.

For some choices of flow this construction may not be appropriate. For example, when employing a linear flow it is clear that (for any reasonable distribution) attempting to follow the flow indefinitely will only lead to regions of decreasing density. However, we note that the flow simulated while tunnelling need not be the same as the deterministic flow of the piecewise-deterministic Markov process.

Algorithm 10 can be used to compute the new state following the tunnelling jump. The algorithm operates by locating the next point along the flow which matches the density of the point at which the jump occurred. We assume here that such a point exists; whether or not this is true may depend both on the flow and the target distribution.

Algorithm 10 Simulation of a continuous-time tunnelling jump.

Given current state z, return new state z'.
Find t^* where

$$t^* = \inf\{t > 0 : \rho(\Phi_t(z)) = \rho(z)\}$$

return $\Phi_{t^*}(z)$.

The algorithm is correct as it satisfies the detailed balance condition for a deterministic transform. To show this, consider only states along the path $\{\Phi_t(z) : t \in \mathbb{R}\}$. Define the deterministic transform $g_z : \mathbb{R} \mapsto \mathbb{R}$ as a function mapping points along this path

$$g_z(t) = \inf\{t^* \in (t, \infty) : \rho(\Phi_{t^*}(z)) = \rho(\Phi_t(z)).\}$$

This transformation leaves invariant $\rho(z)\langle\nabla H(z), \phi(z)\rangle_+$ and thus describes a valid event kernel as it satisfies A1.3. To prove this we show detailed balance of the deterministic transform. First find the gradient of $g_z(t)$ at $t = 0$. Let $a = \rho(\Phi_t(z))$; for t^* such that $\rho(\Phi_{t^*}(z)) = a$, $\frac{\mathrm{d}t^*}{\mathrm{d}a} = \langle\nabla\rho(\Phi_{t^*}(z)), \phi(\Phi_{t^*}(z))\rangle^{-1}$ and $a = \rho(\Phi_t(z))$ gives $\frac{\mathrm{d}a}{\mathrm{d}t} = \langle\nabla\rho(\Phi_t(z)), \phi(\Phi_t(z))\rangle$. Thus taking $\frac{\mathrm{d}}{\mathrm{d}t}g_z(t) = \frac{\mathrm{d}t^*}{\mathrm{d}a}\frac{\mathrm{d}a}{\mathrm{d}t}$ gives

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}g_z(t) &= \frac{\langle\nabla\rho(\Phi_t(z)), \phi(\Phi_t(z))\rangle}{\langle\nabla\rho(\Phi_{t^*}(z)), \phi(\Phi_{t^*}(z))\rangle} \\ &= \frac{\langle\nabla H(\Phi_t(z)), \phi(\Phi_t(z))\rangle}{\langle\nabla H(\Phi_{t^*}(z)), \phi(\Phi_{t^*}(z))\rangle}\end{aligned}$$

thus substituting $\frac{\mathrm{d}}{\mathrm{d}t}g_z(t)|_{t=0}$ into our detailed balance equation for a deterministic transform gives

$$\begin{aligned}\rho(z)\langle\nabla H(z), \phi(z)\rangle_+ \left[\frac{\mathrm{d}}{\mathrm{d}t}g_z^{-1}(t)\right]_{t=0} &= \rho(z)\langle\nabla H(z), \phi(z)\rangle_+ \frac{\langle\nabla H(\Phi_{t^*}(z)), \phi(\Phi_{t^*}(z))\rangle}{\langle\nabla H(z), \phi(z)\rangle} \\ &= \rho(\Phi_{t^*}(z))\langle\nabla H(\Phi_{t^*}(z)), \phi(\mathcal{S} \circ \Phi_{t^*}(z))\rangle_+\end{aligned}$$

The above relies on $g_z(t)$ being a one-to-one function, and also that for any $t^* = g_z(t)$ we have that the energy is decreasing at that point $\langle \nabla H(\Phi_{t^*}(z)), \phi(\Phi_{t^*}(z)) \rangle < 0$, so an event could occur from the time-reversal of the proposed state.

In Section 3.8 we will describe a framework of similar algorithms in discrete time; the construction above can be seen as the limit in decreasing step-size of the algorithm used there.

3

Piecewise-deterministic Markov chain Monte Carlo in discrete time

The continuous-time piecewise-deterministic Markov chain Monte Carlo algorithms of the previous chapter describe an exciting and novel development in the use of Markov processes to sample from distributions. However they are not without their difficulties, of which we highlight a few:

1. Continuous-time algorithms require much more effort on the part of the user. To implement these algorithms, we must be able to simulate bounce times according to the rate $\langle \nabla U(x), v \rangle_+$, or failing this, to derive a bounding rate $\lambda(x, v) \geq \langle \nabla U(x), v \rangle_+$ which we can simulate. In either case, these rates and the methods to simulate from them may not be readily available, and may require substantial innovation. In the case that we must use a bounding rate, it may be seen as a sort of randomized step size adapted to the problem, and poor bounds may preclude an efficient sampling algorithm.

   Contrast this with the Metropolis-Hastings algorithm, where one must simply define the target density up to a normalizing constant. In Bayesian statistics, we typically have access to the density of the posterior (up to a normalizing constant), and so implementing an Markov chain Monte Carlo algorithm is immediate. Algorithms such as Hamiltonian Monte Carlo, which additionally

use the gradient of the density, are often just as easily implemented, largely thanks to innovations in automatic differentiation.

2. The design choices available to the user are more limited than in discrete-time. While in discrete time we are given a great deal of flexibility in choosing e.g. a proposal distribution, the choices available in continuous-time are far less flexible. One might argue that the availability of very few choices for flow, such as the linear and Hamiltonian flows discussed in Chapter 2, is similar to having only two types of discrete-time proposals available.

In light of these difficulties, we explore the idea of *discretizing* these continuous-time algorithms. A discretization may yield an algorithm which does not require the complicated bounds mentioned above; on the other hand, we may find that an algorithm only evaluating the target pointwise will not be performant as it does not take advantage of the extra information provided by such bounds.

Our results are perhaps surprising in that we show that the continuous-time mechanisms of event rate simulation have analogues in the discrete-time acceptance probability which offer similar performance improvements. The particular applications to large datasets and to sparsely-connected models are explored in Chapters 4 and 5, respectively. These results may suggest that the non-reversible and continuous-time aspects of piecewise-deterministic Markov chain Monte Carlo are not crucial for good performance.

In seeking discrete-time piecewise-deterministic Markov chain Monte Carlo methods we will not be approximating the continuous-time methods by a discretization, as one might approximate e.g. a differential equation by a discretization. Instead, we will derive discrete-time algorithms which will be exact in the sense that they leave invariant a target distribution. The analogies drawn between continuous- and discrete-time will be straightforward and intuitive.

If these discrete-time algorithms are to be considered a discretization of the continuous-time algorithms in any way, it will be in the sense that, if imbued with a notion of *step size*, they will converge weakly to their continuous-time counterparts.

The difficulty in establishing this convergence is substantial and has been done for a single case, see Vanetti et al. (2017).

We begin by describing a discrete-time analogue of a piecewise-deterministic Markov process.

3.1 The discrete-time piecewise-deterministic Markov process

We introduce here a class of discrete-time Markov chains which we will call *discrete-time piecewise-deterministic Markov processes*. As in the continuous-time scenario, we assume for simplicity that the process is defined on $\mathcal{Z} = \mathbb{R}^n$. A $\mathcal{Z}$-valued discrete-time piecewise-deterministic Markov process process $\{z_t; t \in \mathbb{N}_{\geq 0}\}$ involves a deterministic path altered by random jumps at random event times. It is defined through

1. a diffeomorphism $\Phi : \mathcal{Z} \to \mathcal{Z}$ with the absolute value of the determinant of the Jacobian satisfying $|\mathbf{J}_\Phi(z)| > 0$ for all z,

2. an acceptance probability $\alpha : \mathcal{Z} \to [0, 1]$ with $1 - \alpha(z)$ being the probability of having an event at the next time step when the current state is z, and

3. a Markov transition kernel Q from $\mathcal{Z}$ to $\mathcal{Z}$ where the state at event time t is given by $z_t \sim Q(z_{t-1}, \cdot)$.

These three components mimic the three components of the continuous-time piecewise-deterministic Markov process as defined in Davis (1984) and given in Section 2.1.

We emphasize that this should not be thought of as an approximation of a continuous-time piecewise-deterministic Markov process. Rather, we see that the three components have obvious analogues in continuous-time: the diffeomorphism Φ is analogous to the flow; the rejection probability analogous to the bounce rate, and discrete event times analogous to continuous event times; and the event transition kernels are analogous to the continuous-time transition kernels.

Algorithm 11 describes how to simulate the path of a discrete-time piecewise-deterministic Markov process. Allow $\prod_{i=0}^{j} a_i = 1$ for all $j < 0$ and any a_i, $\Phi_0(z) = z$, $\Phi_{r+1}(z) = \Phi_r(\Phi(z))$ for $r \in \mathbb{N}_{>0}$, i.e., Φ satisfies the semigroup property $\Phi_s \circ \Phi_r = \Phi_{s+r}$ for all $s, r \in \mathbb{N}_{\geq 0}$. We have used the convention that τ_k represents the number of successive applications of Φ before undertaking a jump. Thus each iteration of the algorithm adds $\tau_k + 1$ discrete-time samples.

Each step of the algorithm mimics a corresponding step in the continuous-time piecewise-deterministic Markov process: the simulation of τ_k is analogous to the simulation of t_k and, as mentioned, the jump kernel Q is analogous to that of the continuous-time algorithm.

Algorithm 11 Simulation of a discrete-time piecewise-deterministic Markov process

$t_0 \leftarrow 0$.
Initialize z_0 arbitrarily on $\mathcal{Z}$.
for $k \in \{1, 2, \dots\}$ **do**
  Sample inter-event time $\tau_k \in \mathbb{N}_{\geq 0}$ with distribution

$$\mathbb{P}[\tau_k = j] = \left\{1 - \alpha(\Phi_j(z_{t_{k-1}}))\right\} \prod_{i=0}^{j-1} \alpha(\Phi_i(z_{t_{k-1}})). \tag{3.1}$$

  if $\tau_k \geq 1$ **then**
    Set $z_{t_{k-1}+r} \leftarrow \Phi_r(z_{t_{k-1}})$ for $r \in \{1, \dots, \tau_k\}$.
  end if
  $t_k \leftarrow t_{k-1} + \tau_k + 1$.
  Sample $z_{t_k} \sim Q(z_{t_k - 1}, \cdot)$.
end for

The process $\{z_t; t \in \mathbb{N}_{\geq 0}\}$ can be seen simply as a Markov process of transition kernel

$$K(z, \mathrm{d}z') = \alpha(z)\delta_{\Phi(z)}(\mathrm{d}z') + (1 - \alpha(z))Q(z, \mathrm{d}z'). \tag{3.2}$$

3.2 From discrete-time piecewise-deterministic Markov process to discrete-time piecewise-deterministic Markov chain Monte Carlo

As in the continuous-time case (Section 2.2), we are now interested in designing discrete-time piecewise-deterministic Markov processes which leave invariant a

target density $\rho(z)$ given by

$$\rho(z) \propto \exp(-H(z))$$

using a discrete-time piecewise-deterministic Markov process process. Invariance of the kernel K with respect to ρ is satisfied if

$$\int \rho(\mathrm{d}z)K(z,\mathrm{d}z') = \rho(\mathrm{d}z'). \tag{3.3}$$

From (3.2), (3.3) can be rewritten as

$$\rho(\Phi^{-1}(z'))\alpha(\Phi^{-1}(z'))|\mathbf{J}_{\Phi^{-1}}(z')|\mathrm{d}z' + \int \rho(\mathrm{d}z)\left\{1-\alpha(z)\right\}Q(z,\mathrm{d}z') = \rho(\mathrm{d}z'). \tag{3.4}$$

We provide here sufficient conditions on Φ, α, and Q to ensure ρ-invariance of the associated discrete-time piecewise-deterministic Markov process

(**A3**) Assumptions on Φ, α, and Q:

1. There exists a ρ-preserving mapping $\mathcal{S} : \mathcal{Z} \to \mathcal{Z}$.
2. The acceptance probability α satisfies

$$\rho(z)\alpha(z)|\mathbf{J}_{\Phi}(z)|^{-1} = \rho(\mathcal{S}\circ\Phi(z))\alpha(\mathcal{S}\circ\Phi(z)). \tag{3.5}$$

3. The kernel Q satisfies

$$\int \rho(\mathrm{d}z)(1-\alpha(z))Q(z,\mathrm{d}z') = \rho(\mathcal{S}^{-1}(\mathrm{d}z'))(1-\alpha(\mathcal{S}(z'))). \tag{3.6}$$

Proposition 3.1. *Assume A3. Then the discrete-time piecewise-deterministic Markov process admits ρ as invariant distribution.*

Proof. The transition kernel is

$$K(z,\mathrm{d}z') = \alpha(z)\delta_{\Phi(z)}(\mathrm{d}z') + (1-\alpha(z))Q(z,\mathrm{d}z').$$

Plugging into the global balance equation,

$$\begin{aligned}\int_{z\in\mathcal{Z}} \rho(\mathrm{d}z)K(z,\mathrm{d}z') &= \int_{z\in\mathcal{Z}} \rho(\mathrm{d}z)\alpha(z)\delta_{\Phi(z)}(\mathrm{d}z') + \int_{z\in\mathcal{Z}} \rho(\mathrm{d}z)(1-\alpha(z))Q(z,\mathrm{d}z') \\ &= \rho(\Phi^{-1}(\mathrm{d}z'))\alpha(\Phi^{-1}(z'))|\mathbf{J}_{\Phi^{-1}}(z')| + \rho(\mathcal{S}(\mathrm{d}z'))(1-\alpha(\mathcal{S}(z')))\end{aligned}$$

Thus it is sufficient that $\rho(\Phi^{-1}(\mathrm{d}z'))\alpha(\Phi^{-1}(z'))|\mathbf{J}_{\Phi^{-1}}(z')| - \rho(\mathcal{S}(\mathrm{d}z'))\alpha(\mathcal{S}(z')) = 0$. Replacing $z' = \Phi(z)$ and $|\mathbf{J}_{\Phi^{-1}}(\Phi(z))| = |\mathbf{J}_{\Phi}(z)|^{-1}$, this is satisfied by A3.2. □

Conditions A3.1 to A3.3 parallel analogous conditions A1.1 through A1.3 by which the continuous-time chain is made invariant to ρ.

We have not specified any condition on $\mathcal{S}$ beyond that it is ρ-preserving. However, the availability of a solution for A3.2 is dependant on the choice of $\mathcal{S}$. We suggest that the only form of $\mathcal{S}$ for which this is generally available is when $\mathcal{S}$ "reverses" the transformation Φ in the sense that

$$\mathcal{S} \circ \Phi_k \circ \mathcal{S} \circ \Phi_k = \mathrm{Id} \tag{3.7}$$

for any $k \geq 0$. In this case, an acceptance rate α satisfying A3.2 can easily be found, as discussed in the next section. Henceforth we assume that $\mathcal{S}$ satisfies (3.7); since this also implies that $\mathcal{S} \circ \mathcal{S} = \mathrm{Id}$ we refer to $\mathcal{S}$ as an involution.

3.2.1 As a lifted chain

We now show that the discrete-time piecewise-deterministic Markov chain Monte Carlo algorithm can be interpreted as a lifted Markov chain. In Section 2.5.1, we showed that this was the case for the event kernel Q of the continuous-time algorithm. In the discrete-time case, we can describe the entire kernel K in a similar way, wherein we first apply a reversible kernel and then subsequently apply the involution $\mathcal{S}$.

So whereas the algorithm appears to propose a non-reversible move from state z to state $\Phi(z)$, in the lifted framework we interpret this move as a composition of two reversible kernels:

1. the first kernel assigns $z' \leftarrow \mathcal{S} \circ \Phi(z_t)$ with probability $\alpha(z)$; otherwise $z' \sim Q'(z_t, \cdot)$. We use the same notation as in Section 2.5.1 where Q' is constructed by first sampling from Q and then applying the involution, i.e. $Q'(z, \mathrm{d}z') = \int Q(z, \mathrm{d}z'')\delta_{\mathcal{S}(z'')}(\mathrm{d}z')$, or vice-versa.

2. the second kernel assigns $z_{t+1} \leftarrow \mathcal{S}(z')$.

As $\mathcal{S} \circ \mathcal{S} = \mathrm{Id}$, we see that the net effect of these two kernels is that $z_{t+1} \leftarrow \Phi(z_t)$ with probability $\alpha(z)$ and otherwise $z_{t+1} \sim Q(z_t, \cdot)$. As such, applying these two

kernels in order is equivalent to a single step of discrete-time piecewise-deterministic Markov chain Monte Carlo.

Note that Q' satisfies

$$\int \rho(\mathrm{d}z)(1-\alpha(z))Q'(z,\mathrm{d}z') = \rho(\mathrm{d}z')(1-\alpha(z')). \tag{3.8}$$

This has an immediate interpretation as a global balance equation in which Q' can be seen as a kernel which leaves invariant the distribution $\nu(\mathrm{d}z) \propto \rho(\mathrm{d}z)(1-\alpha(z))$ (since $0 \leq 1-\alpha(z) \leq 1$ this measure is also positive and finite if ρ is). This is the distribution of z when taken at times immediately preceding rejections of $\Phi(z)$, and is analogous to the invariant distribution of the jump chain of the continuous-time algorithm (Section 2.2).

3.2.2 Choosing α

The conditions on α and on Q are clear: as for the event kernel in the continuous-time case (Section 2.5.1), the skew-detailed balance conditions for α and Q allow us to make use of general Markov chain Monte Carlo kernels.

For α we use the Metropolis-Hastings algorithm for a deterministic proposal, as presented in Section 1.2.3. We cannot use the argument of Peskun (1973) that this choice is optimal in terms of the asymptotic variance of ergodic averages, since those results only apply to reversible chains (though see Andrieu and Livingstone (2019) for some results on continuous time chains). In any case, this is the choice that maximizes the acceptance probability.

We mention one other choice, available when using a flow Φ_k which is also continuous, that is, when Φ_k is defined not just for $k \in \mathbb{N}_{>0}$ but for $k \in \mathbb{R}_+$. If we simulate the continuous-time rate $\lambda(\Phi_t(z))$ over the time interval $t \in [0,1]$, then if no events occur during this time then we can accept the transition. For e.g. a linear flow the acceptance rate would be $\alpha(x,v) = \exp\left(-\int_0^1 \langle \nabla U(x+vs), v\rangle_+ \mathrm{d}s\right)$. This acceptance rate is in general less than the Metropolis-Hastings probability which can be written $\exp\left(-[\int_0^1 \langle \nabla U(x+vs), v\rangle \mathrm{d}s]_+\right)$. This option is perhaps more of a curiosity, but nonetheless helps to emphasize one difference between the continuous-

and discrete-time algorithms. In Section 3.6 we discuss how these continuous-time rates might be leveraged in a discrete-time algorithm.

Here we detail the conditions for when we use a Metropolis-Hastings update for both α and Q. This is the acceptance probability used by all existing algorithms and of all the algorithms we propose in this thesis (with the exception of the fast-forwarding schemes described in Section 3.6), and so detailing the precise expressions will be useful.

As in the continuous-time case, the rejection of a Metropolis-Hastings acceptance step results in the application of the involution, i.e. the new state is $\mathcal{S}(z)$.

3.2.2.1 The Metropolis-Hastings acceptance probability for α

With our choice of $\mathcal{S}$ as a Φ-reversing involution (3.7), the condition A3.2 can be interpreted as a detailed balance condition for a deterministic proposal. The Metropolis-Hastings acceptance probability for such a proposal is

$$\alpha(z) = \min\left\{1, \frac{\rho(\mathcal{S}\circ\Phi(z))}{\rho(z)}|\mathbf{J}_{\mathcal{S}\circ\Phi}(z)|\right\}.$$

Note that as $\rho(\mathcal{S}(z)) = \rho(z)$ by A3.1 and $|\mathbf{J}_{\mathcal{S}}(z)| = 1$ the acceptance rate is unchanged by omitting the $\mathcal{S}$:

$$\alpha(z) = \min\left\{1, \frac{\rho(\Phi(z))}{\rho(z)}|\mathbf{J}_{\Phi}(z)|\right\}. \tag{3.9}$$

3.2.2.2 The Metropolis-Hastings acceptance probability for Q

We provide conditions to implement an acceptance ratio for Q, where we have shown that Q' must satisfy the invariance equation (3.6) and thus we can use any ν-invariant kernel, including any ν-invariant Metropolis-Hastings algorithm. Writing the density of the proposal as $q'(z, z')$, such an algorithm would take the acceptance probability

$$\beta'(z, z') = \min\left\{1, \frac{\rho(z')(1-\alpha(z'))q'(z', z)}{\rho(z)(1-\alpha(z))q'(z, z')}\right\}. \tag{3.10}$$

Using α as in (3.9) gives

$$\begin{aligned}\beta'(z,z') &= \min\left\{1, \frac{\rho(z')\left(1-\min\left\{1,\frac{\rho(\Phi(z'))}{\rho(z')}\right\}\right)q'(z,z')}{\rho(z)\left(1-\min\left\{1,\frac{\rho(\Phi(z))}{\rho(z)}\right\}\right)q'(z',z)}\right\} \\ &= \min\left\{1, \frac{[\rho(z')-\rho(\Phi(z')]_+\, q'(z,z')}{[\rho(z)-\rho(\Phi(z))]_+\, q'(z',z)}\right\}.\end{aligned}$$

For the above, the next state of the chain would be obtained after applying the involution, either $\mathcal{S}(z')$ in the case of an acceptance, or $\mathcal{S}(z)$ otherwise. We may also define an acceptance probability which allows us to define a Metropolis-Hastings kernel in terms of the final state (but where a rejection would still yield $\mathcal{S}(z)$). Here the proposal has density $q(z,z') = q'(z,\mathcal{S}(z'))$:

$$\begin{aligned}\beta(z,z') &= \beta'(z,\mathcal{S}(z')) \\ &= \min\left\{1, \frac{[\rho(\mathcal{S}(z'))-\rho(\Phi\circ\mathcal{S}(z'))]_+}{[\rho(z)-\rho(\Phi(z))]_+}\frac{q'(\mathcal{S}(z'),z)}{q'(z,\mathcal{S}(z'))}\right\} \\ &= \min\left\{1, \frac{[\rho(z')-\rho(\Phi\circ\mathcal{S}(z'))]_+}{[\rho(z)-\rho(\Phi(z))]_+}\frac{q(\mathcal{S}(z'),\mathcal{S}(z))}{q(z,z')}\right\} \qquad (3.11)\end{aligned}$$

For completeness we include the case when the proposal $q(z,z')$ is deterministic. Denoting the proposal Ψ, the acceptance rate is

$$\beta'(z,\mathcal{S}\circ\Psi(z)) = \min\left\{1, \frac{[\rho(\Psi(z))-\rho(\Phi\circ\mathcal{S}\circ\Psi(z))]_+}{[\rho(z)-\rho(\Phi(z))]_+}|\mathbf{J}_\Psi(z)|\right\}. \qquad (3.12)$$

3.2.3 As a delayed rejection algorithm

The transition kernel of the discrete-time piecewise-deterministic Markov process (3.2) can be understood as an initial proposal $\Phi(z)$ which is accepted with probability $\alpha(z)$. If this proposal is rejected, then we sample the next state from the kernel $Q(z,\cdot)$.

In the case that α is the Metropolis-Hastings acceptance probability, and when Q itself is a Metropolis-Hastings kernel, Sherlock and Thiery (2017) point out that this is an instance of the *delayed rejection* algorithm of Tierney and Mira (1999).

The delayed rejection framework provides a way to specify a sequence of proposal distributions, $\{q_k; k \in [K]\}$. Each proposal is used in turn until a proposal is accepted, at which point the algorithm terminates. In the general formulation,

the acceptance probability of the first kernel is the standard Metropolis-Hastings acceptance probability for a single proposal. However, the acceptance probabilities associated to the subsequent kernels take a complicated form; Tierney and Mira (1999) only give the expression for first and second kernels. To express these, take $y_1 \sim q_1(x, \cdot)$ as the first proposal and $y_2 \sim q_2(x, y_1, \cdot)$ as the second proposal, and use π as the target density. They give

$$\begin{aligned} \alpha_1(x, y_1) &= \min\left\{1, \frac{\pi(y_1)q_1(y_1, x)}{\pi(x)q_1(x, y_1)}\right\} \\ \alpha_2(x, y_1, y_2) &= \min\left\{1, \frac{\pi(y_2)q_1(y_2, y_1)(1-\alpha(y_2, y_1))q_2(y_2, y_1, x)}{\pi(x)q_1(x, y_1)(1-\alpha(x, y_1))q_2(x, y_1, y_2)}\right\} \end{aligned} \tag{3.13}$$

where we see that the acceptance probability of the second proposal also involves the value of the first. The acceptance probability (3.13) is, however, not applicable to our setting as it does not accommodate a deterministic proposal.

An alternate interpretation of the acceptance probability β is that of a standard Metropolis-Hastings acceptance probability when targeting the distribution $\rho(z)(1-\alpha(z))$. Thus we suggest that this represents a sort of "remainder" form for the delayed rejection, where subsequent kernels could be developed based on this new invariant distribution.

3.3 The discrete-time bouncy particle sampler

We are now prepared to describe the discrete-time bouncy particle sampler. As with the continuous-time bouncy particle sampler, we target a density augmented by a velocity distribution. Take $\rho(x, v) = \pi(x)\psi(v)$ where $(x, v) \in \mathbb{R}^d \times \mathbb{R}^d$. For ψ, we choose a multivariate normal

$$\psi(v) = \mathcal{N}(v; 0, \epsilon^2 I_d)$$

where ϵ can be interpreted as a step size. For Φ we select a linear flow, taking $\Phi(x, v) = (x + v, v)$. And again, we choose $\mathcal{S}(x, v) = (x, -v)$ as an appropriate involution. All of the above choices mimic their counterparts in continuous time.

For the acceptance rate we apply the result (3.9) and simplify to

$$\alpha(x, v) = \min\left\{1, \frac{\pi(x+v)}{\pi(x)}\right\}.$$

And for the event kernel Q, we propose a Metropolis-Hastings kernel with a deterministic proposal taking a reflection from the current state (x, v):

$$\Psi(x, v) = (x, R(v; \nabla U(x))),$$

where R is a reflection, identical to (2.19) as used in the continuous-time bouncy particle sampler. Applying (3.12), we arrive at the following expression, where we use the fact that Ψ is volume preserving and leaves ρ invariant, and that Φ leaves ψ invariant:

$$\beta'((x, v), (x, R(v; \nabla U(x)))) = \min\left\{1, \frac{[\pi(x) - \pi(x - R(v; \nabla U(x)))]_+}{[\pi(x) - \pi(x+v)]_+}\right\}.$$

As in continuous time, we introduce an extra refreshment kernel which simply updates the velocity independently of the current state, taking $v \sim \psi(\cdot)$. This is done with some probability p_{ref}, the ideal value of which is determined experimentally, as with the λ_{ref} parameter of the continuous-time algorithm. This gives us an algorithm which is a mixture of the refreshment kernel and the discrete-time bouncy particle sampler step, this is detailed in Algorithm 12.

3.3.1 The discrete-time bouncy particle sampler as a special case of the discretized reflective slice sampler

In Neal (2003; Section 7), the reflective slice sampler is given as an extension of the slice sampler to the multidimensional case. Both the flow and the bounces used in the reflective slice sampler are identical to those used in the bouncy particle sampler. We discussed the reflective slice sampler in Chapter 2 due to its close connections with the several exiting algorithms which derive Markov chain Monte Carlo proposals by simulating continuous-time Hamiltonian dynamics.

The reflective slice sampler and its variants use an explicit slice variable u, targeting a joint density

$$\rho(x, v, u) = \mathbf{1}_{(0,\pi(x))}(u)\psi(v),$$

Algorithm 12 Discrete-time bouncy particle sampler

Given state x_t, v_t return new state x_{t+1}, v_{t+1}.

function DISCRETETIMEBPS(x_t, v_t)
  with probability p_{ref}
    $x_{t+1} \leftarrow x_t$.
    Simulate $v_{t+1} \sim \psi(\cdot)$.
  otherwise
    with probability $\min\{1, \frac{\pi(x_t+v_t)}{\pi(x_t)}\}$
      $x_{t+1} \leftarrow x_t + v_t$.
      $v_{t+1} \leftarrow v_t$.
    otherwise
      $v^* \leftarrow R(v_t; \nabla U(x_t))$.
      with probability $\min\{1, \frac{[\pi(x_t)-\pi(x_t-v^*)]_+}{[\pi(x_t)-\pi(x_t+v_t)]_+}\}$
        $x_{t+1} \leftarrow x_t$.
        $v_{t+1} \leftarrow v^*$.
      otherwise
        $x_{t+1} \leftarrow x_t$.
        $v_{t+1} \leftarrow -v_t$.
  return x_{t+1}, v_{t+1}.
end function

and make several updates of x and v while holding the slice variable u constant. The slice variable simplifies the algorithm somewhat as accept-reject decisions are based on whether proposed points lie on the slice or not. We will show that a variant of this algorithm, wherein the slice variable is updated before each update of x and v, is equivalent to the discrete-time bouncy particle sampler (Algorithm 12). This variation is explicitly mentioned by Neal (2003).

Reflective slice sampling uses a forward proposal $\Phi(z) = (x + \epsilon v, v)$ for some $\epsilon > 0$ — this is equivalent to our construction using a velocity with standard deviation ϵ, so we continue assuming $\epsilon = 1$.

For the event a deterministic proposal is used. This kernel modifies only the velocity, so in keeping with our terminology of Section 2.5. Two types of bounces are considered:

1. *inner reflections* which takes $\Psi_{\text{inner}}(x, v) = (x, R(v; \nabla U(x))$, while

2. *outer reflections* uses $\Psi_{\text{outer}}(x, v) = (x+v+R(v; \nabla U(x+v)), R(v; \nabla U(x+v)))$. This is simply a composition of $\Psi_{\text{outer}} = \Phi \circ \Psi_{\text{inner}} \circ \Phi$.

Both proposals satisfy $\Psi^{-1} = \mathcal{S} \circ \Psi \circ \mathcal{S}$. The outer reflection operator has been recently proposed independently in Sherlock and Thiery (2017); see also Skilling (2012) for a related proposal in the context of nested sampling. The outer reflection can also be seen as a special case of a more general scheme described in Neal (2003), which we identify as a form of *tunnelling* in Section 3.8.

The discretized reflective slice sampler algorithm is given in Algorithm 13. The main distinction between this discretized reflective slice sampler algorithm and the discrete-time bouncy particle sampler is that the reflective slice sampler makes use of a fixed slice variable which is unchanged over several steps of the algorithm.

We claim that the bouncy particle sampler is equivalent to a step of reflective slice sampler if the slice variable in the reflective slice sampler is updated before the step. To make this claim, we must establish that using a slice variable is equivalent to using the acceptance probabilities (3.9) and (3.12).

First, note that if the slice variable is always updated before the deterministic step, we can consider the combination of these two steps as a single kernel. Whether we use a slice variable, as in the reflective slice sampler, or not, as in the bouncy particle sampler, the kernel can be expressed as a mixture over the three possible outcomes of accepting the forward move $x + v$, accepting the bounce, or rejecting both:

$$K(x, v, \mathrm{d}x', \mathrm{d}v') = \min\left\{1, \frac{\pi(x+v)}{\pi(x)}\right\} \delta_{x+v}(\mathrm{d}x')\delta_v(\mathrm{d}v') + a_{\text{bounce}}\delta_{x^*}(\mathrm{d}x')\delta_{v^*}(\mathrm{d}v') + a_{\text{reject}}\delta_x(\mathrm{d}x')\delta_{-v}(\mathrm{d}v').$$

where $\min\{1, \pi(x+v)/\pi(x)\} + a_{\text{bounce}} + a_{\text{reject}} = 1$. For both algorithms the probability of accepting the forward proposal is $\min(1, \pi(x + \epsilon v)/\pi(x))$. The algorithms differ, however, in the value of a_{bounce}, as follows:

- for the reflective slice sampler, updating a slice variable $u \sim \text{Uniform}(0, \pi(x))$ then taking a single step of the reflective slice sampler algorithm (Algorithm 13

for $K = 1$) yields

$$\begin{aligned} a_{\text{bounce,RSS}} &= \int_0^{\pi(x)} \frac{1}{\pi(x)} \mathbb{I}(u > \pi(x+v)) \mathbb{I}(u < \pi(x^*)) \mathbb{I}(u > \pi(x^* - v^*)) \text{Leb}(\mathrm{d}u) \\ &= \frac{1}{\pi(x)} \left[\min\left\{ \pi(x), \pi(x^*) \right\} - \max\left\{ \pi(x+v), \pi(x^* - v^*) \right\} \right]_+ , \end{aligned}$$

whereas

- in the bouncy particle sampler, using the probability (3.12) yields

$$\begin{aligned} a_{\text{bounce,BPS}} &= \left[1 - \min\left\{ 1, \frac{\pi(x+v)}{\pi(x)} \right\} \right] \min\left\{ 1, \frac{[\pi(x^*) - \pi(x^* - v^*)]_+}{[\pi(x) - \pi(x+v)]_+} \right\} \\ &= \frac{1}{\pi(x)} \left[\min\left\{ \pi(x) - \pi(x+v), \pi(x^*) - \pi(x^* - v^*) \right\} \right]_+ . \end{aligned}$$

Comparing these probabilities of accepting a bounce proposal, we see that in general the probability of accepting a bounce in the bouncy particle sampler, which does not use a slice variable, is at least that of the reflective slice sampler scheme, which does use a slice variable. However, it is easy to verify that the two algorithms are equivalent for both inside and outside reflections:

- for inside reflection we have $x^* = x$ and

$$\begin{aligned} a_{\text{bound,RSS}} &= \frac{1}{\pi(x)} \left[\pi(x) - \max\left\{ \pi(x+v), \pi(x^* - v^*) \right\} \right]_+ \\ a_{\text{bound,BPS}} &= \frac{1}{\pi(x)} \left[\pi(x) - \max\left\{ \pi(x+v), \pi(x^* - v^*) \right\} \right]_+ , \end{aligned}$$

- whereas for outside reflection we have $x + v = x^* - v^*$ and

$$\begin{aligned} a_{\text{bound,RSS}} &= \frac{1}{\pi(x)} \left[\min\left\{ \pi(x), \pi(x^*) \right\} - \pi(x+v) \right]_+ \\ a_{\text{bound,BPS}} &= \frac{1}{\pi(x)} \left[\min\left\{ \pi(x), \pi(x^*) \right\} - \pi(x+v) \right]_+ . \end{aligned}$$

Thus we claim that the reflective slice sampler algorithm represents an implementation of the discrete-time version of the bouncy particle sampler process.

Algorithm 13 A single step of the discretized reflective slice sampler

function DiscretizedReflectiveSliceSampler(x_t, v_T)
  Given current state x_t, v_t.
  Sample $u \sim \text{Uniform}(0, \pi(x_t))$.
  $x'_0 \leftarrow x_t$.
  $v'_0 \leftarrow v_t$.
  for $k \in \{1, \ldots, K\}$ **do**
    if $\pi(x'_{k-1} + v'_{k-1}) > u$ **then**
      $x'_k \leftarrow x'_{k-1} + v'_{k-1}$.
      $v'_k \leftarrow v'_{k-1}$.
    else
      $(x^*, v^*) \leftarrow \Psi(x'_{k-1}, v'_{k-1})$.
      if $\pi(x^*) > u$ and $\pi(x^* - v^*) < u$ **then**
        $x'_k \leftarrow x^*$.
        $v'_k \leftarrow v^*$.
      else
        $x'_k \leftarrow x'_{k-1}$.
        $v'_k \leftarrow -v'_{k-1}$.
      end if
    end if
  end for
  return x'_K, v'_K.
end function

3.4 Decomposable rates in discrete time

In Section 2.2.1 we extended the continuous-time invariance conditions to allow for multiple event types. We now prove a generalization of Assumption A2 which is analogous.

First, we draw an analogy between continuous-time rates and discrete-time acceptance probabilities. Consider the discrete-time skew-detailed balance condition A3.2,

$$\rho(z)\alpha(z)|\mathbf{J}_\Phi(z)|^{-1} = \rho(\mathcal{S} \circ \Phi(z))\alpha(\mathcal{S} \circ \Phi(z))$$

and rewrite it as

$$\log \alpha(\mathcal{S} \circ \Phi(z)) - \log \alpha(z) = -\log |\mathbf{J}_\Phi(z)| + H(\mathcal{S} \circ \Phi(z)) - H(z) \tag{3.14}$$

by applying the negative log and rearranging terms. This assumes $\rho(z) > 0$ everywhere however these developments will not require this condition as we can

account for regions of zero density by having a special case for the acceptance probability, see Tierney (1994). Recalling the continuous-time rate condition (2.6)

$$\lambda(\mathcal{S}(z)) - \lambda(z) = \nabla \cdot \phi(z) - \langle \nabla H(z), \phi(z) \rangle$$

we draw a direct comparison to (3.14), suggesting that $\lambda(z)$ is analogous to $-\log \alpha(z)$.

We now extend the acceptance probability to a "decomposable" acceptance probability, as we did with decomposable continuous-time rates. Define an overall acceptance probability

$$\alpha(z) = \exp\left\{ -\int_{\Omega} [-\log \alpha_{\omega}(z)] \mu(\mathrm{d}\omega) \right\} \tag{3.15}$$

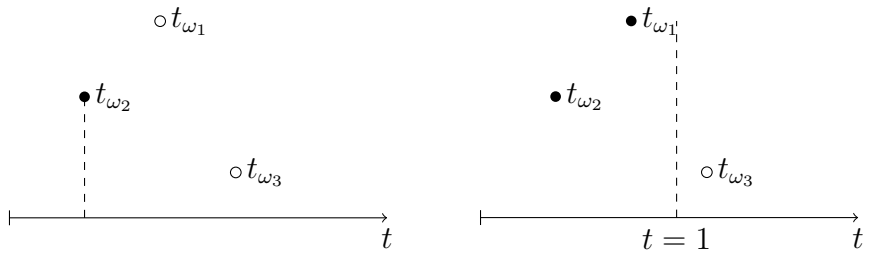
where $\alpha_{\omega}(z) \in (0, 1]$ for all $\omega \in \Omega$ and where μ is assumed to be a finite measure. One may interpret this as the void probability of a Poisson process with rate $-\log \alpha_{\omega}(z)\mu(\mathrm{d}\omega)$. Thus a corresponding algorithm would be to simulate a Poisson process with rate $-\log \alpha_{\omega}(z)$, and reject the proposal if the Poisson process contains any positive number of points.

Denote a realization of the Poisson process as P. For a Poisson process on Ω with rate $-\log \alpha_{\omega}(z)\mu(\mathrm{d}\omega)$, denote the law of this process $\mathbb{Q}_z(\mathrm{d}P)$. Take $\mathbb{Q}(\mathrm{d}P)$ as the law of the process with rate $\mu(\mathrm{d}\omega)$, then $\mathbb{Q}_z$ has a Radon-Nykodym derivative with respect to the latter of

$$\mathcal{Q}_z(P) = \frac{\mathrm{d}\mathbb{Q}_z(P)}{\mathrm{d}\mathbb{Q}(P)} = \prod_{\omega \in P} [-\log \alpha_{\omega}(z)].$$

In the continuous-time case, we are able to attribute an event to a single ω, and then select our event kernel based on this ω. In the discrete-time case, we instead have a *set* of distinct values of ω which are the points of the Poisson process P. To further motivate this, consider taking a fixed-time window of the continuous algorithm; there, we might expect that multiple events could occur, each associated with different values of ω. From this viewpoint, it is perhaps intuitive that a discrete-time algorithm should associate a rejection with multiple values of ω. See Figure 3.1 for an illustration. Of course, the discrete-time events do not

Figure 3.1: An illustration of how events are treated in continuous time (left) and in discrete time (right). In both cases, we have an event time associated to each element of Ω; here $\Omega = \{\omega_1, \omega_2, \omega_3\}$. In continuous-time, an event is triggered by the first arriving time t_{ω_2}, and so the event will be simulated from Q_{ω_2}. In discrete-time, all events from a fixed window $t \in (0, 1)$ are included and the event kernel Q_P is associated to the subset $P \subseteq \Omega$; in this case $P = \{\omega_2, \omega_3\}$.



actually occur at continuous times; we must emphasize that this reasoning is not meant to establish a precise relationship between the continuous- and discrete-time algorithms, but merely an appeal to intuition;

Hence if the number of points is null, i.e. $|P| = 0$, then we will set $z' \leftarrow \Phi(z)$. If $|P| \geq 1$, that is $P \in \mathcal{P}_{>0}$ where $\mathcal{P}_{>0}$ is the set of configurations of the Poisson process having at least one point, then we will sample $z' \sim Q(z, \cdot)$ where

$$Q(z, \mathrm{d}z') = \int_{P \in \mathcal{P}_{>0}} \frac{\mathcal{Q}_z(P)}{1 - \mathcal{Q}_z(\emptyset)} Q_P(z, \mathrm{d}z') \mathbb{Q}(\mathrm{d}P). \tag{3.16}$$

In this expression Q_P is a Markov kernel and $\mathcal{Q}_z(P)/(1 - \mathcal{Q}_z(\emptyset))$ is the density of the Poisson process conditioned upon the event that $|P| > 0$.

(**A4**) Conditions on ϕ, $\{\alpha_\omega : \omega \in \Omega\}$ and $\{Q_P : P \in \mathcal{P}_{>0}\}$

1. There exists a ρ-preserving mapping $\mathcal{S} : \mathcal{Z} \to \mathcal{Z}$.

2. The "acceptance probabilities" $\{\alpha_\omega \in (0, 1] : \omega \in \Omega\}$ satisfy

$$\int \left[\{-\log \alpha_\omega(\mathcal{S} \circ \Phi(z))\} - \{-\log \alpha_\omega(z)\}\right] \mu(\mathrm{d}\omega) = \log |\mathbf{J}_\Phi(z)| - \{H(\Phi(z)) - H(z)\}. \tag{3.17}$$

3. For all $P \in \mathcal{P}_{>0}$, the transition kernel Q_P satisfies

$$\int \rho(\mathrm{d}z) \mathcal{Q}_z(P) Q_P(z, \mathrm{d}z') = \rho(\mathcal{S}^{-1}(\mathrm{d}z')) \mathcal{Q}_{\mathcal{S}(z')}(P). \tag{3.18}$$

Proposition 3.2. *Assume A4. Then the discrete-time piecewise-deterministic Markov process admits ρ as an invariant distribution.*

To show correctness, we demonstrate that the kernel satisfies global balance. Assume that ρ admits a density with respect to the Lebesgue measure which we denote $\rho(z)$, and similarly, $Q_P(z, z')$. We now write the density of the transition kernel as a mixture over possible values of P:

$$K(z, z') = \mathcal{Q}_z(\emptyset)\delta_{\Phi(z)}(z') + \int_{P\in\mathcal{P}_{>0}} \mathcal{Q}_z(P)Q_P(z, z').$$

Plugging into the global balance equation,

$$\begin{aligned}
&\int \rho(z)K(z, z')\mathrm{Leb}(\mathrm{d}z) \\
&= \int \rho(z)\mathcal{Q}_z(\emptyset)\delta_{\Phi(z)}(z')\mathrm{Leb}(\mathrm{d}z) + \int_{z\in\mathcal{Z}}\int_{P\in\mathcal{P}_{>0}} \rho(z)\mathcal{Q}_z(P)Q_P(z, z')\mathrm{Leb}(\mathrm{d}z') \\
&= \rho(\Phi^{-1}(z'))\mathcal{Q}_{\Phi^{-1}(z')}(\emptyset)|\mathbf{J}_{\Phi^{-1}}(z')| + \rho(\mathcal{S}(z'))\int_{P\in\mathcal{P}_{>0}} \mathcal{Q}_{\mathcal{S}(z')}(P) \\
&= \rho(\Phi^{-1}(z'))\mathcal{Q}_{\Phi^{-1}(z')}(\emptyset)|\mathbf{J}_{\Phi^{-1}}(z')| + \rho(\mathcal{S}(z'))(1 - \mathcal{Q}_{\mathcal{S}(z')}(\emptyset)).
\end{aligned}$$

Thus it is sufficient that $\rho(\Phi^{-1}(z'))\mathcal{Q}_{\Phi^{-1}(z')}(\emptyset)|\mathbf{J}_{\Phi^{-1}}(z')| - \rho(\mathcal{S}(z'))\mathcal{Q}_{\mathcal{S}(z')}(\emptyset) = 0$. Replace $z' = \Phi(z)$ and rewrite, also noting that $|\mathbf{J}_{\Phi^{-1}}(\Phi(z))| = |\mathbf{J}_\Phi(z)|^{-1}$,

$$\frac{\mathcal{Q}_z(\emptyset)}{\mathcal{Q}_{\mathcal{S}\circ\Phi(z)}(\emptyset)} = |\mathbf{J}_\Phi(z)|\frac{\rho(\mathcal{S}\circ\Phi(z))}{\rho(z))}$$

$$\begin{aligned}
\exp\left\{-\int[-\log\alpha_\omega(z)]\mu(\mathrm{d}\omega) + \int[-\log\alpha_\omega(\mathcal{S}\circ\Phi(z))]\mu(\mathrm{d}\omega)\right\} \\
= \exp\left\{\log|\mathbf{J}_\Phi(z)| - H(\mathcal{S}\circ\Phi(z)) + H(z)\right\}
\end{aligned}$$

which is satisfied by A4.2.

For those settings where we will decompose H itself, we introduce the notation

$$H(z) = \int h_\omega(z)\mu(\mathrm{d}\omega)$$

which in those cases will allow us to express an α_ω sufficient for A4.2 by equating the integrands.

3.4.1 For measures containing atoms

In the previous section, we assumed that the measure μ was non-atomic. Here we consider the case where μ may contain atoms and propose a new interpretation for such a case; this will help bridge to the case where μ is strictly atomic, which we discuss in the next section.

The essential idea here is to recognize that, when simulating $\alpha(z)$ as in (3.15), a rejection occurs if the Poisson process contains a non-zero number of points. In the case that Ω contains atoms, it may be sensible to implement this simulation as a Poisson process for each atom. When doing so, the points of the Poisson process P associated with an atom are in a sense redundant; simulating the event that we have one or more points of P associated with an individual atom is sufficient. While this approach may or may not yield any computational benefit, it will allow us to draw important connections to other algorithms and other literature.

Some descriptions of the Poisson process forbid the possibility of multiple points associated to the same value; to avoid any such issues, we simulate here a Poisson process $\tilde{P}$ on $\Omega \times \mathbb{R}_+$ with rate $\mathbf{1}_{(0,-\log \alpha_\omega(z))}(y)\mu(\mathrm{d}\omega)\mathrm{Leb}(\mathrm{d}y)$, which projected onto Ω is equivalent to the rate we used in the non-atomic case.

Allow Ω_A to represent the subset of Ω which are atoms of μ. Without loss of generality let the measure of any atom be 1, i.e. $\mu(\{\omega\}) = 1$ for any $\omega \in \Omega_A$ (this is always possible as for any α and μ we can replace these with α' such that $-\log \alpha'_\omega(z) = -\log \alpha_\omega(z)\mu(\{\omega\})$ and $\mu'(\{\omega\}) = 1$). The probability of an atom $\omega \in \Omega_A$ having no associated points in the Poisson process $\tilde{P}$, that is, that $\tilde{P} \cap \{\omega, \mathbb{R}_+\} = \emptyset$ is

$$\exp\left(-\int_0^{-\log \alpha_\omega(z)\mu(\{\omega\})} \mathrm{Leb}(\mathrm{d}y)\right) = \alpha_\omega(z).$$

From this, we can rewrite the overall acceptance probability, which is the void probability of $\tilde{P}$, as

$$\alpha(z) = \exp\left(-\int_{\Omega\setminus\Omega_A} [-\log \alpha_\omega(z)]\mu(\mathrm{d}\omega)\right) \times \prod_{\omega\in\Omega_A} \alpha_\omega(z). \tag{3.19}$$

This expression is a combination of the void probability of a Poisson process and a number $|\Omega_A|$ of independent accept-reject tests.

We define two new objects which can be simulated to determine an acceptance:

1. P_D, to express the realization of the Poisson process limited to $\Omega \setminus \Omega_A$, the law of which is simply that of a Poisson process restricted to $\Omega \setminus \Omega_A$ with rate $-\log \alpha_\omega(z)\mu(\mathrm{d}\omega)$; and

2. $\{B_\omega : \omega \in \Omega_A\}$ to express the rejections associated with the atoms Ω_A, with distribution

$$B_\omega \sim \text{Bernoulli}(1 - \alpha_\omega(z)).$$

If $P_D = \emptyset$ and $B_\omega = 0$ for all $\omega \in \Omega_A$, then the proposal $\Phi(z)$ is accepted.

3.4.2 For strictly atomic measures

The case for strictly atomic measures is merely a special case of the above, but its importance warrants explicit presentation.

Here we assume Ω is finite and represent this as an index set $\Omega = \Omega_A = [N]$. While this remains a special case of the above, we henceforth use the subscript i (instead of ω as before) to emphasize that Ω is a simple index set. The acceptance probability adapted from (3.19) now has the form

$$\alpha(z) = \prod_{i=1}^{N} \alpha_i(z).$$

This can be seen as a sort of "unanimous" acceptance probability, as it is equivalent to performing an accept-reject test for each α_i and only accepting the move only if each individual factor successfully accepts.

We note that this case is closely connected to the *delayed acceptance* algorithm (Ceperley 1995, Christen and Fox 2005). In that algorithm, one typically has access to a cheap approximation of ρ. This approximation can be considered as a particular factorization, expressing the energy as

$$H(z) = \underbrace{\tilde{H}(z)}_{H_1(z)} + \underbrace{H(z) - \tilde{H}(z)}_{H_2(z)}$$

and thus with $\rho(z) = \exp(-H(z))$ and $\tilde{\rho}(z) = \exp(-\tilde{H}(z))$ then we may take

$$\alpha(z) = \underbrace{\min\left\{1, \frac{\tilde{\rho}(\Phi(z))}{\tilde{\rho}(z)}\right\}}_{\alpha_1(z)} \underbrace{\min\left\{1, \frac{\rho(\Phi(z))}{\rho(z)}\frac{\tilde{\rho}(z)}{\tilde{\rho}(\Phi(z))}\right\}}_{\alpha_2(z)}.$$

By first performing an accept-reject step with probability α_1, a rejection of this precludes the need to evaluate α_2. If additionally $\rho(z)/\tilde{\rho}(z)$ is approximately one, then α_2 will be near to one and the overall acceptance probability will be near to the ideal Metropolis-Hastings probability.

Delayed acceptance has also been employed in the large dataset setting (Banterle et al. 2015), where for a posterior $\pi(x) \propto \pi_0(x)\prod_{i=1}^N \pi(y_i|x)$ (where y_i are the data, x are the parameters, and π_0 is the prior). We discuss this setting at length in Chapter 4.

3.4.3 Algorithmic descriptions

We provide a description of the algorithms for the cases above, where μ is non-atomic (Algorithm 14) and where μ is strictly atomic (Algorithm 15). Allow B to represents the per-ω rejections variables as defined in Section 3.4.1; here we index this by i to emphasize that this is a finite set, and we replace Q_P by Q_B. For the atomic case, we have that Q_B must satisfy

$$\int \rho(\mathrm{d}z)\left[\prod_{i=1}^N \text{Bernoulli}(B_i; 1-\alpha_i(z))\right] Q_B(z, \mathrm{d}z') = \rho(\mathcal{S}^{-1}(\mathrm{d}z'))\left[\prod_{i=1}^N \text{Bernoulli}(B_i; 1-\alpha_i(\mathcal{S}(z')))\right]. \quad (3.20)$$

For each of these algorithms, we can quite simply derive a Metropolis-Hastings equivalent by defining a joint density over the current state and the proposal, as done in Section 1.2.3. Thus the decomposition is based on this joint distribution, and we have

$$\pi(x)q(x^*|x) \propto \exp\left(-\int h_\omega(x, x^*)\mu(\mathrm{d}\omega)\right).$$

The case of a symmetric proposal is similar, where we can take merely $h_\omega(x, x^*) = h_\omega(x)$. For the atomic case we have $\pi(x)q(x^*|x) \propto \prod_i \pi_i(x, x^*)$. See Algorithms 16

Algorithm 14 A single step of discrete-time piecewise-deterministic Markov chain Monte Carlo for a non-atomic μ.

Given current state z, return next state z'.

function DiscreteTimePDMCMC-NonAtomic(z)
  Simulate a Poisson process P on Ω with rate

$$[h_\omega(\Phi(z)) - h_\omega(z)]_+ \mu(\mathrm{d}\omega).$$

  if $P = \emptyset$ **then**
    return $\Phi(z)$.
  else
    Simulate $z^* \sim Q_P(z, \cdot)$ where Q_P satisfies (3.18).
    return z^*.
  end if
end function

Algorithm 15 A single step of discrete-time piecewise-deterministic Markov chain Monte Carlo for an atomic μ.

Given current state z, return next state z'.

function DiscreteTimePDMCMC-Atomic(z)
  Simulate Bernoulli variables B_i for $i \in [N] = \Omega$ with distribution

$$B_i \sim \text{Bernoulli}\left(1 - \min\left\{1, \frac{\rho_i(\Phi(z))}{\rho_i(z)}\right\}\right).$$

  if $B_i = 0$ for all i **then**
    return $\Phi(z)$.
  else
    Simulate $z^* \sim Q_B(z, \cdot)$ where Q_B satisfies (3.18).
    return z^*.
  end if
end function

for the non-atomic case and 17 for the atomic. We note that Algorithm 17 is equivalent to delayed acceptance.

3.4.4 A discrete-time zig-zag sampler.

The developments of this chapter allow us to suggest that a discrete-time version of the zig-zag process could be constructed based on the per-dimension acceptance rates

$$\alpha_i(x, v) = \min\left\{1, \frac{\pi(x_i + v_1, \ldots, x_i + v_i, x_{i+1}, \ldots, x_n)}{\pi(x_1 + v_1, \ldots, x_{i-1} + v_{i-1}, x_i, \ldots, x_n)}\right\}$$

Algorithm 16 A single step of decomposed Metropolis-Hastings for a non-atomic μ.

Given current state x, return next state x'.

function MetropolisHastings-NonAtomic(x)
  Simulate proposal $x^* \sim q(\cdot|x)$.
  Simulate a Poisson process P with rate

$$[h_\omega(x^*, x) - h_\omega(x, x^*)]_+ \mu(\mathrm{d}\omega).$$

  if $P = \emptyset$ **then**
    return x^*.
  else
    return x.
  end if
end function

Algorithm 17 A single step of decomposed Metropolis-Hastings for an atomic μ.

Given current state x, return next state x'.

function MetropolistHastings-Atomic(z)
  Sample $x^* \sim q(\cdot|x)$.
  Simulate Bernoulli variables B_i for $i \in [N] = \Omega$ with distribution

$$B_i \sim \text{Bernoulli}\left(1 - \min\left\{1, \frac{\pi_i(x^*, x)}{\pi_i(x, x^*)}\right\}\right).$$

  if $B_i = 0$ for all i **then**
    return x^*.
  else
    return x.
  end if
end function

for $i \in [d]$. This is correct if care is taken to appropriately match the ordering in which the variables are updated, or by e.g. picking a random permutation in each iteration. It seems unlikely that a discrete-time algorithm using this decomposition would be useful, as compared with a standard Metropolis-Hastings algorithm proposing $x + v$ it increases the computational cost of an iteration to $n - 1$ potential evaluations and decreases the probability of accepting the overall move.

3.5 Decomposable acceptance probabilities, slices, and auxiliary variables

Here we develop several extensions to the decomposable acceptance probabilities of Section 3.4 which will be relevant in discrete time. These extensions will relate to a wide range of algorithms, including those applied to posterior simulations over diffusions and Gaussian processes, and to algorithms attaining sparsity in big data. While it may be tempting to claim that the decomposable-rate framework serves as a sort of "master" algorithm which generalizes a wide range of algorithms, we instead suggest that the algorithms developed for the relevant applications already incorporate several improvements beyond the decomposable framework, and therefore the framework is not beneficial in those settings. Nonetheless we suggest that these connections are important in understanding the close relationship between all of these algorithms, and also the potential utility of any attempts to apply the framework.

In Section 1.2.7, we showed how the Metropolis-Hastings acceptance probability can be recovered from a deterministic accept-reject algorithm wherein a slice variable is simulated in each step; see Algorithm 2. We revisit this here, and show how the decomposable rates may be connected to a class of auxiliary variable algorithms.

In the "slice" interpretation of Metropolis-Hastings, we operate on the density augmented by both a proposal state x^* and also by a slice variable u, i.e. $\pi(x, x^*, u) = \mathbb{I}(u < \pi(x)q(x, x^*))$. A single step of Metropolis-Hastings is interpreted as a sequence of three substeps

1. Refresh the proposal by taking $x^* \sim q(x, \cdot)$.

2. Simulate the slice variable $u \sim \text{Uniform}(0, \pi(x)q(x, x^*))$.

3. Swap x and x^* if $u < \pi(x^*)q(x^*, x)$.

This understanding is instructive, as the interpretation involving slices and a proposal-augmented density can be useful when constructing and validating complex transition schemes.

We also note that the slice interpretation can have some computational benefits. The most obvious example is that of slice sampling (Neal 2003), demonstrating that incorporating the slice variable u into the density explicitly has a computational benefit in that we can sometimes find efficient ways to compute the density of x conditional on the slice.

Motivated by this, we derive new "slice variables" by interpreting the objects simulated in the acceptance step as auxiliary variables. We show that these slice variables are closely connected to existing algorithms in the literature.

3.5.1 Slices for non-atomic measures

To simplify exposition, we will henceforth take as our target density $\pi(x) = \exp(-U(x))$, and assume a symmetric proposal distribution; the necessary extensions to accommodate an asymmetric proposal density (as in the previous section) are straightforward.

We begin with the following sketch for an algorithm. First, whereas the framework of Section 3.4 allows us to define a decomposition for the change in energy due to the proposal $H(\mathcal{S} \circ \Phi(z)) - H(z)$, we restrict ourselves to the setting where we define a decomposition for $U(x)$ alone, taking

$$U(x) = \int_\Omega h_\omega(x)\mu(\mathrm{d}\omega).$$

Assuming a symmetric proposal, a hypothetical "slice" version of Algorithm 14 consists of three steps:

1. Simulate a Poisson process $\tilde{P}$ on $\Omega \times \mathbb{R}$ with rate

$$\mathbb{I}(y > h_\omega(x))\mathrm{Leb}(\mathrm{d}y)\mu(\mathrm{d}\omega).$$

2. Simulate a proposal $x^* \sim q(x, \cdot)$.

3. Accept x^* as the next sample if $y > h_\omega(x^*)$ for all $(\omega, y) \in \tilde{P}$. Otherwise, retain x as the next sample.

Together the last two steps are equivalent to accepting a swap when a simulation of the Poisson process with rate $[h_\omega(x^*) - h_\omega(x)]_+ \mu(\mathrm{d}\omega)$ yields a null process, as used in Algorithm 16.

Our claim is that the process $\tilde{P}$ functions in the same way as a slice, and that the conditional of x on $\tilde{P}$ is the uniform distribution over all x where the condition $y > h_\omega(x^*)$ is true for all $(y, \omega) \in \tilde{P}$. To establish this properly, we will in the next section define a joint density over x and P by modifying our construction to ensure that the point process is finite.

3.5.2 Upper bounds for $h_\omega(x)$

The use of the rate $\mathbb{I}(y > h_\omega(x))\mathrm{Leb}(\mathrm{d}y)\mu(\omega)$ implies that $\tilde{P}$ contains an infinite number of points so the above algorithm is not implementable. The following extension allows us to ensure almost-surely finite P. Take an upper bound $\bar{h}_\omega(x)$ which satisfies:

1. $\bar{h}_\omega(x) \geq h_\omega(x)$ for all x and ω, and
2. $\bar{U}(x) = \int \bar{h}_\omega(x)\mu(\mathrm{d}\omega)$ is computable for all x.

The adapted algorithm is

1. Simulate a Poisson process $\tilde{P}$ on $\Omega \times \mathbb{R}$ with rate
$$\mathbb{I}(0 \leq y \leq \bar{h}_\omega(x) - h_\omega(x))\mathrm{Leb}(\mathrm{d}y)\mu(\mathrm{d}\omega).$$
2. Simulate a proposal $x^* \sim q(x, \cdot)$.
3. If $y \leq \bar{h}_\omega(x^*) - h_\omega(x^*)$ for all $(y, \omega) \in \tilde{P}$, accept x^* as the next sample with probability
$$\exp\left(-\left[\int \bar{h}_\omega(x^*) - \bar{h}_\omega(x)\mu(\mathrm{d}\omega)\right]_+\right).$$
Otherwise, retain x as the next sample.

The algorithm implements an acceptance probability conditional on $\tilde{P}$, which we can write

$$\alpha((x,P),(x^*,P)) = \min\left\{1, \frac{\exp\left(-\int \bar{h}_\omega(x^*)\mu(\mathrm{d}\omega\right)}{\exp\left(-\int \bar{h}_\omega(x)\mu(\mathrm{d}\omega)\right)}\right\} \prod_{(\omega,y)\in P} \mathbb{I}(0 < y < \bar{h}_\omega(x^*) - h_\omega(x^*)). \tag{3.21}$$

The algorithm satisfies detailed balance, which we now demonstrate. First, note that the probability of the condition $\mathbb{I}(y \leq \bar{h}_\omega(x^*) - h_\omega(x^*))$ holding for all $(\omega, y) \in \tilde{P}$ is the void probability of the region above $\bar{h}_\omega(x^*) - h_\omega(x^*)$ but below $\bar{h}_\omega(x) - h_\omega(x)$:

$$\exp\left(-\int [(\bar{h}_\omega(x) - h_\omega(x)) - (\bar{h}_\omega(x^*) - h_\omega(x^*))]_+ \mu(\mathrm{d}\omega)\right).$$

Therefore the total acceptance probability, having marginalized out $\tilde{P}$, is

$$\begin{aligned}\alpha(x,x^*) = &\exp\left(-\int [(\bar{h}_\omega(x) - h_\omega(x)) - (\bar{h}_\omega(x^*) - h_\omega(x^*))]_+ \mu(\mathrm{d}\omega)\right) \\ &\times \exp\left(-\left[\int [\bar{h}_\omega(x^*) - \bar{h}_\omega(x)]\mu(\mathrm{d}\omega)\right]_+\right)\end{aligned}$$

Then taking $\pi(x)\alpha(x,x^*)$ (recall that q is assumed to be symmetric), we have

$$\begin{aligned}\rho(x)\alpha(x,x^*) = \exp\Bigg(&- \int h_\omega(x)\mu(\mathrm{d}\omega) \\ &- \int [(\bar{h}_\omega(x) - h_\omega(x)) - (\bar{h}_\omega(x^*) - h_\omega(x^*))]_+ \mu(\mathrm{d}\omega) \\ &- \left[\int [\bar{h}_\omega(x^*) - \bar{h}_\omega(x)]\mu(\mathrm{d}\omega)\right]_+\Bigg).\end{aligned}$$

Noting that $[a-b]_+ = [a-b]_+ + b - b = \max(a,b) - b$,

$$\begin{aligned}&h_\omega(x) + [(\bar{h}_\omega(x) - h_\omega(x)) - (\bar{h}_\omega(x^*) - h_\omega(x^*))]_+ \\ &= h_\omega(x) + [(h_\omega(x^*) - \bar{h}_\omega(x^*)) - (h_\omega(x) - \bar{h}_\omega(x))]_+ \\ &= h_\omega(x) + \max\left\{h_\omega(x^*) - \bar{h}_\omega(x^*), h_\omega(x) - \bar{h}_\omega(x)\right\} - h_\omega(x) + \bar{h}_\omega(x) \\ &= \max\left\{h_\omega(x^*) - \bar{h}_\omega(x^*), h_\omega(x) - \bar{h}_\omega(x)\right\} + \bar{h}_\omega(x).\end{aligned}$$

Thus,

$$\begin{aligned}\rho(x)\alpha(x,x^*) = \exp\Bigg(&- \int \max\left\{h_\omega(x^*) - \bar{h}_\omega(x^*), h_\omega(x) - \bar{h}_\omega(x)\right\}\mu(\mathrm{d}\omega) \\ &- \int \bar{h}_\omega(x)\mu(\mathrm{d}\omega) \\ &- \left[\int \bar{h}_\omega(x^*) - \bar{h}_\omega(x)]\mu(\mathrm{d}\omega)\right]_+ \Bigg) \\ = \exp\Bigg(&- \int \max\left\{h_\omega(x^*) - \bar{h}_\omega(x^*), h_\omega(x) - \bar{h}_\omega(x)\right\}\mu(\mathrm{d}\omega) \\ &- \max\left\{\int \bar{h}_\omega(x^*)\mu(\mathrm{d}\omega), \int \bar{h}_\omega(x)\mu(\mathrm{d}\omega)\right\}\Bigg)\end{aligned}$$

which is a symmetric function of x and x^*, as so $\rho(x)\alpha(x,x^*) = \rho(x^*)\alpha(x^*,x)$.

To show that $\tilde{P}$ can be considered an auxiliary variable we give the joint density of x and $\tilde{P}$. We construct our density as a Radon-Nykodym derivative with respect to a unit-rate Poisson process, similarly to Wang et al. (2019). First, assume the existence of an upper bound on $\bar{h}_\omega(x) - h_\omega(x)$ for all x, call this M_ω. Assume that for $\mathcal{O} = \{(\omega, y) \in \Omega \times \mathbb{R} : 0 < y < M_\omega\}$ that $(\mu \times \mathrm{Leb})(\mathcal{O}) < \infty$. Define the measure of the Poisson process on $\mathcal{O}$ with rate $\mathrm{Leb}(\mathrm{d}y)\mu(\mathrm{d}\omega)$ as $\tilde{\mathbb{Z}}$.

We give the density of this joint with respect to the product measure $\mathbb{X} \times \tilde{\mathbb{Z}}$, where $\mathbb{X}$ is the measure on $\mathcal{X}$ on which we usually define $\pi(x)$:

$$\frac{\mathrm{d}\pi}{\mathrm{d}\mathbb{X} \times \tilde{\mathbb{Z}}}(x, \tilde{P}) = \exp\left(-\int \bar{h}_\omega(x)\mu(\mathrm{d}\omega)\right) \prod_{(\omega,y)\in P} \mathbb{I}(0 < y < \bar{h}_\omega(x) - h_\omega(x)) \quad (3.22)$$

To show that this yields π as a marginal, take $\tilde{\mathcal{P}}$ as the space of all Poisson processes on Q,

$$\begin{aligned}\int_{\tilde{\mathcal{P}}} \pi(x,\tilde{P})\tilde{\mathbb{Z}}(\mathrm{d}\tilde{P}) &= \exp\left(-\int \bar{h}_\omega(x)\mu(\mathrm{d}\omega)\right) \int_{\tilde{\mathcal{P}}} \prod_{(\omega,y)\in\tilde{P}} \mathbb{I}(0 \le y \le \bar{h}_\omega(x) - h_\omega(x))\tilde{\mathbb{Z}}(\mathrm{d}\tilde{P}) \\ &= \exp\left(-\int \bar{h}_\omega(x)\mu(\mathrm{d}\omega)\right) \mathbb{P}_{\tilde{\mathbb{Z}}}[\tilde{P} \cap \{(\omega,y) : y > \bar{h}_\omega(x) - h_\omega(x)\} = \emptyset] \\ &= \exp\left(-\int \bar{h}_\omega(x)\mu(\mathrm{d}\omega)\right) \exp\left(-\int [M_\omega - (\bar{h}_\omega(x) - h_\omega(x))]\mu(\mathrm{d}\omega)\right) \\ &= \exp\left(-\int h_\omega(x)\mu(\mathrm{d}\omega)\right) \exp\left(-\int M_\omega\mu(\mathrm{d}\omega)\right) \propto \pi(x).\end{aligned}$$

The acceptance procedure of the given algorithm can now be seen as the implementation of a simple Metropolis-Hastings acceptance probability using this density.

3.5.3 Removing slices but retaining sparsity in Ω

Applying the upper bound $\bar{h}_\omega(x)$ as described in Section 3.5.2 means that there will only a finite number of points in the Poisson process $\tilde{P}$. This is an important feature, as it implies that the process of computing the acceptance of a new state x^* only requires computing $h_\omega(x)$ and $h_\omega(x^*)$ for those values of ω which are present in $\tilde{P}$. We refer to this here as "sparsity."

This sparsity can be beneficial in a big data setting (where $\Omega = [n]$, so each atom corresponds to a datapoint). It is essential in a setting where e.g. $\Omega = \mathbb{R}$ and $h_\omega(x)$ is a function $h : \mathcal{X} \times \mathbb{R} \mapsto \mathbb{R}$ which can only be evaluated at points of x and ω.

While the Poisson process slice construction of (3.22) can yield sparsity, we note that the construction is suboptimal in the sense that the overall acceptance probability is decreased. To see this, we compare the acceptance rate when using the Poisson process auxiliarization versus the Metropolis-Hastings acceptance probability which could be implemented if we could evaluate $U(x) = \int h_\omega(x)\mu(\mathrm{d}\omega)$. We have for the acceptance probability of moving from x to a proposal x^*:

$$\begin{aligned}\alpha((x,P),(x^*,P)) &= \exp\left(-\int [h_\omega(x^*) - h_\omega(x)]_+ \mu(\mathrm{d}\omega)\right) \\ &\leq \exp\left(-\left[\int [h_\omega(x^*) - h_\omega(x)]\mu(\mathrm{d}\omega)\right]_+\right) = \alpha_{\mathrm{MH}}(x,x^*)\end{aligned}$$

where we have assumed for illustration that $\bar{h}_\omega(x) = \bar{h}_\omega(x^*)$.

By simply "integrating out" the y variables we arrive at a construction which remains sparse in Ω but does not incur the acceptance rate penalty as above. Define the measure $\mathbb{Z}$ the measure of a Poisson process defined on Ω and with rate $M_\omega\mu(\mathrm{d}\omega)$. Denote this Poisson process P.

Now a joint density over x and the Poisson process P is defined as

$$\frac{\mathrm{d}\pi}{\mathrm{d}\mathbb{X}\times\mathbb{Z}} = \exp\left(-\int \bar{h}_\omega(x)\mu(\mathrm{d}\omega)\right)\prod_{\omega\in P}\frac{\bar{h}_\omega(x) - h_\omega(x)}{M_\omega}.$$

and similarly to before, the marginal density is

$$\begin{aligned}\int_{\mathcal{P}} \frac{\mathrm{d}\pi}{\mathrm{d}\mathbb{X}\times\mathbb{Z}}(x,P)\mathbb{Z}(\mathrm{d}P) &= \exp\left(-\int \bar{h}_\omega(x)\mu(\mathrm{d}\omega)\right)\int_{\mathcal{P}}\prod_{\omega\in P}\frac{\bar{h}_\omega(x)-h_\omega(x)}{M_\omega}Z(\mathrm{d}P)\\ &= \exp\left(-\int \bar{h}_\omega(x)\mu(\mathrm{d}\omega)-\int M_\omega\mu(\mathrm{d}\omega)+\int[\bar{h}_\omega(x)-h_\omega(x)]\mu(\mathrm{d}\omega)\right)\\ &\propto \pi(x).\end{aligned}$$

Deriving the corresponding Metropolis-Hastings acceptance probability for this density is a straightforward matter of taking the ratio between densities. For a symmetric proposal, we have

$$\alpha((x,P),(x^*,P)) = \min\left\{1, \frac{\exp\left(-\int \bar{h}_\omega(x^*)\mu(\mathrm{d}\omega)\right)}{\exp\left(-\int \bar{h}_\omega(x)\mu(\mathrm{d}\omega)\right)}\prod_{w\in P}\frac{\bar{h}_\omega(x^*)-h_\omega(x^*)}{\bar{h}_\omega(x)-h_\omega(x)}\right\}; \tag{3.23}$$

which can be evaluated as the integrals over $\bar{h}_\omega$ are assumed to be computable.

The expectation of this acceptance probability over the distribution of $P\sim\pi(x,\cdot)$ is still less than the Metropolis-Hastings ratio; note simply that taking the expectation of the $\min\{1,\cdot\}$ argument yields the Metropolis-Hastings ratio, α is then inferior by Jensen's inequality.

On the other hand, this new ratio dominates the acceptance probability attained when including the y variables (3.21). To see this, take the expectation of that acceptance probability over the y variables which have distribution conditional on the corresponding ω of $y|\omega\sim\mathrm{Uniform}(0,\bar{h}_\omega(x)-h_\omega(x))$:

$$\mathbb{E}_y\alpha((x,P),(x^*,P)) = \min\left\{1, \frac{\exp\left(-\int \bar{h}_\omega(x^*)\mu(\mathrm{d}\omega)\right)}{\exp\left(-\int \bar{h}_\omega(x)\mu(\mathrm{d}\omega)\right)}\right\}\prod_{(\omega,y)\in P}\min\left\{1,\frac{\bar{h}_\omega(x^*)-h_\omega(x^*)}{\bar{h}_\omega(x)-h_\omega(x)}\right\} \tag{3.24}$$

which is inferior to (3.23) as $\min\{1,\prod_i r_i\}\leq\prod_i\min\{1,r_i\}$ for any r_i.

3.5.4 Slices for strictly atomic measures

Previously we had first developed the case where Ω contained atoms but was not strictly atomic; for simplicity we only develop here the case that μ is strictly atomic, the extension to the mixed case is straightforward.

We make the same observation as before, namely that for atoms of μ we have multiple redundant points of the Poisson process associated with each atom. Using the construction of a Poisson process over the space $\Omega \times \mathbb{R}$ and without the upper bound $\bar{h}_\omega$, we have the situation that for each atom there are an infinite number of points in $\tilde{P}$ associated with the atom. When determining whether to accept the swap we compare whether these points $\{y : (y, \omega) \in \tilde{P}\}$ are above the new function $h_\omega(x^*)$. To make this determination, we only actually need to check that the least of the points associated with ω is above the new function $h_\omega(x^*)$, i.e. if $\min\{y : (y, \omega) \in \tilde{P}\} > h_\omega(x)$ then $y > h_\omega(x)$ for all $\{y : (y, \omega) \in \tilde{P}\}$.

Therefore, we suggest that the process $\tilde{P}$ be replaced by an individual scalar for each value of ω which is equal in distribution to that least value of y associated with the atom. This least value has an exponential distribution truncated to $[h_\omega(x), \infty)$. Call these values $w = \{w_\omega : \omega \in \Omega_A\}$. These are independent conditional on x and have conditional density

$$\begin{aligned}\pi(w|x) &= \prod_{\omega \in \Omega_A} \text{Exp}(w_\omega - h_\omega(x); 1) \\ &= \prod_{\omega \in \Omega_A} \exp(-w_\omega + h_\omega(x)) \mathbb{I}(w_\omega > h_\omega(x))\end{aligned}$$

and so the joint density is

$$\begin{aligned}\pi(x, w) = \pi(w|x)\pi(x) &\propto \prod_{\omega \in \Omega_A} \exp(-w_\omega + h_\omega(x)) \mathbb{I}(w_\omega > h_\omega(x) \exp(-h_\omega(x)) \\ &= \prod_{\omega \in \Omega_A} \exp(-w_\omega) \mathbb{I}(w_\omega > h_\omega(x)).\end{aligned}$$

Reparametrizing w by taking $u = \{u_\omega = \exp(-w_\omega) : \omega \in \Omega_A\}$ gives a distribution

$$p(x, u) \propto \prod_{\omega \in \Omega_A} \mathbb{I}(u_\omega < \exp(-h_\omega(x))) \tag{3.25}$$

which we recognize as a unique slice variable for each factor of the distribution.

Introducing an upper bound yields the same sort of sparsity property found in the diffuse case. Here, denote $\tilde{F}$ a set of elements (but not a Poisson process) in the space $\Omega_A \times \mathbb{R}$ which defines the set of atoms which are present; we use the

uniform slice construction (3.25). The density of this set is

$$p(x, \tilde{F}) \propto \prod_{\omega \in \Omega_A} \exp(-\bar{h}_\omega(x)) \times \prod_{(\omega,u) \in \tilde{F}} \mathbb{I}(u < \exp(-h_\omega(x)) - \exp(-\bar{h}_\omega(x))).$$

Integrating out the uniform component u is straightforward: now we have a set $F \subseteq \Omega_A$, and the density is

$$p(x, F) \propto \prod_{\omega \in \Omega_A} \exp(-\bar{h}_\omega(x)) \prod_{\omega \in F} [\exp(-h_\omega(x)) - \exp(-\bar{h}_\omega(x))]. \tag{3.26}$$

For this final expression, the Metropolis-Hastings acceptance ratio for a proposal x^* (with a symmetric proposal) while keeping F fixed would be

$$\begin{aligned}\alpha((x, F), (x^*, F)) &= \min\left\{1, \frac{p(x^*, F)}{p(x, F)}\right\} \\ &= \min\left\{1, \prod_{\omega \in \Omega_A} \frac{\exp(-\bar{h}_\omega(x^*))}{\exp(-\bar{h}_\omega(x))} \prod_{\omega \in F} \frac{\exp(-h_\omega(x^*)) - \exp(-\bar{h}_\omega(x^*))}{\exp(-h_\omega(x) - \exp(-\bar{h}_\omega(x))}\right\}\end{aligned}$$

which is analogous to the acceptance probability for the diffuse case (3.24).

3.5.5 Previous work

The above framework encompasses a wide range of techniques seen in the literature. While the framework connects several disparate techniques — perhaps most uniquely in establishing the link between diffuse and atomic measures — the particular algorithms derived in this section have appeared previously in various forms.

The Poisson process constructions for the non-atomic case bear strong resemblances to the rejection schemes of Beskos et al. (2006) where a sample of a diffusion is accepted if all the points of a Poisson process lie above the diffusion; the diffusion is only realized at the points of the Poisson process.

An auxiliary process construction along the lines of that in Section 3.5.3 is first found in Adams et al. (2009), where it was motivated as the imputation of "rejected data." In that model, a Gaussian process is used to model the intensity of a Cox process; data may be generated by thinning a Poisson process. The realization that these auxiliary variables follow a Poisson process was realized by Rao (2012).

An extension of these models to the case where the number of data are fixed was formulated by Murray et al. (2009), and conditional simulation for the auxiliary process was developed by Rao et al. (2016); note however that this latter process is *not* subsumed by the framework developed here.

For the atomic case, the multiple slice construction (3.25) is common, used by e.g. Edwards and Sokal (1988); this construction will play an important role in the exposition of Chapter 5. The version incorporating the bound $\bar{h}_\omega$ yields a formulation identical the potential switching algorithm of Mak (2005) and the Firefly Monte Carlo algorithm of Maclaurin and Adams (2015). We will revisit the latter in Chapter 4.

3.6 Fast-forwarding

All implementations of discrete-time piecewise-deterministic Markov chain Monte Carlo described above consist of simulating the algorithm using the kernel (3.2), that is, at each time step we simulate event with probability $\alpha(z)$ when in state z. This may appear to be wasteful in comparison to the continuous-time algorithms where one can directly simulate the time travelled before a bounce occurs, skipping over any intermediate time steps.

Motivated by this, we describe "fast-forwarding" schemes for the discrete-time case, where we compute directly the number of steps taken before a rejection occurs. First, we show that by bounding the acceptance rejection probability we can derive schemes which efficiently compute the Metropolis-Hastings acceptance probabilities for multiple steps. Second, we show that, under some conditions, a continuous-time bounce simulation can be used directly in the discrete-time context.

For a discrete-time piecewise-deterministic Markov chain Monte Carlo algorithm, the rejection of the forward proposal $\Phi(z)$ is analogous to a bounce in the continuous-time piecewise-deterministic Markov chain Monte Carlo algorithm. We introduce the term *discrete-time rejection time* (or simply *rejection time*) to refer to the number of steps taken before the forward proposal is rejected; this time is distributed according to (3.1). So given a current state z_t, simulating a rejection time $\tau \geq 0$ implies that

the chain will visit states $z_{t+k} = \Phi_k(z)$ for $k \in \{1, \dots, \tau\}$ if $\tau \geq 1$, and in any case for time $t + \tau + 1$ the state is sampled from the event kernel $z_{t+\tau+1} \sim Q(z_{t+\tau}, \cdot)$.

3.6.1 Efficient thinning of rejections

Assume there exists $\bar{\alpha} : \mathcal{Z} \to [0, 1]$ such that for $k \in \mathbb{N}$ we have

$$\alpha\left(\Phi_k(z)\right) \geq \bar{\alpha}(z, k) > 0.$$

It is then possible to simulate a *candidate rejection time* by simulating a time from the instrumental distribution

$$\mathbb{P}[\tau = j] = \{1 - \bar{\alpha}(z, j)\} \prod_{i=0}^{j-1} \bar{\alpha}(z, i) \tag{3.27}$$

which is then accepted (i.e. selected as an "actual" rejection time) with probability

$$\frac{1 - \alpha(\Phi_{\tau^*}(z))}{1 - \bar{\alpha}(z, \tau)}.$$

For a linear flow $\Phi(z) = (x + v, v)$, we can obtain such bounds by upper bounding the derivative of $t \mapsto U(x + vt)$.

3.6.2 Rejections from thinning continuous-time events

Here, we show that many schemes used to simulate bounce time in the continuous-time setting can be used in the discrete-time case. We have previously used the symbol Φ to indicate a deterministic flow in both the continuous- and discrete-time settings; in this section we assume that a continuous-time flow is available so that we might evaluate Φ_t for $t \in \mathbb{R}^+$, even though the discrete-time algorithm only considers points yielded by Φ_k for $k \in \{0, 1, \dots\}$.

In these approaches we will first simulate a real-valued bounce time t then take as our discrete-time rejection time

$$\tau = \lfloor t \rfloor,$$

i.e. the last discretized time point reached before the bounce time. This simple "rounding down" of the time will allow most of the apparatus of continuous-time event simulation to be applied in a discrete-time setting.

In the continuous-time setting, we used bounce rates $\bar{\lambda}(z)$ which were required to satisfy

$$\bar{\lambda}(z) \geq \lambda(z) = \left[\frac{\mathrm{d}}{\mathrm{d}t} H(\Phi_t(z))|_{t=0}\right]_+ .$$

Simulating events arriving at rate $\bar{\lambda}$ yields *candidate* event times which are then thinned to yield an actual event time, as described in Section 3.6.1. The probability of the first candidate bounce time t^* (i.e. the first realisation of an event with rate $\bar{\lambda}$) occurring during some window $[j, j+1)$ for any $j \in \{0, 1, 2, \dots\}$ is

$$\mathbb{P}\left[j \leq t^* < j+1\right] = \left\{1 - \exp\left(-\int_j^{j+1} \bar{\lambda}(\Phi_t(z))\mathrm{d}t\right)\right\} \prod_{i=0}^{j-1} \exp\left(-\int_i^{i+1} \bar{\lambda}(\Phi_t(z))\mathrm{d}t\right), \tag{3.28}$$

which is an instance of (3.27) where $\bar{\alpha}(z,k) = \exp\left(-\int_k^{k+1} \bar{\lambda}(\Phi_t(z))\mathrm{d}t\right)$. Thus we can simulate a candidate bounce time t^* and then take $\tau^* = \lfloor t^* \rfloor$ as our candidate discrete-time rejection time. This is always a valid candidate rejection time when using the Metropolis-Hastings acceptance probability $\alpha(z) = \min\{1, \rho(\Phi(z))/\rho(z)\}$, as the probability of Metropolis-Hastings rejection is bounded by $\bar{\alpha}$:

$$\begin{aligned}
1 - \alpha(\Phi_k(z)) &= 1 - \exp\left(-\left[H(\Phi_{k+1}(z)) - H(\Phi_k(z))\right]_+\right) \\
&= 1 - \exp\left(-\left[\int_k^{k+1} \left[\frac{\mathrm{d}}{\mathrm{d}t} H(\Phi_t(z))\right] \mathrm{d}t\right]_+\right) \\
&\leq 1 - \exp\left(-\int_k^{k+1} \lambda(\Phi_t(z))\mathrm{d}t\right) \\
&\leq 1 - \exp\left(-\int_k^{k+1} \bar{\lambda}(\Phi_t(z))\mathrm{d}t\right) = 1 - \bar{\alpha}(z,k).
\end{aligned}$$

Therefore, the continuous-time event-time simulation can be used to first simulate a candidate discrete-time rejection time which would then be thinned with probability

$$\frac{1 - \alpha(\Phi_\tau(z))}{1 - \bar{\alpha}(z,\tau)} = \frac{1 - \min\left\{1, \frac{\rho(\Phi_{\tau+1}(z))}{\rho(\Phi_\tau(z))}\right\}}{1 - \exp\left(-\int_\tau^{\tau+1} \bar{\lambda}(\Phi_t(z))\mathrm{d}t\right)},$$

which requires computing the integral in the denominator. The above procedure is made explicit in Algorithm 18.

Algorithm 18 Fast-forwarding with continuous-time bounce rates

function FastForward(z)
  Set $\tau_0 \leftarrow 0$.
  repeat
    Simulate $t^* \geq \tau_0$ with survival function

$$\mathbb{P}[t^* \geq t] = \int_{\tau_0}^{t} \bar{\lambda}(\Phi_s(z))\mathrm{d}s.$$

    $\tau^* \leftarrow \lfloor t^* \rfloor$.
    $\tau_0 \leftarrow \lceil t^* \rceil$.
    Sample $u \sim \text{Uniform}(0, 1)$.
    $\alpha \leftarrow \left\{1 - \min\left\{1, \frac{\rho(\Phi_{\tau^*+1}(z))}{\rho(\Phi_{\tau^*}(z))}\right\}\right\} / \left\{1 - \exp\left(-\int_{\tau^*}^{\tau^*+1} \bar{\lambda}(\Phi_s(z))\mathrm{d}s\right)\right\}$.
  until $u < \alpha$.
  return τ^*.
end function

We highlight an algorithm for convex targets as a special case of these continuous-time bounce schemes: assume that $\pi(x) = \exp(-U(x))$ where U is strictly convex, $\Phi(z) = (x + v, v)$ and $\alpha(z) = \min\{1, \rho(\Phi(z))/\rho(z)\}$. In continuous time, an event time may be found by first finding the time at which the trajectory reaches the minimum potential $t_{\min} = \text{argmin}_{t \in (0,\infty)} U(x + tv)$, taking $E \sim \text{Exp}(1)$, then solving for $t > 0$ in $U(x+tv) - U(x+t_{\min}v) = E$ (Bouchard-Côté et al. 2018; Section 2.3.1).

A naïve approach directly using the above continuous-time mechanism would, for a candidate $t^* < \lceil t_{\min} \rceil$, require thinning of this time as $\alpha((x + \lfloor t_{\min} \rfloor v, v)) \geq \bar{\alpha}((x, v), \lfloor t_{\min} \rfloor)$. Algorithm 19 implements this scheme where by considering the case of $U(x + \lfloor t_{\min} \rfloor v) > U(x + \lceil t_{\min} \rceil v)$ and the opposite, allows for an algorithm that does not require thinning. This adapts the approaches developed in the continuous-time bouncy particle sampler algorithm to the discrete-time case.

Algorithm 19 Simulating discrete-time rejection times from continuous-time event times for convex potentials

function RejectionTimeConvex(x, v)
$t_{\min} \leftarrow \operatorname{argmin}_{t\in(0,\infty)} U(x+tv)$. ▷ Minimize potential along continuous trajectory.
$k_{\min} \leftarrow \operatorname{argmin}_{k\in\{\lfloor t_{\min}\rfloor, \lceil t_{\min}\rceil\}} U(x+kv)$.
Sample $E \sim \text{Exp}(1)$.
Solve

$$U(x+tv) - U(x+k_{\min}v) = E$$

for $t \geq t_{\min}$.
return $\tau = \lfloor t \rfloor$.
end function

3.6.2.1 Using continuous-time bounce times without thinning

The above procedure of thinning continuous-time bounce times to recover the Metropolis-Hastings acceptance probability is limited by the need to compute the probability of a bounce occurring within a time interval. We note here that trajectories simulated from a continuous-time algorithm may be used directly as discrete samples.

Specifically, if we simulate z_t as a continuous-time piecewise-deterministic Markov chain Monte Carlo process targeting distribution ρ, and take samples $\{z_{t_1}, z_{t_2}, z_{t_3}, \dots\}$ which are evenly spaced (i.e. $t_{i+1} - t_i = \epsilon$ for all i), this forms a valid discrete-time Markov chain targeting ρ. That this chain is valid is a consequence of the fact that the continuous-time chain is skew-reversible for any fixed time step Andrieu and Livingstone (2019).

3.7 Bounces in discrete time

In Section 3.3 we described a bounce in the context of the discrete-time bouncy particle sampler. That procedure involved taking the gradient at the current point x: with v^* as the proposed reflected velocity which for the inside reflection variant we take $v^* = R(v; \nabla U(x))$. Recall that the acceptance probability for inside reflection:

$$\beta_x(v, v^*) = \min\left\{1, \frac{[\pi(x) - \pi(x - R(v; \nabla U(x)))]_+}{[\pi(x) - \pi(x+v)]_+}\right\};$$

x is unchanged in this proposal.

This procedure is a discretization of the continuous-time algorithm in the sense that they are equivalent when the step-size of the discrete-time algorithm is very small. In such a setting, we may consider a linearization of the potential such that $U(x+v) \approx U(x) + \langle \nabla U(x), v \rangle$ for small v. This gives $U(x+v) \approx U(x-v^*)$, since $\langle \nabla U(x), v \rangle = \langle \nabla U(x), v^* \rangle$ due to the construction of the bounce, suggesting that the bounce proposal would always be accepted. Nonetheless, the bounce operation in discrete-time is not as efficient as in continuous-time: not only does the acceptance procedure require additional evaluation of the potential at the point $x - v^*$, but the possibility of rejecting a bounce means that some effort is wasted. We might even expect that for larger step-sizes the algorithm as a whole would perform more efficiently as it would explore the state-space more quickly (i.e. with fewer target density evaluations), yet larger step-sizes could yield bounces which are accepted with diminished frequency.

3.7.1 For decomposable rates

The interpretation that a discrete-time move can involve multiple rejections — in other words, that a rejection is associated with multiple points in Ω — motivates the choice to use a bounce kernel which depends on those rejected points. For this, we suggest that for the rejected set $P \subseteq \Omega$, that the gradient be chosen as a sum over these points:

$$\sum_{\omega \in P} \nabla_x h_\omega(x)$$

in the case that $U(x) = \int h_\omega(x) \mu(\mathrm{d}\omega)$. The proposed velocity is then taken as $v^* = R(v; \sum_{\omega \in P} \nabla_x h_\omega(x))$. We argue that this choice appropriately accounts for those values of ω which cause the rejection — in other words, are moving towards regions of higher energy — and directs the subsequent flow towards regions of higher probability for those values of ω.

It is not quite clear that this extension is the most natural extension of the continuous-time procedure to discrete-time. We can say, at least, that with

decreasing step-size we expect this to become similar to the continuous-time algorithm as, when using a small step-size, a rejection will be associated to a single ω as in continuous-time. Additionally, for independent factors we see that that the effect of applying the multiple bounces would be equivalent to using a sum of gradients.

3.8 Tunnelling in discrete time

The algorithms above have focused on bounce events where the trajectory is reflected off of the contours of the target potential at rejection times. In Chapter 2 we introduced another important class of event kernels to which we gave the name *tunnelling*. Upon encountering a rejection, these event kernels arrive at a new state by continuing to take deterministic steps until a point of high density is again reached. In some cases these deterministic steps even follow the same flow Φ used to propose the piecewise-deterministic proposals.

To construct these algorithms in discrete-time, a slice construction will prove useful. As usual, we assume a slice variable $u \in \mathbb{R}_+$ and the slice-augmented density is proportional to $\mathbb{I}(u < \rho(z))$. The algorithm differs from the bounce-type algorithms in that the event kernel Q is implemented as a piecewise-deterministic flow which propagates z until the condition $u < \rho(z)$ is again satisfied. Algorithm 20 gives a general algorithm using a flow Φ and a distinct tunnelling flow Φ'.

Many variants on this algorithm are possible; we suggest that the most important aspect here is that by using slice variables we can easily establish skew-detailed balance for fairly complicated proposals. Algorithm 20 is general enough to describe those algorithms we have identified in the literature.

Algorithm 20 describes a skew-reversible move if $|\mathbf{J}_\Phi| = 1$, $|\mathbf{J}_{\Phi'}| = 1$, and $z^* = \Phi' \circ \Phi(z) \iff z = \Phi' \circ \Phi \circ \mathcal{S}(z^*)$. For this it is sufficient that $\Phi'(z)$ be a composition of the typical transform Φ and some other volume-preserving transform Φ'' (so so $\Phi' = \Phi \circ \Phi''(z)$) and for which $\mathcal{S} \circ \Phi'' \circ \mathcal{S} \circ \Phi'' = \mathrm{Id}$. The move is skew-reversible as if in state z we accept state z', then starting the dynamics beginning in $\mathcal{S}(z')$ will trace the same path backwards, accepting $\mathcal{S}(z)$ as the first on the slice.

Algorithm 20 Tunnelling piecewise-deterministic Markov chain Monte Carlo.

function TUNNELLINGPDMCMC(z)
  Simulate $u \sim \text{Uniform}[0, \rho(z)]$.
  $z^* \leftarrow \Phi(z)$.
  if $u < \rho(z^*)$ **then**
    return z^*.
  else
    while stopping condition has not been reached **do**
      $z^* \leftarrow \Phi'(z^*)$.
      if $u < \rho(z^*)$ **then**
        return z^*.
      end if
    end while
    return $\mathcal{S}(z)$.
  end if
end function

The stopping condition is quite flexible, the only condition being that if it allows a trajectory to continue then it must also allow the reverse trajectory to continue.

Possibly the first description of an algorithm implementing tunnelling ideas is the "outside reflection" variant of the reflective slice sampler (Neal 2003). There $\Phi(x, v) = (x + v, v)$, and the off-slice dynamics implement a combination of a reflection then a step forward, i.e. $\Phi'(x, v) = \Phi(x, R(v; \nabla U(x)))$.

Tunnelling with Hamiltonian dynamics was introduced in Sohl-Dickstein et al. (2014) and Campos and Sanz-Serna (2015). In Sohl-Dickstein et al. (2014) delayed-rejection acceptance probabilities are employed, while in Campos and Sanz-Serna (2015) a slice construction is used. Unlike the linear dynamics of the reflective slice sampler, the Hamiltonian leapfrog dynamics will typically return to the slice, so the same flow Φ is also used for Φ' (alternatively, $\Phi'' = \text{Id}$). It is shown by Campos and Sanz-Serna (2015) that both the slice and the delayed-rejection constructions yield the same acceptance probabilities in this context.

In Park and Atchadé (2019) the authors consider a variant where the tunnelling dynamics only terminate once the L-th point on the slice is reached. They demonstrate that the slice and delayed-rejection implementations differ in the case that $L > 1$, but it is unclear why choosing $L > 1$ would yield improvements in

sample quality and the authors do not explore this experimentally.

Typical stopping conditions are to apply Φ' a maximum number of times (Neal 2003, Sohl-Dickstein et al. 2014, Campos and Sanz-Serna 2015) and if no point satisfying $u < \rho(z^*)$ is found the algorithm returns $\mathcal{S}(z)$. Other ideas include stopping the sequence if $\rho(z^*)$ is a very small value; in the case of Hamiltonian dynamics this may indicate that the integration is not stable (Hoffman and Gelman 2014).

Some other less general but similar constructions exist (Moriarty et al. 2019). We also note that some algorithms such as the No-U-Turn sampler (Hoffman and Gelman 2014) use very similar tunnelling-like constructions but not in a piecewise-deterministic setting.

Finally, we point out that the continuous-time version (Algorithm 10) can be seen as a version of this algorithm obtained in the limit of decreasing step size. For a small step size, the difference between $\rho(\Phi(z))$ and $\rho(z)$ is small, implying that the slice variable u is close to $\rho(z)$. Thus the procedure in continuous-time of running the flow until the density again reaches the value (and we have $\rho(z) = \rho(\Phi_t(z))$) is the same as waiting until the density returns to the slice (when $u < \rho(\Phi_t(z))$).

4

Piecewise-deterministic Markov chain Monte Carlo for large datasets

In this chapter we consider the setting of i.i.d. data where we have a posterior distribution of the form

$$\pi(x) \propto \pi_0(x)\pi(y|x) = \pi_0(x) \prod_{n=1}^{N} \pi(y_n|x),$$

where $x \in \mathbb{R}^d$ is a parameter vector, $\pi_0(x)$ is the prior over these parameters, and $\pi(y_n|x)$ is the likelihood of observation y_n. Write the corresponding potential

$$U(x) = f_0(x) + \sum_{n=1}^{N} f_n(x) \tag{4.1}$$

where $f_0(x) = -\log \pi_0(x)$ and $f_n(x) = -\log \pi(y_n|x)$.

We will concern ourselves with the case of large N which we call the *large dataset* setting. We will be interested in finding methods which scale well in N; to compare these algorithms we will determine the "big-O" time cost of computing a single iteration of an Markov chain Monte Carlo algorithm. A naïve Metropolis-Hastings algorithm applied to the above requires accessing N observations (that is, evaluating N observation-specific terms) per iteration. We will discuss here algorithms which scale with the dataset size so as to require accessing $O(1)$ and even $O(N^{-1/2})$ observations per iteration.

Note that the potential (4.1) is a candidate for our framework of decomposable rates, as described in both Chapters 2 and 3. Here the set of factors is $\Omega = \{0, 1, 2, \ldots, N\}$, μ is the counting measure i.e. $\mu(A) = |A|$ for $A \subseteq \Omega$, and $U_n(x) = f_n(x)$ for all $n \in \{0, 1, \ldots, N\}$. We will review some continuous-time algorithms leveraging this factorization, develop discrete-time versions, and demonstrate some ways in which these discrete-time versions can be improved. Finally, we connect these discrete-time algorithms to other algorithms in the literature.

4.1 Subsampling of rates for the per-observation factorization

Here we present the continuous-time algorithm as applied to the large dataset setting. As discussed, the large dataset posterior can be factorized in the following way:

$$U(x) = \underbrace{f_0(x)}_{U_0(x)} + \sum_{n=1}^{N} \underbrace{f_n(x)}_{U_n(x)} \tag{4.2}$$

which places each observation in its own factor and the prior in another factor. A continuous-time algorithm using such a factorization will simulate events where each event rate depends only on a single observation. We will see that these rates can indeed be simulated in an efficient way, yielding algorithms which scale sublinearly in N.

Placing the above in a framework allowing for arbitrary flow is straightforward. We can, for example, use an energy $H(x, v) = \sum_n U_n(x) + K(v)$ and place the kinetic energy in its own factor, or we can distribute K amongst the N data factors, etc. Henceforth we assume $N + 1$ factors indexed $\{0, \ldots, N\}$ where each factor is associated with at most one observation, and use a general expression for the rate as a function over time $\lambda_n(\Phi_t(z)) : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ in terms of an arbitrary flow Φ.

As discussed in Section 2.2.1, we can simulate from the appropriate rate

$$\lambda(\Phi_t(z)) = \sum_{n \in \{0,\ldots,N\}} \lambda_n(\Phi_t(z)) = \sum_{n \in \{0,\ldots,N\}} \langle \nabla H_n(\Phi_t(z)), \phi(\Phi_t(z)) \rangle_+$$

by either

1. simulating $N+1$ event times, each of rate $\lambda_n(\Phi_t(z))$, and taking the least of these, or by

2. simulating an event of rate $\lambda(\Phi_t(z)) = \sum_n \lambda_n(\Phi_t(z))$ and then selecting an index k with probability $\lambda_k(\Phi_t(z))/\sum_n \lambda_n(\Phi_t(z))$. We can interpret this as a *marked* Poisson process (Kingman 1993; Section 5.2), which can be interpreted as a Poisson process on the space $\Omega \times \mathbb{R}$ with $\lambda_k(\Phi_t(z))$ an intensity function over both index and position.

While there may be some way to build off of the first approach and attain an algorithm sublinear in N (perhaps by taking e.g. a branch-and-bound approach) we focus on the second approach which is apparently simpler.

Simulating from the overall rate $\lambda(\Phi_t(z))$ directly will not be possible in most scenarios and we resort to *thinning* approaches as discussed in Section 2.3, which allow us to instead simulate events with a bounding rate $\bar{\lambda}(\Phi_t(z)) \geq \lambda(\Phi_t(z))$ and "thin" these events, accepting them with probability $\lambda(\Phi_t(z))/\bar{\lambda}(\Phi_t(z))$.

Extending this idea to the large data setting, we will employ the algorithm given in Algorithm 21. This proceeds by first simulating a pair (t, k) from a Poisson process with a bounding rate $\bar{\lambda}_k(\Phi_t(z))$, which is then thinned to the correct rate.

To efficiently implement Algorithm 21, we will consider two main aspects:

1. It must be possible to efficiently simulate from the rate $\sum_n \bar{\lambda}_n(\Phi_\tau(z))$. This implies at least that the aggregate bound be easily computable and of a form for which we can simulate candidate times.

   The bound here is directly related to the scaling of the algorithm, as it determines how often we visit observations. There are two aspects to this: as N increases the posterior distribution from which we are sampling will tend to concentrate around a point, implying that these bounds only need to be effective over a smaller region. At the same time, we need to ensure that with increasing data these bounds do not increase too much.

Algorithm 21 Efficient subsampling of multiple bounded rate functions.

Given starting state z, rates $\{\lambda_n(\Phi_t(z))\}_{n\in\{0,\dots,N\}}$, and bounding rates $\{\bar{\lambda}_n(\Phi_t(z))\}_{n\in\{0,\dots,N\}}$.
$t_0 \leftarrow 0$.
repeat
Simulate the first arrival $\tau > t_0$ of an inhomogenous Poisson process with rate

$$\sum_n \bar{\lambda}_n(\Phi_\tau(z)).$$

Set $t_0 \leftarrow \tau$.
Select an index $k \in \{0, \dots, N\}$ with probability

$$\mathbb{P}[k = \ell] = \frac{\bar{\lambda}_\ell(\Phi_\tau(z))}{\sum_n \bar{\lambda}_i(\Phi_\tau(z))}. \tag{4.3}$$

Simulate $u \sim \text{Uniform}(0, 1)$.
until $u < \lambda_k(\Phi_\tau(z))/\bar{\lambda}_k(\Phi_\tau(z))$
return τ.

2. The selection of the index k would generally have a time cost of $O(N)$ as it samples from the categorical distribution over the $N + 1$ elements as expressed by (4.3). However we note that in some cases this can be reduced to $O(1)$, for example when the rates are bounded uniformly, i.e. if we have a bound $\bar{\lambda}_*(\Phi_t(z))$ such that $\lambda_n(\Phi_t(z)) \leq \bar{\lambda}_*(\Phi_t(z))$ for all n. In this case, the distribution over k is uniform and can be easily sampled in $O(1)$ time. This latter strategy is employed by e.g. the zig-zag algorithm as applied to logistic regression (Bierkens et al. 2019a).

These concerns are interrelated and may not be particularly enlightening as to what kinds of bounds are appropriate. Nonetheless they at least illustrate that a naïve implementation will scale linearly with N.

Despite several attempts (Banterle et al. 2015, Bouchard-Côté et al. 2018), the simple factorization of (4.2) has not led to scalable large dataset algorithms, though it seems that some efficiency gains are possible. In the next section we discuss an improved approach which leverages approximations to the potential centered about the mode.

4.2 An improved factorization using Taylor-series approximations

Here we introduce a particular scheme for leveraging approximations to the large dataset posterior. This approximation has several antecedents in the literature, see the ensuing Section 4.2.1 for a detailed discussion of these.

The primary idea is to introduce a per-observation approximation

$$\widehat{f}_n(x) \approx f_n(x)$$

for $n \in \{1, \ldots, N\}$ which can be aggregated to form a *global* approximation

$$\widehat{f}(x) = \sum_n \widehat{f}_n(x) \approx f(x).$$

We will consider the factorization

$$U(x) = f_0(x) + \sum_{n=1}^{N} f_i(x) \tag{4.4}$$

$$= f_0(x) + \sum_{n=1}^{N} f_n(x) + \widehat{f}(x) - \sum_{n=1}^{N} \widehat{f}_n(x) \tag{4.5}$$

$$= \underbrace{f_0(x) + \widehat{f}(x)}_{U_0(x)} + \sum_{n=1}^{N} \underbrace{\left[f_n(x) - \widehat{f}_n(x)\right]}_{U_n(x)} \tag{4.6}$$

which has an intuitive interpretation: the factor U_0 represents an approximation to the overall U, and each U_i represents the discrepancy between the likelihood of each observation and its approximation. Later we will bound the difference between $\hat{f}_n$ and f_n, though this will be both algorithm- and problem-dependent.

We note an alternative factorization which distributes the global approximation $\hat{f}$ amongst the per-observation factors:

$$U(x) = \underbrace{f_0(x)}_{U_0(x)} + \sum_{n=1}^{N} \underbrace{[\frac{1}{N}\hat{f}(x) + f_n(x) - \widehat{f}_n(x)]}_{U_n(x)}. \tag{4.7}$$

In our experiments we find little practical difference between (4.6) and (4.7).

While there may be many possible approaches, we find a Taylor-series expansion of f_n to be both sufficient and straightforward. We write the Kth order Taylor

series expansion of f_n about the point $\hat{x}$ as

$$\widehat{f}_n(x) = \sum_{k=0}^{K} \sum_{\alpha:|\alpha|=k} \frac{1}{\alpha!} \partial^\alpha f_n(\hat{x})(x - \hat{x})^\alpha \tag{4.8}$$

where α represents a multi-index, i.e. $\alpha \in \mathbb{N}^d_{\geq 0}$, $|\alpha| := \sum_{i=1}^d \alpha_i$, $\alpha! := \alpha_1!\alpha_2! \ldots \alpha_d!$, $\partial^\alpha := \frac{\partial^{|\alpha|}}{\partial x_1^{\alpha_1} \partial x_2^{\alpha_2} \ldots \partial x_d^{\alpha_d}}$, and $x^\alpha := \prod_{i=1}^d x_i^{\alpha_i}$.

The Taylor series affords a simple computation of the overall approximation $\widehat{f}$. Taking a sum of the per-observation approximations

$$\begin{aligned}\hat{f}(x) = \sum_n \hat{f}_n(x) &= \sum_n \sum_{k=0}^{K} \sum_{\alpha:|\alpha|=k} \frac{1}{\alpha!} [\partial^\alpha f_n(\hat{x})](x - \hat{x})^\alpha \\ &= \sum_{k=0}^{K} \sum_{\alpha:|\alpha|=k} \frac{1}{\alpha!} (x - \hat{x})^\alpha \sum_n \partial^\alpha f_n(\hat{x}).\end{aligned}$$

The sums $\sum_i \partial^\alpha f_i(\hat{x})$ can be precomputed so $\widehat{f}$ can be evaluated with $O(1)$ cost.

We choose $\hat{x}$ to be the mode of the distribution; this requires a precomputation step in which we optimize the full likelihood to find this point. This choice is critical as in a large dataset setting the posterior will concentrate around the mode; that the approximation errors are small around this point will allow us to control the number of observations visited in each iteration of the Markov chain Monte Carlo algorithm.

Based on this approximation, we can derive a rate for a factor U_n as in (4.6) (the derivation for (4.7) is similar). We derive this rate for a linear flow in terms of time t and some initial point (x_0, v_0):

$$\lambda_n(x_0 + v_0 t) = \left\{ \frac{\mathrm{d}}{\mathrm{d}t} f_n(x_0 + v_0 t) - \frac{\mathrm{d}}{\mathrm{d}t} \widehat{f}_n(x_0 + v_0 t) \right\}_+ .$$

Substituting for $\widehat{f}_n$ the approximation (4.8) with $K = 1$, We have

$$\begin{aligned}\left\{ \frac{\mathrm{d}}{\mathrm{d}t} f_n(x_0 + v_0 t) - \frac{\mathrm{d}}{\mathrm{d}t} \hat{f}_n(x_0 + v_0 t) \right\}_+ &= \left\{ \langle \nabla f_n(x_0 + v_0 t), v_0 \rangle - \frac{\mathrm{d}}{\mathrm{d}t} \langle \nabla f_n(\hat{x}), x_0 + v_0 t - \hat{x} \rangle \right\}_+ \\ &= \left\{ \langle \nabla f_n(x_0 + v_0 t) - \nabla f_n(\hat{x}), v_0 \rangle \right\}_+ .\end{aligned}$$

Following Bierkens et al. (2019a), we can bound this when the partial derivatives are Lipschitz with coordinate-dependent coefficients L_j, i.e.

$$\left| \frac{\partial}{\partial x_j} f_n(x) - \frac{\partial}{\partial x_j} f_n(y) \right| \leq L_j \|x - y\|_p. \tag{4.9}$$

Applying this assumption, we have the following. Let x_{0j} be the jth coordinate of x_0, then

$$\begin{aligned}
&\{\langle \nabla f_n(x_0 + v_0 t) - \nabla f_n(\hat{x}), v_0 \rangle\}_+ \qquad (4.10)\\
&\leq \sum_{j=1}^{d} \left| \frac{\partial}{\partial x_j} f_n(x_0 + v_0 t) - \frac{\partial}{\partial x_j} f_n(\hat{x}) \right| |v_{0j}| \\
&\leq \sum_{j=1}^{d} \left\{ \left| \frac{\partial}{\partial x_j} f_i(x_0 + v_0 t) - \frac{\partial}{\partial x_j} f_i(x_0) \right| + \left| \frac{\partial}{\partial x_j} f_i(x_0) - \frac{\partial}{\partial x_j} f_i(\hat{x}) \right| \right\} |v_{0j}| \\
&\leq \{\|v_0\|_p t + \|x_0 - \hat{x}\|_p\} \sum_{j=1}^{d} L_j |v_{0j}|. \qquad (4.11)
\end{aligned}$$

Thus we have a bounding rate of the form $\bar{\lambda}_n(x_0 + v_0 t) = a_n t + b_n$ where $a_n, b_n \geq 0$ for which event times can be easily simulated.

While by this interpretation we can claim that the continuous-time algorithm relies on the first-order Taylor-series approximation, it is unclear how we might extend these bounds to higher-order approximations, suggesting that this interpretation may not be the most appropriate. We leave this extension for future work.

4.2.1 Connections to previous work

Potentials factorized in the form (4.1) have been considered by Banterle et al. (2015). They propose a scheme whereby the observations are grouped and then agglomerated into blocks in a manner which achieves a reasonable acceptance rate.

Maclaurin and Adams (2015) propose an auxiliary variable method allowing for an exact Markov chain Monte Carlo scheme with subsampling. There an approximation is also used, but this must be a lower bound of the likelihood. For the logistic regression model they used the lower bound of Jaakkola and Jordan (1997). See Section 4.8 for more details, where we discuss some connections between these algorithm and propose an extension.

Bardenet et al. (2017) used similar likelihood approximations, but based on Taylor-series expansions as we have done here, to construct their "improved confidence sampler." Their algorithm does not exactly target the posterior, but requires many similar aspects of the approximation including the ability to bound the error of the approximation.

Bierkens et al. (2019a) apply their zig-zag algorithm (see Section 2.4.2) to the big data setting. The natural application of zig-zag to the naïve factorization (4.1) would yield a rate for each dimension and for each index $\{\lambda_{n,j} : n \in \{1, \ldots, N\}, j \in [d]\}$ where

$$\lambda_{n,j}(x, v) = v_j \frac{\partial}{\partial x_j} f_n(x).$$

To improve upon this, they introduce a control-variate scheme which gives rates of the form

$$\lambda_{n,j}(x, v) = v_j \left[\frac{\partial}{\partial x_j} f(\hat{x}) + \frac{\partial}{\partial x_j} f_n(x) - \frac{\partial}{\partial x_j} f_n(\hat{x}) \right]. \tag{4.12}$$

Here, $\hat{x}$ is some high-density point; we can effectively consider this to be the mode as in our exposition. This resembles control variates commonly used in stochastic-gradient algorithms, where a variance-reducing approximation $\frac{\partial}{\partial x_j} f_n(x) \approx \frac{\partial}{\partial x_j} f(\hat{x}) + \frac{\partial}{\partial x_j} f_n(x) - \frac{\partial}{\partial x_j} f_n(\hat{x})$ is made. Note however that the construction of Bierkens et al. (2019a) is exact: we have established in the previous section that this control variate is the application of the zig-zag algorithm to the factorization employing Taylor-series approximations given by (4.7).

An analogous application of the bouncy particle sampler to the same factorization was used by Galbraith (2016), however they do not correctly identify a corresponding factorization of the potential. Nonetheless the correct rates are given which are given, and are equal to those in (4.11). They find empirically that the bouncy particle sampler is more efficient than zig-zag.

The identification of the factorization (4.6) and the approximation (4.8) affords several avenues for generalization:

1. we may use the factorized potential in other algorithms, including discrete-time and more standard algorithms such as random-walk Metropolis-Hastings; and

2. we may consider other approximations, such as higher-order Taylor series approximations.

These generalizations were exploited by Cornish, Vanetti, Bouchard-Côté, Deligiannidis, and Doucet (2019), and are explored further in Section 4.4.

4.3 Alias tables

Besides bounding the rates uniformly as discussed in Section 4.1, there is another method which can be used to reduce the complexity of sampling the distribution (4.3) which relies on a data structure known as an *alias table*. First introduced by Walker (1977), an alias table is a simple data structure which enables sampling from a categorical distribution with constant cost. The original algorithm required $O(K \log K)$ time to construct a table of size K; this was improved by Kronmal and Peterson Jr (1979) to linear time, $O(K)$.

Since constructing an alias table to sample from a distribution over observations requires linear set-up time, it is only helpful for us in situations where it can be precomputed, so the set-up time is thus amortized over the full set of samples. This includes situations where the rates $\{\bar{\lambda}_i(\Phi_t(z))\}_{i=\{0,\dots,N\}}$ are fixed in proportion to each other regardless of the state z, as in this case the categorical distribution (4.3) does not change. Somewhat more generally, when we can express the rates in the form

$$\bar{\lambda}_n(\Phi_t(z)) = \sum_{j=1}^{J} \zeta_j(\Phi_t(z))\eta_{n,j}$$

for constants $\eta_{n,j} \in \mathbb{R}_{\geq 0}$, then we can sample from (4.3) using J alias tables. The $\zeta_j(z)$ are interpretable as the state-dependent scaling of the rates associated with each table, and the $\eta_{n,j}$ for a given j form the entries of the table. To sample from the required distribution, which is

$$\mathbb{P}[k] = \frac{\sum_{j=1}^{J} \zeta_j(\Phi_t(z))\eta_{k,j}}{\sum_{j=1}^{J} \zeta_j(\Phi_t(z)) \sum_n \eta_{n,j}}.$$

We present Algorithm 22 which first samples a table j and then samples an index

k from this table. This algorithm is correct as

$$\begin{aligned}\mathbb{P}[k] &= \sum_j \mathbb{P}[k|j]\mathbb{P}[j] \\ &= \sum_j \left[\frac{\eta_{k,j}}{\sum_n \eta_{n,j}} \times \frac{\zeta_j(\Phi_t(z)) \sum_n \eta_{n,j}}{\sum_{j'} \zeta_{j'}(\Phi_t(z)) \sum_n \eta_{n,j'}}\right]\end{aligned}$$

yields the desired distribution over k. If the J constants $\sum_n g_{n,j}$ are also computed ahead of time in addition to the tables themselves, then the first step requires $O(J)$ time and the second step requires $O(1)$ time, suggesting that this scheme will be useful when J is relatively small and N is large.

Algorithm 22 Sampling a categorical distribution using multiple alias tables.

Sample a table j according to

$$\mathbb{P}[j] = \frac{\zeta_j(\Phi_t(z)) \sum_n \eta_{n,j}}{\sum_{j'} \zeta_{j'}(\Phi_t(z)) \sum_n \eta_{n,j'}}.$$

Sample a factor i according to

$$\mathbb{P}[k|j] = \frac{\eta_{k,j}}{\sum_n \eta_{n,j}}.$$

return i.

This scheme was used by Bouchard-Côté et al. (2018; Appendix C) to implement alias tables over data points for logistic regression where they used $2d$ tables, one for positive velocity and one for negative velocity in each dimension.

4.4 Adapting discrete-time piecewise-deterministic Markov chain Monte Carlo for large datasets

The previous sections were focused on sampling in continuous-time and in choosing rates and subsampling schemes which are appropriate for a large N setting. We now turn to discrete-time and present the necessary adaptations to the schemes above.

We begin by describing the naïve application of the algorithms of Chapter 3 by selecting the same factorizations as used in continuous-time, either (4.2) or (4.6). In both cases we use a straightforward implementation of a discrete-time piecewise-deterministic Markov chain Monte Carlo algorithm where $\Omega = \{0, \ldots, N\}$

and the measure $\mu(A \subseteq \Omega) = |A|$. We describe this algorithm explicitly in Algorithm 23, where we use the Metropolis-Hastings acceptance probability for the proposal $\Phi(z)$, and where for the event kernel Q we choose a Metropolis-Hastings update with proposal q_B dependent on the set of Bernoulli "rejection" variables $B = \{B_n : n \in \{0, \ldots, N\}\}$. The factorization here is denoted $\rho(z) = \prod_n \rho_n(z)$; as before, we suggest that the specific factorization used is not important, so long as there is at most one observation per factor.

Algorithm 23 Discrete-time piecewise-deterministic Markov chain Monte Carlo for large data

for $n \in \{0, \ldots, N\}$ **do**
  Sample $B_n \sim \text{Bernoulli}\{[\rho_n(z) - \rho_n(\Phi(z))]_+ / \rho_n(z)\}$.
end for
if $B_n = 0$ for all $n \in \{0, \ldots, N\}$ **then**
  return $\Phi(z)$.
else
  Sample $z^* \sim q_B(z, \cdot)$.
  with probability

$$\min\left\{1, \frac{q_B(\mathcal{S}(z^*), \mathcal{S}(z))}{q_B(z, z^*)} \prod_{n:B_n=1} \frac{[\rho_n(z^*) - \rho_n(\Phi \circ \mathcal{S}(z^*))]_+}{[\rho_n(z) - \rho_n(\Phi(z))]_+} \prod_{n:B_n=0} \frac{\min\{\rho_n(z^*), \rho_n(\Phi \circ \mathcal{S}(z^*))\}}{\min\{\rho_n(z), \rho_n(\Phi(z))\}}\right\} \tag{4.13}$$

    return z^*.
  otherwise
    return $\mathcal{S}(z)$.
end if

This algorithm is poorly suited to the large data setting for two reasons:

1. computing the the Bernoulli variables B_n requires visiting each data point and thus requires $O(N)$ time; and

2. computing the event proposal acceptance probability (4.13) requires visiting each observation and so also requires $O(N)$ time.

To deal with the first issue, we present a scheme to efficiently sample Bernoulli variables, making use of the efficient categorical simulation schemes of Section 4.4.2. To deal with the second issue, we will replace the event acceptance probability with

a factorized version which will also be amenable to the same efficient Bernoulli simulation schemes used in the simulation of B_n.

We note that Algorithm 23 is easily extended to the many variants discussed previously, for example to a deterministic proposal for the event kernel. It can also be adapted to a Metropolis-Hastings version (analogous to Algorithm 17) which foregoes the need for an event kernel; in this case we do not need to access the set B and can reject a proposal as soon as we discover some $B_n = 1$, but nonetheless we still need a sublinear approach to sampling the B_n. We discuss this Metropolis-Hastings variant in Section 4.5.

4.4.1 The Taylor-series approximation in discrete time

Algorithm 23 describes the acceptance procedure for the proposal $\Phi(z)$ in terms of simulating independent Bernoulli variables. In this section we discuss the specific forms of these Bernoulli probabilities when using the factorization (4.6).

We have only given the factorization of $\pi(x)$ and not of the full density $\rho(z)$. Assuming $z = (x, v)$ and $\rho(x, v) = \pi(x)\psi(v) \propto \exp(-U(x) - K(v))$, we suggest a factorization for ρ which includes the kinetic energy as part of the 0th term:

$$\begin{aligned} H_0(x, v) &= f_0(x) + \hat{f}(x) + K(v) \\ H_n(x, v) &= f_n(x) - \hat{f}_n(x), \quad n \in \{1, \dots, N\}. \end{aligned}$$

With this, the rejection probabilities become, letting $z' = (x', v') = \Phi(x, v)$ for simplicity, for the 0th term

$$\mathbb{P}[B_0 = 1] = 1 - \min\left\{1, \frac{\rho_0(z')}{\rho_0(z)}\right\} = 1 - \min\left\{1, \frac{\hat{\pi}(y|x')\pi_0(x')\psi(v')}{\hat{\pi}(y|x)\pi_0(x)\psi(v)}\right\}$$

where we have denoted $\hat{\pi}(y|x) = \prod_n \hat{\pi}(y_n|x) = \prod_n \exp(-\hat{f}_n(x))$ to emphasize that this is an approximation to the likelihood over all data points.

For the observation-specific terms, we have for $n \in \{1, \dots, N\}$,

$$\mathbb{P}[B_n = 1] = 1 - \min\left\{1, \frac{\rho_n(z')}{\rho_n(z)}\right\} = 1 - \exp\left(-\left[[f_n(x') - \hat{f}_n(x')] - [f_n(x) - \hat{f}_n(x)]\right]_+\right).$$

Denoting the residual of the per-observation approximation as $R_n(x) = f_n(x) - \hat{f}_n(x)$, if we can bound these residuals then we can bound the rejection probability by bounding

$$[R_n(x') - R_n(x)]_+ \leq |R_n(x')| + |R_n(x)|. \tag{4.14}$$

When using a Taylor-series approximation this is available when the Taylor series remainder terms can be bounded; we do this for the logistic regression likelihood in Section 4.6.

4.4.2 Fast simulation of Bernoulli variables

Consider the simulation of $\{B_n\}_{n\in\{0,\ldots,N\}}$ of Algorithm 23. We note that the forward proposal is only accepted if *all* Bernoulli variables are zero, i.e. $B_n = 0$ for all $n \in \{0, \ldots, N\}$. This represents a bottleneck: any algorithm expected to scale sublinearly with dataset size will need to use a sublinear implementation of this step.

First, we suggest that the simulation of B_0 always be done; this term is a special case and cannot in general be bounded in the same way as the other factors. In the Metropolis-Hastings setting (where we do not require access to the set B) the rejection of this factor allows us to bypass the simulation of rejections for the remaining N factors.

We describe two methods for efficiently simulating the remaining N Bernoulli variables. These are analogous to our methods for efficiently simulating from an ensemble of rates in continuous time. Whereas in continuous time we required access to upper bounds on the rates, here we will need to develop upper bounds on the probability of rejecting a factor, $1 - \alpha_n(z)$. As in continuous time, the first method relies on a uniform upper bound on $1 - \alpha_n(z)$ over all factors, and one which allows us to use a factor-specific upper bound.

These algorithms can be understood as methods to sample a set of Bernoulli variables, and could be used in other contexts. An alternative to the methods described here is that of Shanthikumar (1985) (see also (Devroye 2006, Michel et al. 2019)). For the uniform bound the algorithms are similar; for per-coordinate

bounds they instead require bounds be sorted in decreasing order, as opposed to our method for which we must precompute an alias table.

4.4.2.1 With a uniform bound

First, consider the setting where we can uniformly bound the probability that $B_n = 1$, that is we have some bound $0 \leq \bar{p} \leq 1$ such that for all $n \in \{1, \ldots, N\}$,

$$\mathbb{P}[B_n = 1] := 1 - \alpha_n(z) \leq \bar{p}.$$

In this case, we can determine the set $\{n : B_n = 1\}$ using Algorithm 24. The computational cost of this algorithm is $O(1 + N\bar{p})$ compared to $O(N)$ for the direct implementation. We suggest that this algorithm can be considered the discrete-time analogue of the thinning procedures using a uniform bound $\bar{\lambda} \geq \lambda_n$ discussed in Section 4.1 and as used in the zig-zag algorithm (Bierkens et al. 2019a).

Algorithm 24 Efficient sampling of Bernoulli variables via Binomial sampling

Given Bernoulli probabilities $\{1-\alpha_n(z)\}_{n\in\{1,\ldots,N\}}$, and bound $\bar{p}$ such that $1-\alpha_n(z) \leq \bar{p}$ for all $n \in \{1, \ldots, N\}$.

function BERNOULLI-BINOMIAL($\{\alpha_n\}_{n\in\{1,\ldots,N\}}$, $\bar{p}$)
  Sample $S \sim \text{Binomial}(N, \bar{p})$.
  Sample S indices $i_1, \ldots, i_S$ in $\{1, \ldots, N\}$ uniformly at random without replacement and denote $\mathcal{S} = (i_1, \ldots, i_S)$.
  for $i \in \mathcal{S}$ **do**
    Sample $B_n \sim \text{Bernoulli}\left(\frac{1-\alpha_n(z)}{\bar{p}}\right)$.
  end for
  for $n \in \{1, \ldots, N\} \setminus \mathcal{S}$ **do**
    $B_n \leftarrow 0$.
  end for
  return $\{B_n\}_{n\in\{1,\ldots,N\}}$.
end function

The algorithm is correct as the marginal distribution of the Bernoulli variables is as desired. First, note that the presence of each index n is i.i.d.:

$$\begin{aligned}\mathbb{P}[\mathcal{S}] &= \binom{N}{S}^{-1} \text{Binomial}(S; N, \bar{p}) \\ &= \prod_{n=1}^{N} \text{Bernoulli}(\mathbb{I}(n \in \mathcal{S}); \bar{p}),\end{aligned}$$

and thus the marginal distribution over the variable B_n is

$$\begin{aligned}\mathbb{P}\left[B_n = 1\right] &= \mathbb{P}[B_n = 1 | n \in \mathcal{S}]\mathbb{P}[i \in \mathcal{S}] + \mathbb{P}[B_n = 1 | n \notin \mathcal{S}]\mathbb{P}[n \notin \mathcal{S}] \\ &= \frac{1 - \alpha_n(z)}{\bar{p}}\bar{p} + 0 = 1 - \alpha_n(z).\end{aligned}$$

4.4.2.2 With per-factor bounds

If we have access to index-specific bounds $0 \leq \bar{p}_n \leq 1$ such that

$$\mathbb{P}[B_n = 1] := 1 - \alpha_n(z) \leq \bar{p}_n,$$

we could use the previous strategy by setting $\bar{p} := \max_{n \in \{1,\ldots,N\}} \bar{p}_n$ but this strategy can be highly inefficient if, e.g., most bounds $\bar{p}_n$ are very close to zero and a few are larger. In this scenario, we can use instead Algorithm 25 which relies on the simulation of Poisson random variables. We suggest that this algorithm is the discrete-time analogue of the thinning procedures using per-factor continuous-time bounds $\bar{\lambda}_n \geq \lambda_n$.

Algorithm 25 Efficient sampling of Bernoulli variables via Poisson sampling

function BERNOULLI-POISSON($\{\alpha_n(z)\}_{n \in \{1,\ldots,N\}}$, $\{\bar{p}_n\}_{n \in \{1,\ldots,N\}}$)
  Sample Poisson counts $\{K_n\}_{n \in \{1,\ldots,n\}}$ with rates $\kappa_n = -\log(1 - \bar{p}_n)$ (using e.g. Algorithm 26).
  Denote $\mathcal{S} = \{n : K_n \geq 1\}$.
  for $n \in \mathcal{S}$ **do**
    Sample $B_n \sim \text{Bernoulli}\left(\frac{1-\alpha_n(z)}{\bar{p}_n}\right)$.
  end for
  for $n \in \{1, \ldots, N\} \setminus \mathcal{S}$ **do**
    $B_n \leftarrow 0$.
  end for
  return $\{B_n\}_{n \in \{1,\ldots,N\}}$.
end function

In order to sample the Poisson counts $\{K_n\}_{n \in \{1,\ldots,N\}}$ we once again make use of alias tables. As in to the continuous-time algorithms, sublinear rates are attained by employing alias tables as the simulation of the multinomial distribution would otherwise incur a cost of $O(N)$. In this case we take

$$\kappa_n = \sum_{j=1}^{J} \zeta_j(z)\eta_{n,j}$$

which is analogous to the formulation in continuous time (4.3). We present Algorithm 26 which describes a means to simulate $\{K_n\}_{n\in\{1,\dots,N\}}$ as required in Algorithm 25 but using precomputed alias tables.

Algorithm 26 Sampling from a Poisson process using multiple alias tables.

Given Poisson rates $\kappa_n = \sum_{j=1}^{J} \zeta_j(z)\eta_{n,j}$ for $n \in \{1,\dots,N\}$.
  function POISSON-ALIAS(z, $\{\zeta_j\}_{j\in\{1,\dots,J\}}$, $\{\eta_{n,j}\}_{n\in\{1,\dots,N\},j\in\{1,\dots,J\}}$)
    for $j \in \{1,\dots,J\}$ **do**
      Sample $S_j \sim \text{Poisson}(\sum_{n=1}^{N} \zeta_j(z)\eta_{n,j})$.
      Sample counts $\{K_{n,j}\}_{n\in\{1,\dots,N\}}$ from a multinomial distribution with S_j trials and event probabilities

$$\left\{\frac{\eta_{n,j}}{\sum_{n'=1}^{N}\eta_{n',j}}\right\}_{n\in\{1,\dots,N\}}$$

    end for
    return $\{K_n = \sum_j K_{n,j}\}_{n\in\{1,\dots,N\}}$.
  end function

Algorithm 26 is correct as for each j, the counts $\{K_{n,j}\}_{n\in\{1,\dots,N\}}$ are Poisson-distributed $K_{n,j} \sim \text{Poisson}(\zeta_j(z)\eta_{n,j})$ and therefore the sum is Poisson-distributed with rate parameter the sum of the constituent rates $K_n = \sum_j K_{n,j} \sim \text{Poisson}(\sum_j \zeta_j(z)\eta_{n,j})$. The event $K_n \geq 1$ occurs with probability $1 - \exp(-\kappa_n) = \bar{p}_n$ and thus the ratio $(1 - \alpha_n(z))/\bar{p}_n$ is a correct thinning probability for sampling B_n.

This latter method resembles the method of Fukui and Todo (2009), where they assume access to actual probabilities and not upper bounds.

4.4.3 Factorizing the event acceptance probability

As mentioned, there are two bottlenecks in Algorithm 23: one, the simulation of variables B_n, for which we can use rate bounds such as (4.14); and two, the simulation of the event kernel acceptance probability (4.13).

To address the simulation of the event kernel acceptance probability, we note simply that a factorization of this acceptance ratio — along the same lines as that done for the acceptance probability of the forward proposal $\Phi(z)$ — allows us to use the methods described in this chapter to efficiently simulate Bernoulli variables.

Recall that the event acceptance probability (4.13) was given as

$$\min\left\{1, \frac{q_B(\mathcal{S}(z^*), \mathcal{S}(z))}{q_B(z, z^*)} \prod_{n:B_n=1} \frac{[\rho_n(z^*) - \rho_n(\Phi \circ \mathcal{S}(z^*))]_+}{[\rho_n(z) - \rho_n(\Phi(z))]_+} \prod_{n:B_n=0} \frac{\min\{\rho_n(z^*), \rho_n(\Phi \circ \mathcal{S}(z^*))\}}{\min\{\rho_n(z), \rho_n(\Phi(z))\}}\right\}.$$

Factorizing the target distribution such that each observation is placed in its own factor, we have a new acceptance probability

$$\min\left\{1, \frac{q_B(\mathcal{S}(z^*), \mathcal{S}(z))}{q_B(z, z^*)}\right\} \prod_{n:B_n=1} \min\left\{1, \frac{[\rho_n(z^*) - \rho_n(\Phi \circ \mathcal{S}(z^*))]_+}{[\rho_n(z) - \rho_n(\Phi(z))]_+}\right\} \times \prod_{n:B_n=0} \min\left\{1, \frac{\min\{\rho_n(z^*), \rho_n(\Phi \circ \mathcal{S}(z^*))\}}{\min\{\rho_i(z), \rho_n(\Phi(z))\}}\right\}. \quad (4.15)$$

Much like the factorized acceptance probability for $\Phi(z)$, this latter acceptance probability can be interpreted as a number (in this case, $N + 2$) of independent Bernoulli simulations. We present Algorithm 27; here, the acceptance of the event proposal corresponding to (4.15) is computed in three stages:

1. simulate the rejection of the factor containing the ratio of proposal densities with probability $1 - \min\left\{1, \frac{q_B(\mathcal{S}(z^*), \mathcal{S}(z))}{q_B(z, z^*)}\right\}$

2. if that event is not rejected, we simulate rejection events of probability $1 - \min\left\{1, \frac{[\rho_n(z^*) - \rho_n(\Phi \circ \mathcal{S}(z^*))]_+}{[\rho_n(z) - \rho_n(\Phi(z))]_+}\right\}$ for each n where $B_n = 1$,

3. if those events are not rejected, then we simulate rejection events of probability $1 - \min\left\{1, \frac{\min\{\rho_n(z^*), \rho_n(\Phi \circ \mathcal{S}(z^*))\}}{\min\{\rho_n(z), \rho_n(\Phi(z))\}}\right\}$ for each n where $B_n = 0$.

Thus we suggest that one can make use of efficient procedures described in Algorithm 24 and Algorithm 25 to sample the event acceptance condition when necessary. Note that for the second stage of the event acceptance condition where we simulate $\{B'_n\}_{n \in V}$, we expect that V is small and thus number of variables sampled here will be small; as such this step may be inexpensive to compute even without using the efficient scheme presented here and there is little to be gained by a more sophisticated simulation scheme.

The factorization of the event acceptance probability is valid as we retain skew-detailed balance between the states z and z^* conditioned on the set $\{B_n\}_{n \in \{0,\ldots,N\}}$.

Algorithm 27 Discrete-time local bouncy particle sampler with factorized bounce acceptance probability.

for $n \in \{0, \dots, N\}$ **do**
  Sample $B_n \sim \text{Bernoulli}\left\{\left[\rho_n(z) - \rho_n(\Phi(z))\right]_+ / \rho_n(z)\right\}$.
end for
$V \leftarrow \{n \in \{0, \dots, N\} : B_n = 1\}$.
if $V = \emptyset$ **then**
  return $\Phi(z)$.
else
  Sample $z^* \sim q(z, \cdot)$.
  Sample $B_0' \sim \text{Bernoulli}\left(1 - \min\left\{1, \frac{q_B(\mathcal{S}(z^*), \mathcal{S}(z))}{q_B(z, z^*)}\right\}\right)$.
  for $i \in V$ **do**
    Sample $B_n' \sim \text{Bernoulli}\left(1 - \min\left\{1, \frac{[\rho_n(z^*) - \rho_n(\Phi \circ \mathcal{S}(z^*))]_+}{[\rho_n(z) - \rho_n(\Phi(z))]_+}\right\}\right)$.
  end for
  for $n \in \{0, \dots, N\} \setminus V$ **do**
    Sample $B_n' \sim \text{Bernoulli}\left(1 - \min\left\{1, \frac{\min\{\rho_n(z^*), \rho_n(\Phi \circ \mathcal{S}(z^*))\})}{\min\{\rho_n(z), \rho_n(\Phi(z))\}}\right\}\right)$.
  end for
  if $B_n' = 1$ for any $n \in \{0, \dots, N\}$ **then**
    return $\mathcal{S}(z)$.
  else
    return z^*.
  end if
end if

4.5 Efficient acceptance probabilities for the Metropolis-Hastings algorithm

So far we have discussed methods of adapting the discrete-time piecewise-deterministic Markov chain Monte Carlo algorithm to scale well with large data. Here we present the Metropolis-Hastings version to emphasize that these schemes may be used outside of the context of piecewise-deterministic Markov chain Monte Carlo methods, and to provide some improvements which are specific to this version; otherwise this is simply an implementation of the factorized Metropolis-Hastings algorithm (Algorithm 15) to the factorization (4.6).

To deal with the proposal, we include the ratio of proposal densities in the 0th factor, similarly to the manner in which we dealt with the kinetic energy previously. The algorithm is given as Algorithm 28, we call this *scalable Metropolis-Hastings* after Cornish, Vanetti, Bouchard-Côté, Deligiannidis, and Doucet (2019).

Algorithm 28 Scalable Metropolis-Hastings

A single step of the scalable Metropolis-Hastings Algorithm. Requires $\zeta(x, x^*)$ and η_n such that

$$\left[\{f_n(x^*) - \hat{f}_n(x^*)\} - \{f_n(x) - \hat{f}_n(x)\}\right]_+ = \kappa_n \leq \bar{\kappa}_n = \zeta(x, x^*)\eta_n.$$

function ScalableMHStep(x)
  Sample $x^* \sim q(x, \cdot)$.
  with probability

$$1 - \min\left\{1, \frac{\hat{\pi}(y|x^*)\pi_0(x^*)q(x^*, x)}{\hat{\pi}(y|x)\pi_0(x)q(x, x^*)}\right\}$$

    return x.
  otherwise
    Simulate a count $S \sim \text{Poisson}(\zeta(x, x^*) \sum_n \eta_n)$.
    for $i \in \{1, \ldots, S\}$ **do**
      Using an alias table, simulate observation index k with probability

$$\mathbb{P}[k = \ell] = \frac{\eta_\ell}{\sum_{n'} \eta_{n'}}.$$

      with probability

$$\frac{[\{f_k(x^*) - \hat{f}_k(x^*)\} - \{f_k(x) - \hat{f}_k(x)\}]_+}{\zeta(x, x^*)\eta_k}$$

        return x.
    end for
    return x^*.
end function

This algorithm avoids the explicit simulation of the set of rejected factors $\{B_n\}_{n\in\{0,\ldots,N\}}$ as there is no need to re-use this set in an event kernel. Instead, the algorithm can be understood as simulating from Poisson variables $\{S_n\}_{n\in\{1,\ldots,N\}}$ with rate $\left[\left\{f_n(x^*) - \hat{f}_n(x^*)\right\} - \left\{f_n(x) - \hat{f}_n(x)\right\}\right]_+$ by thinning from a dominating Poisson rate $\zeta(x, x^*)\eta_n$. The probability that the algorithm does not return x due to the acceptance of a point, equivalently the probability that all of these counts are zero, is

$$\prod_n \exp\left(-\left[\left\{f_n(x^*) - \hat{f}_n(x^*)\right\} - \left\{f_n(x) - \hat{f}_n(x)\right\}\right]_+\right)$$

which is the correct acceptance probability for factors $\{1, \ldots, N\}$.

4.5.1 $\hat{\pi}$-reversible proposals

In the Metropolis-Hastings context we can choose a variety of proposals, some of which make little sense in a piecewise-deterministic context. This includes a family of proposals which are reversible with respect to the global approximation

$$\hat{\pi}(x) \propto \pi_0(x)\pi(y|x) \propto \exp\left(-f_0(x) - \sum_n \hat{f}_n(x)\right),$$

which we call $\hat{\pi}$-reversible proposals. Specifically, these proposals satisfy

$$\hat{\pi}(x)q(x,x') = \hat{\pi}(x')q(x',x). \tag{4.16}$$

In this case, the acceptance probability associated with the 0th factor becomes one:

$$\min\left\{1, \frac{\hat{\pi}(x')q(x',x)}{\hat{\pi}(x)q(x,x')}\right\} = 1 \tag{4.17}$$

and we only need to compute the acceptance probabilities associated with the per-observation factors.

These are of particular relevance when considering the second-order Taylor-series approximation, where the approximation $\hat{\pi}$ is a normal distribution. In this case, we expect that for a model where the posterior distribution converges to a normal distribution, then the bounds on the observation sub-sampling will vanish and we will be left with an algorithm that never rejects a proposal.

For this normal approximation, we use autoregressive proposals of the form

$$q(x,x') = \mathcal{N}(x'; \rho x, (1-\rho^2)\Sigma_{\hat{\pi}})$$

where $\Sigma_{\hat{\pi}}$ is the covariance of the normal distribution $\hat{\pi}$, for some $\rho \in (-1,1)$.

4.6 Application to logistic regression

The mechanisms described thus far are predicated on certain approximations and bounds being available for a model of interest. As is done for a large number of antecedents (Banterle et al. 2015, Maclaurin and Adams 2015, Bardenet et al. 2017, Bierkens et al. 2019a), we demonstrate these methods on a logistic regression model.

We have for the logistic regression case the per-observation potential (i.e. negative log-likelihood)

$$f_n(x) = -\log \sigma(y_n \langle \xi_n, x \rangle).$$

where $\sigma(z) = 1/(1+\exp(-z))$. The covariates are denoted ξ_n and $y_n \in \{-1, 1\}$ the class label (many sources use an alternative expression where $y_n \in \{0, 1\}$ but we find the above to be more convenient).

A useful property of the logistic regression model is that the derivatives are uniformly bounded in x. Take for the sake of exposition $s_n = y_n \langle \xi_n, x \rangle$. We rely on the result that $\frac{\partial}{\partial a}\sigma(b) = \sigma(b)(1-\sigma(b))\frac{\partial}{\partial a} b$.

$$\begin{aligned}
\frac{\partial}{\partial x_i} f_n(x) &= (\sigma(s_n) - 1) y_n \xi_{ni} \\
\frac{\partial^2}{\partial x_i \partial x_j} f_n(x) &= \xi_{ni}\xi_{nj}\sigma(s_n)(1-\sigma(s_n)) \\
\frac{\partial^3}{\partial x_i \partial x_j \partial x_k} f_n(x) &= y_n \xi_{ni}\xi_{nj}\xi_{nk}\sigma(s_n)(1-\sigma(s_n))(1-2\sigma(s_n))
\end{aligned}$$

the magnitude of which can be bounded

$$\begin{aligned}
\left| \frac{\partial}{\partial x_i} f_n(x) \right| &\leq |\xi_{ni}| \\
\left| \frac{\partial^2}{\partial x_i \partial x_j} f_n(x) \right| &\leq \frac{1}{4} |\xi_{ni}||\xi_{nj}| \\
\left| \frac{\partial^3}{\partial x_i \partial x_j \partial x_k} f_n(x) \right| &\leq \frac{1}{6\sqrt{3}} |\xi_{ni}||\xi_{nj}||\xi_{nk}|.
\end{aligned}$$

Note that in all cases we can attain a coordinate-independent bound by taking the maximum over coordinates, e.g.

$$\left| \frac{\partial^2}{\partial x_i \partial x_j} f_n(x) \right| \leq \frac{1}{4} (\max_k |\xi_{nk}|)^2.$$

4.6.1 For continuous-time rates

These bounds can be used to derive a Lipschitz bound for (4.9), as done by Bierkens et al. (2019a). By the mean value theorem and Cauchy-Schwarz

$$\begin{aligned}
\left| \frac{\partial}{\partial x_i} f_n(x) - \frac{\partial}{\partial x_i} f_n(y) \right| &= \left| \langle \nabla \frac{\partial}{\partial x_i} f_n(c(x-y)+y), x-y \rangle \right| \\
&\leq \| \nabla \frac{\partial}{\partial x_i} f_n(c(x-y)+y) \|_2 \|x-y\|_2
\end{aligned}$$

where $c \in (0,1)$. Introducing our bounds on the second partial derivatives,

$$\|\nabla \frac{\partial}{\partial x_i} f_n(c(x-y)+y)\|_2 \leq \frac{1}{4}|\xi_{ni}|\|\xi_n\|_2$$

allowing $L_i \leq \frac{1}{4}|\xi_{ni}|\|\xi_n\|_2$ for $p = 2$.

4.6.2 For discrete-time residuals

With $\hat{f}^{(K)}$ as the Kth Taylor-series approximation of $\hat{f}^{(K)}$, we use the mean-value form of the remainder

$$\begin{aligned} f_n(x) - \hat{f}_n^{(K)}(x) &= \sum_{\alpha:|\alpha|=K+1} \frac{1}{\alpha!} \partial^\alpha f_n(c(x-\hat{x})+\hat{x})(x-\hat{x})^\alpha \\ &\leq \sum_{\alpha:|\alpha|=K+1} \frac{1}{\alpha!} |\partial^\alpha f_n(c(x-\hat{x})+\hat{x})||x-\hat{x}|^\alpha. \end{aligned}$$

for some $c \in (0,1)$. This can be bounded by replacing the partial derivative with the bound of the appropriate order. We give the specific cases for $K = 1$ and $K = 2$.

- For $K = 1$,

$$\begin{aligned} f_n(x) - \hat{f}_n^{(1)}(x) &\leq \frac{1}{2} \sum_{i=1}^d \sum_{j=1}^d \left| \frac{\partial^2}{\partial x_i \partial x_j} f_n(c(x-\hat{x})+\hat{x}) \right| |x_i - \hat{x}_i||x_j - \hat{x}_j| \\ &\leq \frac{1}{8} \left(\max_j |\xi_{nj}| \right)^2 \|x-\hat{x}\|_1^2 \end{aligned}$$

  so for the bound on rejection probability (4.14) we have

$$|R_n^{(1)}(x^*)| + |R_n^{(1)}(x)| \leq \frac{1}{8} \left(\max_j |\xi_{nj}| \right)^2 \left[\|x-\hat{x}\|_1^2 + \|x^*-\hat{x}\|_1^2 \right]$$

  so we can take a single alias table $\zeta(x,x^*) = \frac{1}{8}\left[\|x-\hat{x}\|_1^2 + \|x^*-\hat{x}\|_1^2\right]$ and $\eta_n = \max_j |\xi_{nj}|^2$. Alternatively, we can use the coordinate-independent bound to obtain either

$$f_n(x) - \hat{f}_n^{(1)}(x) \leq \frac{1}{8} \sum_{i=1}^d \sum_{j=1}^d |\xi_{ni}||\xi_{nj}||x_i - \hat{x}_i||x_j - \hat{x}_j|$$

  allowing $d \times d$ alias tables with coefficients

$$\zeta_{ij}(x,x^*) = \frac{1}{8} \left\{ |x_i - \hat{x}_i||x_j - \hat{x}_j| + |x_i^* - \hat{x}_i||x_j^* - \hat{x}_j| \right\}$$

and $\eta_{ij,n} = |\xi_{ni}||\xi_{nj}|$. Or,

$$f_n(x) - \hat{f}_n^{(1)}(x) \leq \frac{1}{8}(\max_j |\xi_{nj}|)\|x - \hat{x}\|_1 \sum_{i=1}^{d} |\xi_{ni}||x_i - \hat{x}_i|.$$

allowing d alias tables with coefficients

$$\zeta_i(x, x^*) = \frac{1}{8}\left\{\|x - \hat{x}\|_1 |x_i - \hat{x}_i| + \|x^* - \hat{x}\|_1 |x_i^* - \hat{x}_i|\right\}$$

and $\eta_{i,n} = (\max_j |\xi_{nj}|)|\xi_{ni}|$.

- For $K = 2$,

$$f_n(x) - \hat{f}_n^{(1)}(x) \leq \frac{1}{6}\sum_{i=1}^{d}\sum_{j=1}^{d}\sum_{k=1}^{d}\left|\frac{\partial^3}{\partial x_i \partial x_j \partial x_k} f_n(c(x - \hat{x}) + \hat{x})\right| |x_i - \hat{x}_i||x_j - \hat{x}_j||x_k - \hat{x}_k|$$

for which we give only the bound appropriate for a single alias table, while noting that up to d^3 tables could be used in the same fashion as we derived more detailed tables for $K = 1$.

$$f_n(x) - \hat{f}_n^{(1)}(x) \leq \frac{1}{36\sqrt{3}}\left(\max_j |\xi_{nj}|\right)^3 \|x - \hat{x}\|_1^3$$

allowing a single alias table where $\zeta(x, x^*) = \frac{1}{36\sqrt{3}}\left[\|x - \hat{x}\|_1^3 + \|x^* - \hat{x}\|_1^3\right]$ and $\eta_n = (\max_j |\xi_{nj}|)^3$.

We find that the finer-grained tables do not significantly improve performance for the datasets used in our experiments, and so suggest that the versions using a single alias table are adequate.

4.7 Ergodicity and scaling

A pitfall of factorizing the potential is that the resulting Markov chain Monte Carlo algorithm may not be geometrically ergodic — even though the vanilla Metropolis-Hastings version might be — unless the acceptance rates are bounded from below, as shown by Banterle et al. (2015; Condition 1). They repair their scheme by clamping the first $N - 1$ factors to a range $(b, \frac{1}{b})$ for some $b \in (0, 1]$ and suitably modifying the Nth factor.

Cornish, Vanetti, Bouchard-Côté, Deligiannidis, and Doucet (2019) provide an analogous condition for the scalable Metropolis-Hastings algorithm. Noting that the rejection rates are upper bounded by the aggregate Poisson sampling rate $\zeta(x, x^*) \sum_n \eta_n$, the algorithm is modified to simply fall back to the standard Metropolis-Hastings acceptance probability whenever $\zeta(x, x^*) \sum_n \eta_n$ exceeds an arbitrary threshold. Since this is a symmetric function of x and x^* (with our choices of bounds) the algorithm maintains detailed balance. The threshold is chosen such that when the expected number of observations visited would exceed N (the actual number of observations) the algorithm falls back to standard Metropolis-Hastings. In this manner the computational performance of the algorithm is improved in expectation while also ensuring ergodicity.

The scaling properties of the algorithm with increasing N are also reliant on the quantity $\zeta(x, x^*) \sum_n \eta_n$, but in the sense that this also determines the computational cost of the algorithm. Using the bounds derived from Taylor-series residuals, $\zeta(x, x^*)$ is proportional to $\|x' - \hat{x}\|_1^{K+1} + \|x - \hat{x}\|_1^{K+1}$. Due to concentration of the posterior (and assuming $\hat{x}$ is close to the mode of the posterior) the expectation of $\zeta(x, x^*)$ decreases with increasing N. If $\sum_n \eta_n$ does not commensurately increase, then sublinear scaling is attained. See Cornish, Vanetti, Bouchard-Côté, Deligiannidis, and Doucet (2019) for details and further conditions, where it is shown that for the logistic regression case the expected number of data accessed per iteration scales as $O(N^{-(K-1)/2})$, i.e. when $K = 1$ we have $O(1)$ and for $K = 2$ we have $O(N^{-1/2})$.

4.8 Revisiting Firefly Monte Carlo

The Firefly Markov chain Monte Carlo (or "FlyMC") algorithm introduced by Maclaurin and Adams (2015) (see also Mak (2005)) demonstrated the first Markov chain Monte Carlo algorithm which accesses a subsample of the data in each iteration while remaining exact. FlyMC uses auxiliary variables to "turn on" and "turn off" observations while still targeting a distribution that has the correct posterior as the marginal distribution.

We note here that the techniques and bounds discussed in this chapter may be used to efficiently update these subsamples. If we do so in every iteration of the algorithm, then we can avert concerns about the mixing of these auxiliary variables and interpret the algorithm as an Markov chain Monte Carlo algorithm on the state variable x, albeit with a nonstandard acceptance probability.

First, we describe the algorithm. The Firefly algorithm targets a joint distribution on the parameter x augmented by N binary variables $z = \{z_1, z_2, \ldots, z_N\} \in \{0,1\}^N$, with density

$$\pi(x, z) = \pi_0(x) \prod_{n=1}^{N} B_n(x)^{\mathbb{I}(z_n=0)} \left[L_n(x) - B_n(x)\right]^{\mathbb{I}(z_n=1)} \tag{4.18}$$

using the notation of the original paper (Maclaurin and Adams 2015) where $L_n(x) = \pi(y_n|x)$ is the likelihood and B_n is a lower bound on the likelihood such that $L_n(x) - B_n(x) \geq 0$. The z_n effectively determine which observations are currently active ("bright" in the nomenclature of Maclaurin and Adams (2015)), as the conditional distribution of x is

$$\pi(x|z) \propto \pi_0(x) \left[\prod_{n:z_n=0} B_n(x)\right] \left[\prod_{n:z_n=1} L_n(x) - B_n(x)\right], \tag{4.19}$$

so a Metropolis-Hastings update of x conditional on z only requires evaluating the actual likelihood of the observation for which $z_n = 1$.

The z_n are independent conditioned on x with density

$$\pi(z_n|x) = \text{Bernoulli}\left(z_n; \frac{L_n(x) - B_n(x)}{L_n(x)}\right) \tag{4.20}$$

suggesting that so long as $(L_n(x) - B_n(x))/L_n(x)$ is small (i.e. the bound is tight relative to $L_n(x)$), the number of z_n equal to one will be small, and each Metropolis-Hastings update will require few iterations.

While the conditional density of x relies only on those $L_n(x)$ where $z_n = 1$, the expression (4.19) still requires the evaluation of N terms as the $B_n(x)$ must also be evaluated. The approach taken by (Maclaurin and Adams 2015) is to assume that $\prod_{n=1}^{N} B_n(x)$ can be evaluated in $O(1)$ and rearrange the density to

$$\pi(x|z) \propto \pi_0(x) \left[\prod_{n=1}^{N} B_n(x)\right] \left[\prod_{n:z_n=1} \frac{L_n(x) - B_n(x)}{B_n(x)}\right].$$

The algorithm for updating x conditional on z is given by Algorithm 29. Maclaurin and Adams (2015) give two algorithms for updating z conditional on x: Algorithm 30, which uses a number of steps wherein a random z_k is selected and updated using Gibbs sampling; here the number of z_k to update can be tuned. The second algorithm is given as Algorithm 30, which iterates through the set of z updating each using Metropolis-Hastings; here the probability of proposing modifications of z_n can be tuned.

Algorithm 29 FlyMC: updating x

Given x and set $z = \{z_1, z_2, \ldots, z_N\}$, sample the next state of the chain x'.
function FlyMCXStep(x, $\{z_n\}_{n\in\{1,\ldots,N\}}$)
  Sample $x^* \sim q(x, \cdot)$.
  with probability

$$\min\left\{1, \frac{\pi_0(x^*)q(x^*,x)}{\pi_0(x)q(x,x^*)}\prod_{n=1}^{N}\frac{B_n(x^*)}{B_n(x)}\prod_{n:z_n=1}\frac{L_n(x^*)/B_n(x^*)-1}{L_n(x)/B_n(x)-1}\right\}$$

    return x^*.
  otherwise
    return x.
end function

Algorithm 30 FlyMC: updating z by resampling a subset

Given state x, z, and resampling fraction parameter N_r, sample the next state of the chain z'.
function FlyMCZStep-Subset(x, $\{z_n\}_{n\in\{1,\ldots,N\}}$)
  for $j \in \{1, \ldots, N_r\}$ **do**
    $k \sim$ Uniform($\{1, \ldots, N\}$).
    $z_k \sim$ Bernoulli$(1 - B_k(x)/L_k(X))$.
  end for
  return z.
end function

4.8.1 Efficient subsampling for FlyMC

The FlyMC algorithm uses updates of the auxiliary variables which are $O(N)$. So despite the algorithm's capability to subsample exactly, updates of the subsample do not scale sublinearly. It remains unclear as to whether the cost of a single

Algorithm 31 FlyMC: updating z with Metropolis-Hastings

Given state x, z, and resampling parameters $q_{0\to1}$ and $q_{1\to0}$, sample the next state of the chain z'.

function FlyMCZStep-Implicit(x, z)
  for $n \in \{1, \dots, N\}$ **do**
    if $z_n = 1$ **then**
      with probability $q_{1\to0}$
        with probability $\min\{1, \frac{B_n(x)q_{0\to1}}{(L_n(x)-B_n(x))q_{1\to0}}\}$
          $z_n \leftarrow 0$.
    else
      with probability $q_{0\to1}$
        with probability $\min\{1, \frac{(L_n(x)-B_n(x))q_{1\to0}}{B_n(x)q_{0\to1}}\}$
          $z_n \leftarrow 1$.
    end if
  end for
  return z.
end function

iteration of the algorithm is truly sublinear: while the algorithm may explore the conditional on z efficiently, it is unclear how often one needs to update the z such that the chain mixes effectively.

In this section, we describe how the updates to z may be done using techniques developed in this chapter. These are sublinear in N and thus the z can be updated in each iteration; when doing so, the algorithm can be seen as a typical Markov chain on x but with a particular acceptance probability.

To motivate this, we compare FlyMC to the framework developed in Section 3.5 and specifically to the joint density given by (3.26). Noting that the lower bound on the observation likelihood provides an upper bound on the potential, we see that the joint density used in FlyMC is an instance of the sparsity-inducing joint density (3.26).

We show that the ideas of this chapter can be applied to the FlyMC algorithm for updating the z. When the approximation is good (so $\mathbb{P}[z_n = 1] = 1 - B_n(x)/L_n(x)$ is small), then we expect that $z_n = 1$ for only a few values of n. If we can further bound the probabilities $1 - B_n(x)/L_n(x)$ then we can use the same Poisson-based simulation ideas as we did for simulating the acceptance probabilities of

the piecewise-deterministic Markov chain Monte Carlo algorithms; specifically, we can sample the z_n using Algorithm 25.

The specific lower bound used in the original FlyMC paper does not seem amenable to a closed-form bound for the probability $1 - B_n(x)/L_n(x)$ and we have not been able to produce such a bound. However, we note that the Taylor series approximation can be used to produce a lower bound B_n, and for this bound we can derive the necessary bound on $1 - B_n(x)/L_n(x)$ to implement subsampling.

We take an upper bound of the potential which is the Taylor series approximation of order K plus the residual bound $R_n^{(K)}(x)$. Denote the potential bound $\bar{f}_n^{(K)}$ so $B_n(x) = \exp\left(-\bar{f}_n^{(K)}(x)\right)$ and

$$\bar{f}_n^{(K)}(x) = \hat{f}_n^{(K)}(x) + |R_n^{(K)}(x)|, \tag{4.21}$$

we guarantee that $\bar{f}_n^{(K)}(x) \geq f_n(x)$ as $f_n(x) - \hat{f}_n(x) = R_n^{(K)}(x)$. The bound is amenable to computing $\bar{f}^{(K)}(x) = \sum_n \bar{f}_n^{(K)}(x)$; this is

$$\begin{aligned}\bar{f}^{(K)}(x) &= \sum_n \sum_{k=0}^{K} \frac{1}{\alpha!} \partial^\alpha f_n(\hat{x})(x - \hat{x})^\alpha + \sum_n \left| \sum_{\alpha:|\alpha|=K+1} \frac{1}{\alpha!} \partial^\alpha f_n((1-c_n)\hat{x} + cx)(x-\hat{x})^\alpha \right| \\ &= \sum_{k=0}^{K} \frac{1}{\alpha!} [\partial^\alpha \sum_n f_n(\hat{x})](x-\hat{x})^\alpha + \sum_n \sum_{\alpha:|\alpha|=K+1} \frac{1}{\alpha!} \left[\sum_n |\partial^\alpha f_n((1-c_n)\hat{x} + cx)| \right] |x - \hat{x}|^\alpha.\end{aligned}$$

For example, taking $K = 2$ yields

$$\begin{aligned}\bar{f}^{(K)}(x) = f(\hat{x}) &+ \langle \sum_n \nabla f_n(\hat{x}), x - \hat{x} \rangle + \frac{1}{2}(x-\hat{x})^T (\sum_n H_n(\hat{x}))(x-\hat{x}) \\ &+ \sum_{i=1}^{d} \sum_{j=1}^{d} \sum_{k=1}^{d} |x_i - \hat{x}_i||x_j - \hat{x}_j||x_k - \hat{x}_k| \sum_n \left| \frac{\partial^3}{\partial x_i \partial x_j \partial x_k} f_n((1-c)\hat{x} + cx) \right|\end{aligned}$$

where H_n is the Hessian of f_n evaluated at $\hat{x}$.

It remains to bound the conditional probabilities of the z_n (4.20). To use our Poisson sampling techniques, we need to find $\bar{\kappa}_n$ such that $\bar{\kappa}_n \geq \kappa_n = -\log(1 - \pi(z_n = 1|x))$. We derive the expression for κ_n

$$\begin{aligned}\kappa_n &= -\log\left(1 - \left[1 - \frac{B_n(x)}{L_n(x)}\right]\right) \\ &= -\log B_n(x) + \log L_n(x) \\ &= \bar{f}_n^{(K)}(x) - f_n(x) \\ &\leq 2|R_n^{(K)}(x)|\end{aligned} \tag{4.22}$$

which is comparable to the bound we arrived at for the factorized acceptance probabilities (4.14). As with the factorized version, we can derive bounds for logistic regression which allow the use of alias tables to efficiently simulate from the necessary Poisson counts.

We give the modified algorithm in Algorithm 32, which we call the *Poisson firefly* algorithm.

Algorithm 32 Poisson firefly: FlyMC with Poisson subsampling of z.

Given current state x, sample next state x'.

function PoissonFirefly(x)

Use Algorithm 25 with $\kappa_n = -\log(B_n(x)/L_n(x))$ and bounds (4.22) to sample $\{z_n\}_{n\in\{1,\ldots,N\}}$.

Sample $x^* \sim q(x, \cdot)$.

with probability

$$\min\left\{1, \frac{\pi_0(x^*)q(x^*,x)}{\pi_0(x)q(x,x^*)} \prod_{n=1}^{N} \frac{B_n(x^*)}{B_n(x)} \prod_{n:z_n=1} \frac{L_n(x^*)/B_n(x^*) - 1}{L_n(x)/B_n(x) - 1}\right\}$$

return x^*.

otherwise

return x.

end function

4.8.2 Ergodicity and scaling

The theoretical properties of the Poisson firefly algorithm remain to be determined. We note one interesting failure case: we find for *some* of the dataset sizes in our experiment that a version of the Poisson firefly algorithm using first-order Taylor series approximations never accepts any proposal when started at the mode. However, the version using a second-order Taylor series performs adequately. We conjecture that in some situations the bounds are not tight enough to accept any point very far from $\hat{x}$; we leave a thorough analysis to future work.

4.9 Numerical results

We run several algorithms on a 10-dimensional logistic regression model. The data are generated randomly: the data-generating regression parameter is $x^\dagger =$

$(1, 1, \ldots, 1)$, the covariates ξ_n are i.i.d. Gaussian distributed $\xi_n \sim \mathcal{N}(0, I_{10})$, and the labels are sampled from the likelihood $y_n \sim \text{Bernoulli}(\sigma(\langle \xi_n, x^\dagger \rangle))$. We use a improper prior on x which is uniform on $\mathbb{R}^{10}$.

In the graphs, "MH" is standard Metropolis-Hastings, "SMH" is scalable Metropolis-Hastings (Algorithm 28), and "PF" is Poisson firefly (Algorithm 32); the numerical suffixes indicate the order of Taylor-series approximation used.

Figure 4.1 shows the number of likelihood evaluations per iteration, demonstrating the desired sublinear scaling for all algorithms. For all algorithms we use a random walk proposal with a covariance matrix equal to the inverse Hessian at the mode of the distribution, as such we expect that the proposal should be adequately scaled to the posterior distribution for all values of N. For SMH-1, we see that the number of likelihoods per iteration is roughly constant; for SMH-2 and PF-2, we see that the number of likelihoods per iteration is actually decreasing with N. The PF-2 algorithm is not as effective as the SMH-2 algorithm; this may be attributable to the requirement of using a lower bound, which may be less efficient than the usual bound. Zig-zag is not included here as the concept of an iteration is not comparable between discrete and continuous time.

Figure 4.2 shows the effective sample sizes per second for the same model as above. The effective sample size is related to the variance of estimates obtained from the output of the sampling algorithm. We define the effective sample size as the number of iterations T divided by the autocorrelation time τ. The autocorrelation time itself is defined for an integrable function $\phi : \mathcal{X} \to \mathbb{R}$ with expectation $\mu_\phi = \mathbb{E}_\pi[\phi(X)]$ by

$$\tau_\phi = \frac{V_\phi}{\sigma_\phi^2}$$

where $\sigma_\phi^2 = \text{Var}_\pi[\phi(X)]$ is the variance of ϕ under the target π and V_ϕ is the asymptotic variance of the Monte Carlo estimate of ϕ. We estimate this latter variance using overlapping batch means (Flegal et al. 2010), which divides the samples into overlapping "batches" of length m, using the estimate

$$\widehat{V_\phi} = \frac{Tm}{(T-m)(T-m+1)} \sum_{j=1}^{T-m+1} (\widehat{\mu_\phi}^{(j)} - \widehat{\mu_\phi})^2$$

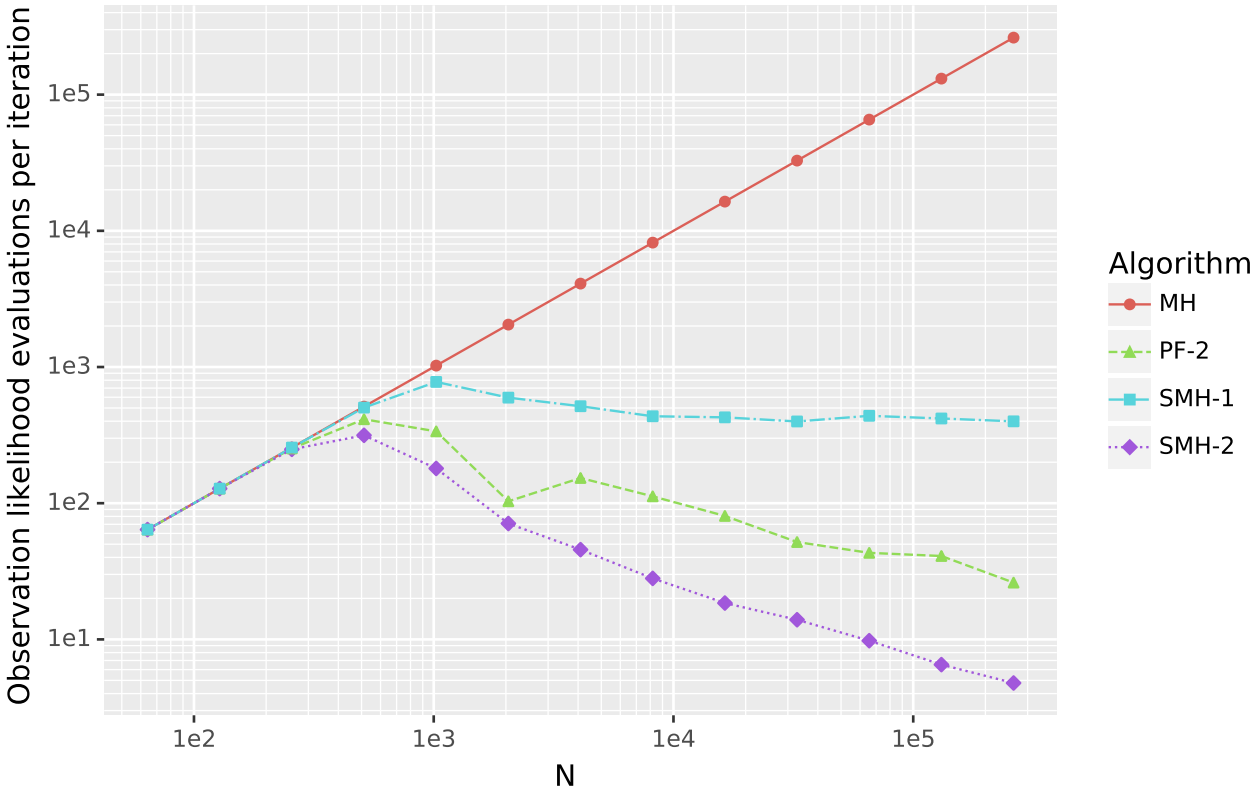
where T is the length of the chain, and m is the length of a batch. Denoting the samples generated as $\{x_i\}_{i\in[T]}$, $\widehat{\mu_\phi}^{(j)} = m^{-1}\sum_{k=j}^{j+m-1}\phi(x_k)$ is the estimate of μ_ϕ from each batch. $\widehat{\mu_\phi}$ is a more precise estimate of $\mathbb{E}_\pi[\phi(x)]$, which we obtain for each unique posterior distribution from a single long run. The autocorrelation time is finally esimated by $\widehat{\tau_\phi} = \widehat{V_\phi}/\widehat{\sigma_\phi^2}$, where $\widehat{\sigma_\phi^2}$ is estimated from a long run, similarly to $\widehat{\mu_\phi}$. In these plots we use $\phi(x) = x^{(1)}$, i.e. the mean of the first coordinate.

For Figure 4.2, a random walk proposal is used (as for Figure 4.1), with covariance matrix equal to the inverse of the Hessian at the mode of the posterior. Each sampler was run for 20000 iterations, and the effective sample size per second was averaged over 20 such runs; the error bars on the plot show the standard errors of the estimate of the effective sample size. Following the guidance of Flegal et al. (2010), the overlapping batch means estimator was implemented with batch size $m = 20000^{1/3}$. Both $\widehat{\mu_\phi}$ and $\widehat{\sigma_\phi^2}$ were estimated from a longer run of SMH-2 for 10^7 iterations.

In terms of practical performance, the effective samples per second shows that none of the algorithms are competitive with vanilla Metropolis-Hastings until the dataset is at least 1024 observations. This is attributable to the extra computational effort in e.g. thinning the Poisson process (note that setup times are not included in this measurement). For very low values of N the various algorithms perform identically as they often fall-back to the vanilla Metropolis-Hastings acceptance probability as described in Section 4.7.

The results suggest that the amount of data required to see any computational benefit is quite high. The scaling of the algorithm relies on the concentration of the posterior to a normal distribution, and it is clear that for increasing dataset sizes, any sort of sampling method becomes irrelevant as the normal approximation serves as an adequate representation of the posterior. Figure 4.3 shows the simulated posterior as compared with the normal approximation, illustrating the convergence of the posterior to the approximation as N increases. We see a qualitative difference between the posterior and the approximation for $N = 2048$, the lowest N for which we saw superior performance for SMH-2. Nonetheless, is is clear that the posterior

Figure 4.1: Likelihood evaluations per MCMC iteration, as a function of dataset size N.



is already very close to the normal approximation after $N = 8192$, but we make no attempt to quantify these differences here.

Figure 4.4 illustrates the behaviour of the algorithms when using a $\hat{\pi}$-reversible proposal. In all cases we see similar scaling with N as for the random walk proposal.

Besides the continuous-time zig-zag algorithm, we do not actually include here any piecewise-deterministic algorithms. Since these algorithms only exhibit an improvement over standard algorithms in a large data setting where the posterior is approximately normal, we suggest that the piecewise-deterministic algorithms offer no advantage over the $\hat{\pi}$-adapted algorithms, and only offer additional complexity in implementing event kernels.

Figure 4.2: ESS per second for a random walk proposal, as a function of N

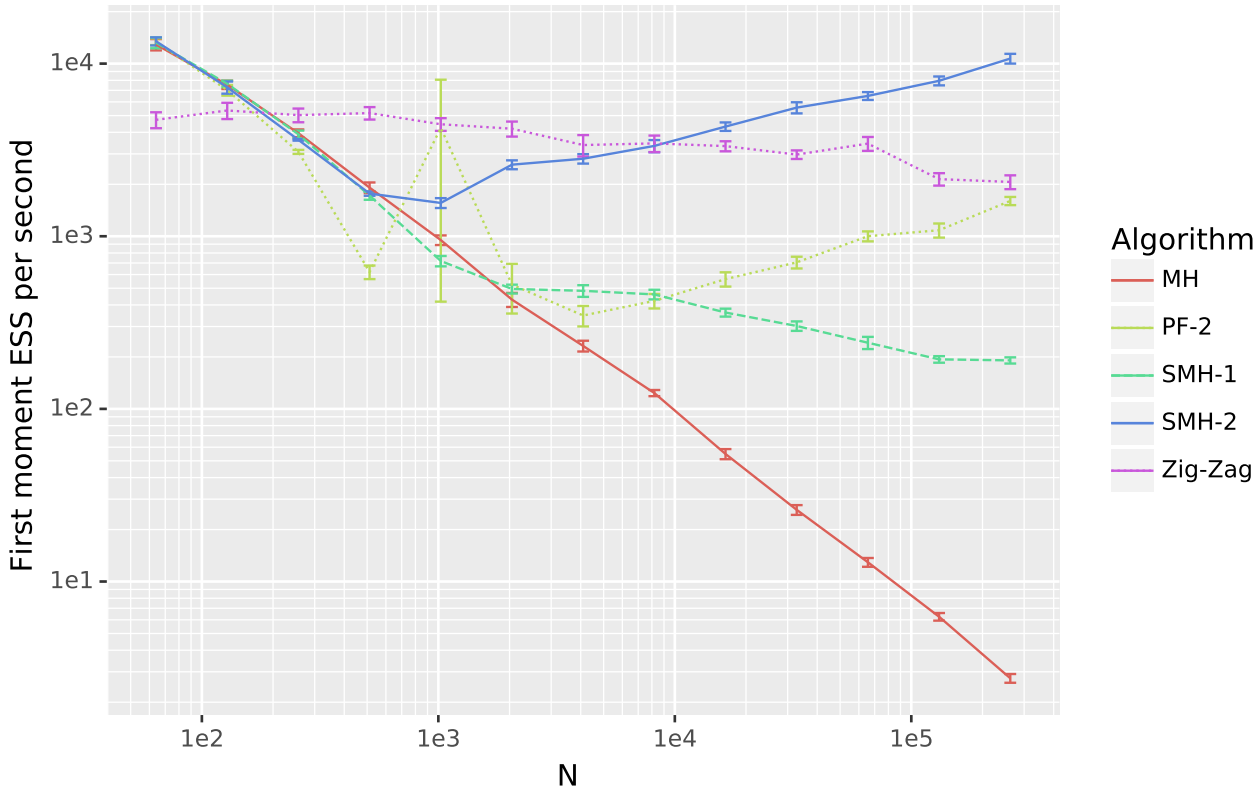


Figure 4.3: Posterior samples (blue histogram) versus marginal of the normal approximation for first regression coefficient x_1.

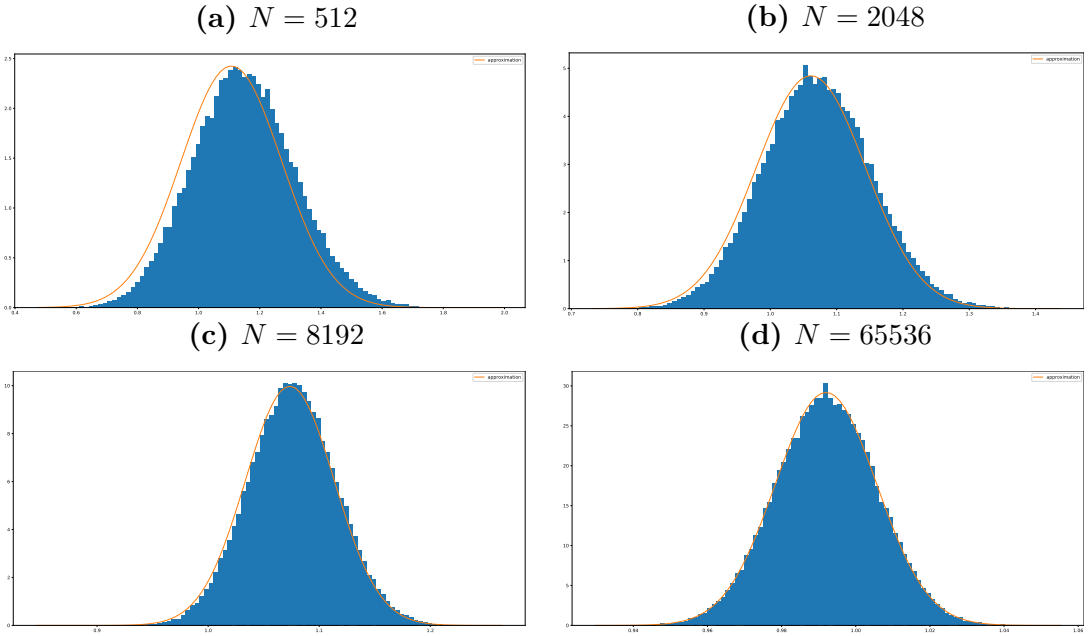
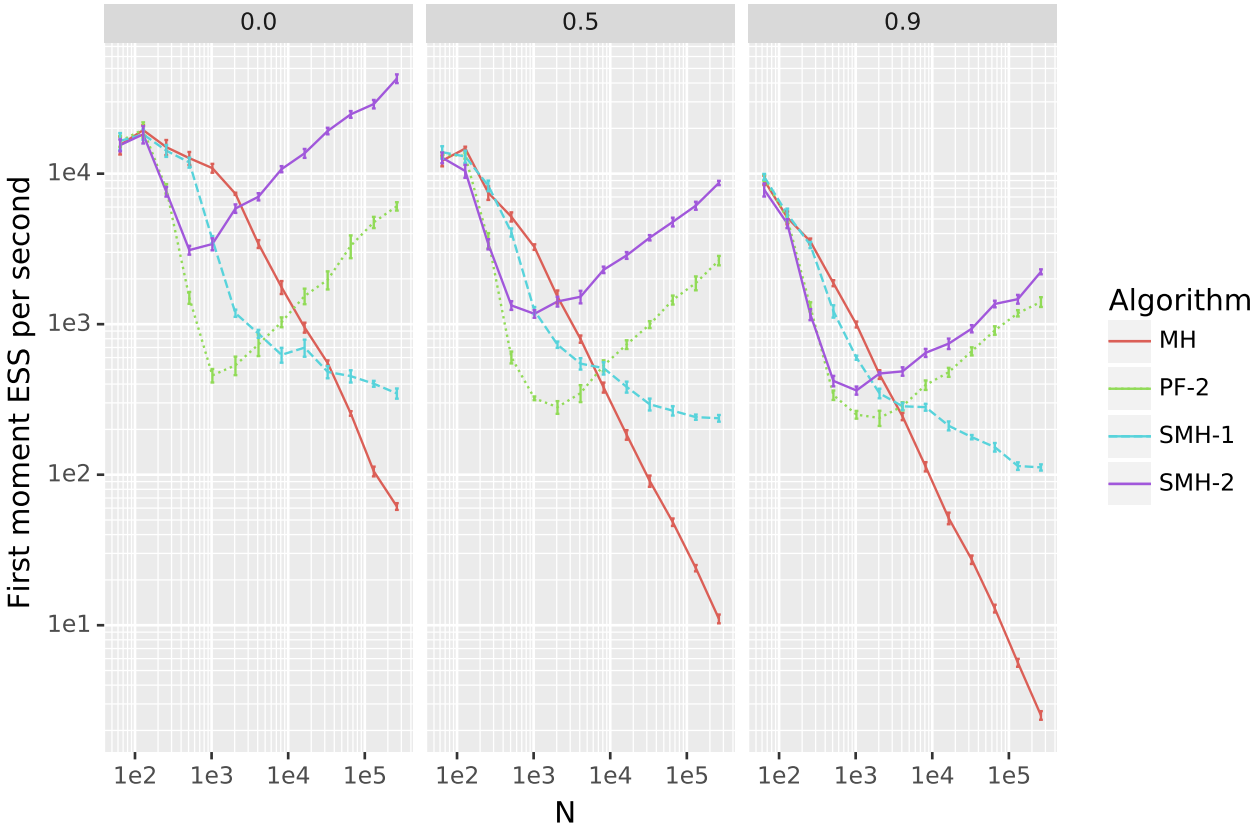


Figure 4.4: ESS per second for a $\hat{\pi}$-autoregressive proposal, as a function of N



5

Piecewise-deterministic Markov chain Monte Carlo for sparsely connected models

One promising application of continuous-time algorithms is the setting we describe as *sparsely connected* models. These are models which have a potential of the form

$$U(x) = \sum_{i=1}^{m} U_i(x_{S_i}) \tag{5.1}$$

where S_i is a set of indices indicating the x coordinates which are used in computing factor U, so $x_{S_i} = \{x_j : j \in S_i\}$ is a subset of the components of x. In order to be sparsely connected, the S_i must be mostly disjoint, so each factor has variables in common with only a small number of other factors. Such models are widely used in e.g. spatial statistics where parameters corresponding to nearby locations are deemed to be related, and in time series where the state is typically assumed to be Markovian and each coordinate of x only depends on the previous and successive coordinates.

Continuous-time algorithms such as the *local bouncy particle sampler* (Peters et al. 2012, Bouchard-Côté et al. 2018) have been adapted to exploit such sparseness in the target distribution. In Chapter 2 we first discussed event times for factorized models, demonstrating that we can perform an iteration of the piecewise-deterministic Markov chain Monte Carlo algorithm by simulating an event time independently

for each factor. The first arrival among these is taken, the state is advanced by this time, and the event kernel corresponding to this is applied. The key observation here is that in sparse models the event kernels only affect the flow of neighbouring factors, and that event times for an unrelated factor remain valid. Thus, an event only requires recomputing event times for a local neighbourhood of factors.

5.1 The local bouncy particle sampler for sparsely connected models

Bouchard-Côté et al. (2018) describe a variant of the bouncy particle sampler called the *local bouncy particle sampler*. In this variant, the bouncy particle sampler is applied to the factorization (5.1), giving event rates

$$\lambda_i(x, v) = \langle \nabla U_i(x), v \rangle_+.$$

In keeping with the original bouncy particle sampler, deterministic bounces are used; the bounce corresponding to factor i yields a new state

$$v' = v - 2\frac{\langle \nabla U_i(x), v \rangle}{\|\nabla U_i(x)\|_2^2} \nabla U_i(x).$$

Critically, rates λ_i only depend on x_{S_i} and v_{S_i} since $\frac{\partial}{\partial x_j} U_i(x) = 0$ for any $j \notin S_i$. For this same reason, a bounce due to factor i only affects those factors which are connected to factor i, i.e. factors $\{j : S_j \cap S_i \neq \emptyset\}$. This is in contrast to a naïve application of the standard ("global") bouncy particle sampler wherein a single rate $\langle \nabla \sum U_i(x), v \rangle_+$ would be used, and computing this and the resulting velocity from each bounce would both involve all factors of the graph. While the bounce rate of the local algorithm is higher than that of the global algorithm (as $\sum_{i=1}^m \langle \nabla U_i(x_{S_i}), v \rangle_+ \geq \langle \nabla U(x), v \rangle_+$), Bouchard-Côté et al. (2018) observe that the reduced computational cost of these bounces can outweigh their increased frequency.

To take advantage of this sparsity, the algorithm must maintain an event time for each factor, and in each iteration must begin by simulating the flow for the *least* of these times. A naïve implementation which simply iterates through these times to find the minimum would thus cost $O(m)$ time, negating any advantage to

this scheme. For this reason, Bouchard-Côté et al. (2018) employ a *priority queue* data structure, which maintains the minimum among event times and, crucially, allows for times to be updated in $O(\log m)$ time.

We present the local bouncy particle sampler algorithm as Algorithm 33. For the purposes of our descriptions, we will use a priority queue data structure maintaining m tuples $\{(i, \tau_i)\}_{i \in \{1,\dots,m\}}$ corresponding to the next event time for each of the m factors; see Cormen et al. (2009) for a thorough presentation. We assume the priority queue Q supports the following operations:

- $\textsc{Min}(Q)$, which returns a tuple (i, τ_i) indicating the factor index and event time for the next event, i.e. τ_i is the minimum among $\{\tau_j\}_{j \in \{1,\dots,m\}}$. This operation takes $O(1)$ time.

- $\textsc{Update}(Q, i, \tau_i)$, which updates the time for factor i to τ_i taking $O(\log m)$ time.

- $\textsc{Advance}(Q, \tau)$, which effectively subtracts all times by τ, "advancing" time forward by an increment of τ. This final method is not strictly necessary but will yield a more convenient description of the algorithm where we do not have to keep track of an overall elapsed time.

We note a particular pathology with this scheme: if we take the coordinates of x to be independent and place each in its own factor, the algorithm still requires $O(\log m)$ time to simulate an event due to the need to simulate events in order.

One may be interested in extending the local bouncy particle sampler to more general flows, however it is not clear whether any alternative flows have the property that a local change in velocity will not affect the future flow for non-overlapping factors. For example, the Hamiltonian flow presented in Section 2.7 is not suitable: while a bounce may be local in the momentum p, the effect of changing any component of p has a global effect on the position x. This can be deduced from the flow equation $x_t = \cos(t)x + \sin(t)M^{-1}p$, noting that in most applications M is typically sparse and M^{-1} is a dense matrix.

Algorithm 33 Local continuous-time bouncy particle sampler.

Given initial state x, v, and number of iterations T. Outputs piecewise-deterministic segments (x, v, τ) where x and v are the initial state and τ the duration of the segment.

Initialize empty priority queue, Q.
for $i \in \{1, \ldots, m\}$ **do**
  Simulate event time τ_i for factor m with distribution

$$\mathbb{P}[\tau_i \geq t] = \int_0^t \langle \nabla U_i(x + vs), v \rangle_+ \mathrm{d}s. \tag{5.2}$$

  UPDATE(Q, i, τ_i).
end for
while $i \in \{1, \ldots, T\}$ **do**
  $i, \tau_i \leftarrow$ MIN(Q).
  Append piecewise-deterministic segment (x, v, τ_i) to output.
  ADVANCE(Q, τ_i).
  $x \leftarrow x + v\tau_i$.
  $v \leftarrow R(v; \nabla U_i(x))$.
  for $\{j : S_j \cap S_i \neq \emptyset\}$ **do**
    Simulate event time τ_i for factor i with distribution (5.2).
    UPDATE(Q, i, τ_i).
  end for
end while

5.2 Local analogues for discrete time

The discrete-time piecewise-deterministic Markov chain Monte Carlo algorithm as given in Algorithm 15 is not able to take advantage of the sparseness of a model: despite assigning each factor an independent rejection probability, the rejection of a single factor results in the rejection of the proposed move to $\Phi(z)$ for all variables.

To motivate our developments here, we examine the effect of an event occurring within a fixed time horizon from a given state z. We begin by computing all event times for each factor. Imagine that an event occurs within the fixed time window for just one factor. The effect of this event will change the velocities of the factor's neighbours. This will then require recomputation of event bounce times for those factors, some of which may now also experience an event within the fixed time window.

This behaviour within a time window suggests a mechanism for our discrete-time

analogues. First, we draw an analogy between bouncing within a time window and rejecting the proposal $\Phi(z)$ — this is intuitive as an event before reaching the end of the time window effectively rejects the eventual end-of-window state. The recomputation of event times is analogous to a factor rejecting the proposal $\Phi(z)$ having a knock-on effect on its neighbouring factors, where with some probability those factors now *also* reject the proposal $\Phi(z)$.

These rejections will only affect a subset of variables; we therefore propose a new algorithm which allows some of the variables to be updated in an Markov chain Monte Carlo move while others are rejected and remain unchanged, a so-called *partial update*, which we describe in the next section.

Our algorithm draws heavily on random cluster methods, first realized by Swendsen and Wang (1987). We will cast our problem in this framework by defining bonding conditions between connected coordinates which partition coordinates into clusters which are updated together. Our contribution is to show that the partial update may be cast in the framework of these random cluster algorithms, and to show that this yields an algorithm which can efficiently cluster variables only around those factors which are rejected.

As usual, we will henceforth assume a state space over $(x, v) \in \mathbb{R}^d \times \mathbb{R}^d$, a target density $\pi(x) = \prod_i \pi_i(x) = \exp(-\sum_i U_i(x))$, a normal distribution for the velocity $\psi(v)$, and a velocity-reversing involution $\mathcal{S}(x, v) = (x, -v)$. Furthermore, we shall assume a linear transformation $\Phi(x, v) = (x + v, v)$, though we propose an alternative in Section 5.8.

We will describe the algorithm as a composition of reversible kernels, as discussed in Section 1.2.8, which will enable us to more easily show correctness by establishing reversibility for each kernel. Recall that in this framework we make an update of x and v with some probability and then unconditionally apply the involution (in this case $\mathcal{S}(x, v) = (x, -v)$).

To further simplify exposition, we will work with a slice-augmented density, as discussed in Section 1.2.7. Again we find slices useful in deriving complex Markov chain Monte Carlo algorithms, as the kernel is now deterministic and verifying that

reversibility is satisfied is a matter of ensuring that the new state has non-zero density (is "on the slice"), the transformation is volume preserving, and applying the kernel twice yields the original state.

5.2.1 Illustrative example

We begin with a simple example which illustrates how clusters are formed. Take as our state $x, v \in \mathbb{R}^2 \times \mathbb{R}^2$, label these $\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}\right)$. Consider the factorization of the density on this state:

$$\pi(x_1, x_2) \propto f_1(x_1) f_{12}(x_1, x_2) f_2(x_2).$$

To simplify exposition, we will use a state augmented by slice variables for each factor. Denote the slice variables $u = (u_1, u_{12}, u_2)$ and

$$\eta(x, v, u) = \mathbb{I}[u_1 < f_1(x_1)]\mathbb{I}[u_{12} < f_{12}(x_{12})]\mathbb{I}[u_2 < f_2(x_2)]\psi(v).$$

We emphasize again the equivalence between multiple-slice constructions and factorized acceptance probabilities, as discussed in Section 3.5.4; the factorization of rates in the local bouncy particle sampler is analogous to the introduction of such slices.

A naïve algorithm proposing a move to $(x + v, -v, u)$ (u is not changed in this kernel) is accepted iff $\eta(x + v, -v, u) > 0$, otherwise we retain (x, v). This move is reversible and deterministic, as repeating the same proposal yields the state (x, v).

We instead consider any of the four possible states obtained by, for each coordinate, either applying the transform or leaving it unchanged. The four states are

$$\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}\right), \quad \left(\begin{bmatrix} x_1 + v_1 \\ x_2 \end{bmatrix}, \begin{bmatrix} -v_1 \\ v_2 \end{bmatrix}\right), \quad \left(\begin{bmatrix} x_1 \\ x_2 + v_2 \end{bmatrix}, \begin{bmatrix} v_1 \\ -v_2 \end{bmatrix}\right), \quad \text{and} \quad \left(\begin{bmatrix} x_1 + v_1 \\ x_2 + v_2 \end{bmatrix}, \begin{bmatrix} -v_1 \\ -v_2 \end{bmatrix}\right).$$

Note that if the following condition holds

$$u_{12} \leq \begin{cases} f_{12}(x_1, x_2) \\ f_{12}(x_1 + v_1, x_2) \\ f_{12}(x_1, x_2 + v_2) \\ f_{12}(x_1 + v_1, x_2 + v_2) \end{cases}$$

then any of the four updates will satisfy the slice constraint for f_{12} and we can treat the updates of x_1 and x_2 independently. Define $f_{12}^*(x, v)$ as the minimum of the four states; the condition becomes simply $u_{12} \leq f_{12}^*(x, v)$.

Define the transformation from the current state (x, v) to the next state (x', v') as:

- when $u_{12} < f^*_{12}(x, v)$, accept updates to x_1 and x_2 independently:

$$(x', v') \leftarrow \begin{cases} (x+v, -v) & u_1 < f_1(x_1+v_1) \text{ and } u_2 < f_2(x_2+v_2) \\ \left(\begin{bmatrix} x_1+v_1 \\ x_2 \end{bmatrix}, \begin{bmatrix} -v_1 \\ v_2 \end{bmatrix}\right) & u_1 < f_1(x_1+v_1) \text{ and } u_2 > f_2(x_2+v_2) \\ \left(\begin{bmatrix} x_1 \\ x_2+v_2 \end{bmatrix}, \begin{bmatrix} v_1 \\ -v_2 \end{bmatrix}\right) & u_1 > f_1(x_1+v_1) \text{ and } u_2 < f_2(x_2+v_2) \\ (x, v) & u_1 > f_1(x_1+v_1) \text{ and } u_2 > f_2(x_2+v_2) \end{cases}$$

- when $u_{12} > f^*_{12}(x, v)$, accept an update of both x_1 and x_2 only if no slices would be violated:

$$(x', v') \leftarrow \begin{cases} (x+v, -v) & u_1 < f_1(x_1+v_1) \text{ and } u_2 < f_2(x_2+v_2) \\ & \text{and } u_{12} < f_{12}(x_1+v_1, x_2+v_{@}) \\ (x, v) & u_1 > f_1(x_1+v_1) \text{ or } u_2 > f_2(x_2+v_2) \end{cases}$$

Note that this move is deterministic and reversible. Each resultant state leaves invariant the conditions which trigger it, and all are inverses of themselves. In other words, when applying the move again in the new state (x', v') the same conditions hold and the same specific modification of x and v is applied, yielding the original state. Just like in Swendsen-Wang, when the condition $u_{12} < f^*_{12}(x, v)$ holds, the coordinates are treated independently. Henceforth we call this the "independence condition," and say that the variables are "bonded" when the condition does not hold.

There are cases in the above where some update would be possible but we choose to remain in the original state. This is when one (or both) of the "mixed states" (x_1+v_1, x_2) and (x_1, x_2+v_2) are on the slice but when a full update (x_1+v_1, x_2+v_2) would not be, and so it may be possible to accept one of these mixed states. We argue that for larger models, considering such mixed states may yield complicated constraints for which determining the set of possible states becomes a problem in itself, and may also lead to situations requiring some sort of tie-breaking procedure to determine which state is accepted. We choose to instead err on the side of simplicity and let coordinates be either independent or bonded.

5.3 Partial updates

In order to extend the above example to more general graphs, we introduce additional notation and terminology. To describe the space of possible partial moves wherein

only some coordinates are modified we introduce a binary vector $b \in \{0,1\}^d$ where $b_i = 1$ indicates acceptance of the transformation Φ for coordinate i and $b_i = 0$ indicates the rejection. Thus we consider moves to states (x', v') where there exists some b such that

$$x' = x + b \odot v,$$
$$v' = (\mathbf{1} - 2b) \odot v,$$

where $\odot$ is the element-wise product and $\mathbf{1}$ is a vector of all ones. (The dimension of $\mathbf{1}$ will not be specified as it will be clear in context; define $\mathbf{0}$ similarly.) For fixed b the move is volume preserving. The velocity is reversed for those coordinates which are updated and thus applying the move twice with the same value of b would yield the original state. Our problem is now to devise an algorithm which selects the same b in both the current state (x, v) and resulting state (x', v'), yielding a reversible algorithm.

Henceforth assume here that the velocity distribution ψ can be factorized in the same way as π, that is, if π_i is a function of x_{S_i} then ψ_i is a function of v_{S_i} and $\psi(v) = \prod_{i=1}^m \psi(v_{S_i})$. Henceforth we will let $z_{S_i} = (x_{S_i}, v_{S_i})$ and $\rho_i(z_{S_i}) = \pi_i(x_{S_i})\psi_i(v_{s_i})$, though we will also use without ambiguity $\rho_i(z) = \rho_i(z_{S_i})$ when convenient. Define the partial transformation indexed by b as

$$\Phi_b(z) = (x + b \odot v, (\mathbf{1} - 2b) \odot v).$$

5.3.1 Slice variables and constraints

A slice variable u_i is introduced for each factor with constraint $\mathbb{I}[u_i < \rho_i(z_{S_i})]$, giving a full joint density

$$\eta(z, u) = \prod_{i=1}^m \mathbb{I}[u_i \leq \rho_i(z_{S_i})].$$

The slices provide constraints on which variables can be updated. Specifically, each factor imposes a restriction on b to a subset $B_i = \{b : u_i \leq \rho_i(\Phi_b(z))\}$. Due to the uniformity of the slice density, all $b \in B = \bigcap_i B_i$ yields $\eta(\Phi_b(z), u)$ with (unnormalized) density of 1 (i.e. "on the slice"), and $b \notin B$ yield zero density.

While we may entertain approaches where e.g. we try and optimize b to update as many variables as possible, computing these would involve finding solutions of a potentially complicated set of constraints. In this sense our setting differs from that of Swendsen-Wang. There, the only possible constraint for the Ising model is that the variables for a factor are the same, whereas in our case the lack of structure on π_i and ψ_i means that these factors may impose arbitrary constraints. To simplify our setting, we impose more restrictive constraints than required but which are sufficient to yield a valid solution for b which is tractable.

The constraints we use are as follows. There are three possible cases:

- **Unconstrained**. When no possible update can violate the slice constraint, that is when

  $$u_i \leq \min_{b_{S_i}} \rho_i(\Phi_{b_{S_i}}(z_{S_i})),$$

  the factor i imposes no constraint on b_{S_i}.

- **Bonded**. When there exists any "mixed" update that can violate the slice constraint, that is when

  $$u_i > \min_{b_{S_i} \in \{0,1\}^{|S_i|} \setminus \{\mathbf{0},\mathbf{1}\}} \rho_i(\Phi_{b_{S_i}}(z_{S_i})),$$

  we allow the factor i to impose that $b_{S_i} \in \{\mathbf{0}, \mathbf{1}\}$, that is, the elements of b_{S_i} are all the same. We say that these factors *bond* together the variables in S_i.

- **Rejected**. If accepting the update for all variables would violate the slice, that is when

  $$u_i > \rho_i(\Phi_{\mathbf{1}}(z_{S_i})),$$

  we allow the factor i to impose that $b_{S_i} = \mathbf{0}$. We say that these factors "reject" the forward move for variables in S_i.

The bonded factors determine a clustering (or partition) of variables: variables i and j must be in the same cluster if there exists a bonded factor k where $i, j \in S_k$. To describe the clusters, we let $\mathcal{V} = \{V_1, V_2, \ldots, V_\ell\}$ be the partition of variable indices associated with each of the ℓ clusters. For convenience, let

$\mathcal{F} = \{F_1, F_2, \ldots, F_\ell\}$ be the factor indices associated with each cluster, so $F_i = \{j : S_j \cap V_i \neq \emptyset\}$; note that $\mathcal{F}$ is not a partition as an unconstrained factor may depend on variables in separate clusters.

The Swendsen-Wang algorithm imposes similar constraints which are simpler than the three cases above as in the Ising model the rejected case never occurs. Nonetheless they share the similarity that the variables in a cluster must all be assigned the same value. Whereas in Swendsen-Wang the new cluster values are sampled independently of the other clusters, in our setting we would ideally accept the move wherever possible, assigning $b_i = 1$ for as many variables as possible.

Our specific selection of b is as follows:

$$b_i = \begin{cases} 0 & \text{if for the cluster } j \text{ containing } i \text{ (i.e. } i \in V_j\text{), some factor in } F_j \text{ is rejected,} \\ 1 & \text{otherwise.} \end{cases}$$

In words, we assign $b_i = 1$ for all clusters except those containing a rejected factor. It is clear that this is a solution satisfying the constraints above. Additionally, we see that this selection is reversible: any unconstrained factor remains unconstrained after any update, and any bonded factor i remains bonded after the update $b_{S_i} = \mathbf{1}$.

5.4 Summarizing constraints and lazy clustering

While the slice variables may offer an intuitive means of constructing the constraints imposed by each factor, we find it convenient to "marginalize out" these slice variables in favour of explicit rejection and bonding variables. The rejection variables allow us to draw a direct correspondence to discrete-time local piecewise-deterministic Markov chain Monte Carlo (Algorithm 17) where the first step is to determine which factors are rejected by the forward move. Also, this description will allow for a "lazy" implementation of the algorithm wherein the rejection variables are sampled first, and subsequently the bonding variables are sampled only as necessary.

Define the rejection variables $r \in \{0, 1\}^m$ where $r_i = 1$ represents a rejection for factor i and $r_i = 0$ the lack thereof; similarly for the bonding variables $s \in \{0, 1\}^m$ let $s_i = 1$ represent a bond while $s_i = 0$ represents no bond. It will be simpler to regard the rejection case $r_i = 1$ as also imposing a bond, so $r_i = 1 \implies s_i = 1$.

Using these new variables, we define an unnormalized joint density $\xi(z, r, s) = \prod_i \xi_i(z_{S_i}, r_i, s_i)$. The density for each our three constraint possibilities are as follows:

- **Unconstrained**. These factors are neither rejected ($r_i = 0$) nor bonded ($s_i = 0$).

$$\xi_i(z_{S_i}, r_i = 0, s_i = 0) = \min_{b_{S_i}} \rho_i(\Phi_{b_{S_i}}(z))$$

- **Bonded**. These are bonded ($s_i = 1$) but not rejected ($r_i = 0$).

$$\xi_i(z_{S_i}, r_i = 0, s_i = 1) = \min\left\{\rho_i(z_{S_i}), \rho_i(\Phi_{\mathbf{1}}(z_{S_i}))\right\} - \min_{b_{S_i}} \rho_i(\Phi_{b_{S_i}}(z_{S_i}))$$

- **Rejected**. These are rejected ($r_i = 1$) and we also consider these to be bonded ($s_i = 1$).

$$\xi_i(z_{S_i}, r_i = 1, s_i = 1) = \rho_i(z_{S_i}) - \min\left\{\rho_i(z_{S_i}), \rho_i(\Phi_{\mathbf{1}}(z_{S_i}))\right\}.$$

As mentioned, these can be understood as equivalent to the slice-based conditions given in Section 5.3.1 except where the slice variable $u_i \sim \text{Uniform}(0, \rho_i(z_{S_i}))$ has been integrated out.

Using these auxiliary variables, a lazy version of the algorithm is possible where we do not actually need to build the full set of clusters. Since $b_i = 0$ only for those clusters which contain a rejecting factor, we can first find those factors which are rejected and then build clusters only around those factors. The unclustered variables then accept the forward move, while the clustered variables remain unchanged. Such an approach will avoid the computational overhead of testing the bonding condition for all factors. It may even be possible that in some cases we can leverage the efficient thinning techniques of Section 4.4.2 to select the rejected factors.

Algorithm 34 describes the clustering scheme applied to a random-walk proposal. We provide for completeness a subroutine Algorithm 35, which describes a depth-first search approach to construct the clusters. BuildClusters() returns a collection of tuples, where each tuple (B, P) describes the set of bonded factors B and the set of unbonded factors P defining the periphery of the cluster. Algorithm 35

only serves as an example of how this function may be implemented and we do not place importance on the particulars of this implementation; we do emphasize that it only needs to construct the clusters including the rejected factors and that the z_i can be sampled only as needed.

Algorithm 34 Clustering with a random-walk proposal.

function ClusterRandomWalk(x)
  Sample $v \sim \psi(\cdot)$. Let $z = (x, v)$.
  for $i \in [m]$ **do**
    Sample $r_i \sim \text{Bernoulli}\left(1 - \min\left\{1, \frac{\rho_i(\Phi_{S_i}(z_{S_i}))}{\rho_i(z_{S_i})}\right\}\right)$.
  end for
  $\mathcal{C} \leftarrow$ BuildClusters(x, v, r).
  $W \leftarrow [d] \setminus \bigcup_{(B,P)\in\mathcal{C}} \bigcup_{i\in B} S_i$.
  $x'_W \leftarrow x_W + v_W$.
  return x'.
end function

5.5 Updates conditional on a clustering

The algorithms above have been modelled after the factorization used in the local bouncy particle sampler in the sense that each factor is accepted or rejected independently of the others. While this may allow for some efficiency in that we can perform lazy clustering as in Algorithm 34, we will find that by imputing a clustering first we arrive at a simpler algorithm.

Here we use only the bonding variables s, which are binary variables as before. The unnormalized joint density is

$$\eta(z, s) = \prod_{i=1}^{m} \eta_i(z_{S_i}, s_i),$$
$$\eta_i(z_{S_i}, s_i) = \left\{\rho_i(z_{S_i}) - \min_{b_{S_i}} \rho_i(\Phi_{b_{S_i}}(z_{S_i}))\right\} \mathbb{I}[s_i = 1] + \min_{b_{S_i}} \rho_i(\Phi_{b_{S_i}}(z_{S_i}))\mathbb{I}[s_i = 0].$$

(We note a resemblance to the joint density used in the FlyMC algorithm (4.18).) As before $\eta_i(z_{S_i}, s_i = 0)$ is invariant to any partial update. The conditional densities of the s_i are

$$\eta(s_i = 0|z_{S_i}) = \frac{\min_{b_{S_i}} \rho_i(\Phi_{b_{S_i}}(z_{S_i}))}{\rho_i(z_{S_i})}. \tag{5.3}$$

Algorithm 35 Implementation of *BuildClusters*

function BUILDCLUSTERS(x, v, r)
  Initialize collection of clusters $\mathcal{C} \leftarrow \emptyset$.
  Initialize s with $s_i = 1$ for all i where $r_i = 1$. Remaining indices of s are uninitialized.
  for $i : r_i = 1$ **do**
    if $i \notin \cup_{(B,P)\in\mathcal{C}} B$ **then**
      $B, P, s \leftarrow$ BUILDCLUSTERS-DFS$(x, v, r, \emptyset, \emptyset, s, i)$.
      $\mathcal{C} \leftarrow \mathcal{C} \cup \{(B, P)\}$.
    end if
  end for
  return $\mathcal{C}$.
end function
function BUILDCLUSTERS-DFS(x, v, r, B, P, s, i)
  if $i \in B \cup P$ **then**
    return B, P, s.
  end if
  if s_i not initialized **then**
    Sample $s_i \sim \xi(s_i | x_{S_i}, v_{S_i}, r_i)$.
  end if
  if $s_i = 1$ **then**
    $B \leftarrow B \cup \{i\}$.
    for $j : S_j \cap S_i \neq \emptyset$ **do**
      $B, P, s \leftarrow$ BUILDCLUSTERS-DFS(x, v, r, B, P, s, j).
    end for
  else
    $P \leftarrow P \cup \{i\}$.
  end if
  return B, P, s.
end function

Henceforth we will consider updates of z *conditional* on s. This provides a few advantages:

- we can now update each cluster in turn, treating the series of updates as a deterministic ordering of reversible kernels;
- the acceptance probabilities of these updates will in general be higher than if we had allowed each factor to reject independently (though we find this does not make a significant difference in practice); and
- deriving correct bounces is simpler.

In these algorithms we use a disjoint set data structure (sometimes called Union-Find). This data structure supports the UNION() operation, which agglomerates the clusters of the provided indices. Denote the set of clusters $\mathcal{C}$, where each cluster $C \in \mathcal{C}$ is a set of coordinate indices, and $\mathcal{C}$ is a partition of $\{1, \ldots, d\}$. We assume this data structure supports the following operations:

- INITCLUSTERS($\mathbb{C}, d$), which initializes the instance $\mathbb{C}$, representing a set of m clusters each containing a single coordinate index from $\{1, \ldots, d\}$. So each coordinate is initially assigned its own cluster $\mathcal{C} = \{\{1\}, \{2\}, \{3\}, \ldots, \{d\}\}$.

- UNION($\mathbb{C}, I$), which combines those clusters containing any of the coordinate indices I. So if $i, j \in I$, then after this operation $i, j \in C$ for some $C \in \mathcal{C}$.

- CLUSTERS($\mathbb{C}$) returns the set of clusters $\mathcal{C}$.

An efficient implementation of this data structure gives an extremely low time complexity for the union operation of $O(\alpha(n))$ where $\alpha(n)$ is the Ackermann function (Cormen et al. 2009); for our purposes this is effectively constant. The representation $\mathcal{C}$ is not used internally by the data structure but can be formed from the internal representation in $O(m)$ time.

The algorithm can be divided into two separate stages: sampling the clustering variables s given the state z, and then sampling the z given the s. To sample z we iterate through the clusters of variables and for each cluster perform an accept-reject step of the transformation Φ. We establish invariance by noting that this iteration through the clusters can be viewed as a succession of reversible kernels. While a fixed ordering of these kernels does not yield a reversible kernel, invariance to the joint density η is maintained; in any case the algorithm could be made reversible by applying each kernel in a random order.

To derive the Metropolis-Hastings algorithm for the update of a single cluster, we can begin by computing the Metropolis-Hastings ratio for the entire joint density.

Algorithm 36 Clustering with a random-walk proposal, building clusters first

Given initial state $x \in \mathbb{R}^d$, return new state x'.

Sample $v \sim \psi(\cdot)$, let $z \leftarrow (x, v)$.
$\mathcal{C} \leftarrow$ InitClusters(d).
for $i \in [m]$ where $|S_i| \geq 2$ **do**
  Sample

$$s_i \sim \text{Bernoulli}\left(1 - \frac{\min_{b_{S_i}} \rho_i(\Phi_{b_{S_i}}(z_{S_i}))}{\rho_i(z_{S_i})}\right).$$

  if $s_i = 1$ **then**
    Union$(\mathcal{C}, S_i)$
  end if
end for
for $C \in \mathcal{C}$ **do**
  Compute proposal $z^* \leftarrow \Phi_{\mathbf{1}_C}(z)$ where $\mathbf{1}_C$ has elements $(\mathbf{1}_C)_j = \mathbb{I}[j \in C]$.
  Compute relevant factors $F \leftarrow \{i : S_i \cap C \neq \emptyset, s_i = 1\}$.
  with probability

$$\min\left\{1, \prod_{i \in F} \frac{\eta_i(z^*, s_i)}{\eta_i(z, s_i)}\right\}$$

    $z'_C \leftarrow z^*_C$.
  otherwise
    $z'_C \leftarrow z_C$.
end for
return x' where $z' = (x', v')$.

Take for the proposed state $z^* = \Phi_{\mathbf{1}_C}(z)$ by defining $\mathbf{1}_C$ to be a vector with elements $(\mathbf{1}_C)_j = \mathbb{I}[j \in C]$. The Metropolis-Hastings ratio is

$$\min\left\{1, \frac{\eta(\Phi_{\mathbf{1}_C}(z), s)}{\eta(z, s)}\right\} = \min\left\{1, \prod_{i : S_i \cup C \neq \emptyset} \frac{\eta_i(z^*, s_i)}{\eta(z, s_i)}\right\}.$$

Noting that z is unchanged for any factor j where $S_j \cap C = \emptyset$, those factors can be removed from the product. Additionally, note that for any factor j where $s_j = 0$, we have

$$\frac{\eta_i(\Phi_b(z_{S_i}), s_i = 0)}{\eta_i(z_{S_i}, s_i = 0)} = \frac{\min_{b_{S_i}} \rho_i(\Phi_{b_{S_i}}(z_{S_i}))}{\min_{b_{S_i}} \rho_i(\Phi_{b_{S_i}}(z_{S_i}))} = 1.$$

This latter statement is true as no matter what updates have been performed to neighbouring variables, as the value of $\min_{b_{S_i}} \rho_i(\Phi_{b_{S_i}}(z))$ is not changed by accepting any update of the form Φ_b.

The complete algorithm is given as Algorithm 36.

5.5.1 Prior work

The algorithm of this section allows us to connect our algorithm to the work of Cai (2009), which we believe is the work most similar to the algorithms presented in this chapter. Like us, they develop a clustering method which builds clusters conditional on the current state and the proposal, allowing partial updates for real-valued state spaces. They consider models with a potential of the form

$$U(x) = \sum_{ij} U_{ij}(|x_i - x_j|) + \sum_i V_i(x_i)$$

which, critically, assumes that U_{ij} is a function only of the absolute difference between x_i and x_j. Furthermore, proposals are generated by first sampling a *reflection point* $r \in \mathbb{R}$ and then computing the proposal

$$x' = r\mathbf{1} - x.$$

For any r, the potential is unchanged by the proposal as $U_{ij}(|[r - x_i] - [r - x_j]|) = U_{ij}(|x_i - x_j|)$. Reflecting one of the variables gives the same potential regardless of which variable is reflected $U_{ij}(|[r - x_i] - x_j|) = U_{ij}(|x_i - [r - x_j]|)$. So, much like for the original Swendsen-Wang algorithm, a partial update can only yield two possible energy levels. Their bonding probability

$$\begin{aligned}\eta(s_i = 0|x, r) &= \exp\left(-\left[U_{ij}(|x_i - x_j|) - U_{ij}(|(r - x_i) - x_j|)\right]_+\right) \\ &= \frac{\min\left\{\exp(-U_{ij}(|x_i - x_j|), \exp(-U_{ij}(|(r - x_i) - x_j)\right\}}{\exp(-U_{ij}(x_i, x_j))}\end{aligned}$$

can be seen as a special case of (5.3) due to the existence of only two energy levels. Thus their algorithm can be seen as an instance of ours where both the form of the interaction potentials and the proposal are restricted.

5.6 Bounces for clusters

The algorithms listed so far describe a valid Markov chain Monte Carlo kernel which could be used effectively in a context where v is refreshed after every iteration. However in a piecewise-deterministic context where v is typically updated after

multiple iterations, we wish to implement bounce events which occur whenever the proposal $\Phi(z)$ is rejected, mimicking the behaviour of the continuous-time algorithm.

A possible difficulty in deriving bounces is that they may complicate the constraints of the previous section: modifying v also modifies the cluster, and depending how we compute the bounce action this may in turn affect how we modify v. For this reason we find that the lazy clustering methods of Section 5.4 are not easily adapted to include bounce proposals as modifications to the velocity may affect the clustering. The approach we take to circumvent these complications is to keep fixed the bonding variables s. Conditional on s the clustering is fixed and unaffected by any modifications to x and v.

Conditional on s, the implementation of the bounce for one cluster mimics closely that of the standard bouncy particle sampler. We take the gradient contributions from the factors which are internal to the cluster, i.e. for the update to the cluster C, we take

$$g = \sum_{i:S_i \subseteq C} \nabla U_i(x)$$

and then propose the new velocity while also including the involution:

$$v^* = -R(v; g) = -\left(v - 2\frac{\langle v, g\rangle}{\|g\|_2^2} g\right).$$

As usual, the acceptance probability for the bounce takes the form, letting $z^* = (x, v^*)$,

$$\begin{aligned}
\beta'_C(v, v^*) &= \min\left\{1, \frac{\eta(z^*, s)\left(1 - \min\left\{1, \frac{\eta(\Phi_{\mathbf{1}_C}(z^*), s)}{\eta(z^*, s)}\right\}\right)}{\eta(z, s)\left(1 - \min\left\{1, \frac{\eta(\Phi_{\mathbf{1}_C}(z), s)}{\eta(z, s)}\right\}\right)}\right\} \\
&= \min\left\{1, \frac{[\eta(z^*, s) - \eta(\Phi_{\mathbf{1}_C}(z^*), s)]_+}{[\eta(z, s) - \eta(\Phi_{\mathbf{1}_C}(z), s)]_+}\right\} \\
&= \min\left\{1, \frac{\left[\prod_{i:S_i \cap C \neq \emptyset} \eta_i(z^*, s_i) - \prod_{i:S_i \cap C \neq \emptyset} \eta_i(\Phi_{\mathbf{1}_C}(z^*), s_i)\right]_+}{\left[\prod_{i:S_i \cap C \neq \emptyset} \eta_i(z, s) - \prod_{i:S_i \cap C \neq \emptyset} \eta(\Phi_{\mathbf{1}_C}(z), s)\right]_+}\right\}.
\end{aligned}$$

Here, we cannot omit the computation of those factors for which $s_i = 0$, as $\min_{b_{S_i}} \rho_i(\Phi_{b_{S_i}}(z))$ is not invariant to changes of the velocity. We do need to include

those factors for which any coordinate of v is modified, these are $\{i : S_i \cap C \neq \emptyset\}$ as the gradient is guaranteed to only modify coordinates in C.

See Algorithm 37 for the complete algorithm.

We note that the choice of g given here is perhaps not unique. It is not necessarily clear that we want to use gradient contributions from all factors with coordinates contained in C; while such bounces may be justified in the standard setting, there may be better options when targeting the conditional on s. We leave it to future work to more carefully establish this.

5.7 Autoregressive proposals for the cluster random walk algorithm

Thus far we have only considered velocities sampled independently of x, which in the Metropolis-Hastings is equivalent to a normal random walk proposal. Here we consider a autoregressive proposals, a popular class of proposals widely used in Markov chain Monte Carlo algorithms (Tierney 1994). These are proposals where, given the current state x, we compute a proposal

$$x^* = \mu + \rho(x - \mu) + \sqrt{1 - \rho^2}\xi \qquad \xi \sim \mathcal{N}(0, \Sigma).$$

If x is distributed according to the distribution $\mathcal{N}(\mu, \Sigma)$, then x^* will be marginally distributed according to this same distribution. ρ is a tuning parameter in the range $(-1, 1)$ which controls the correlation between x and x^*.

We explore autoregressive proposals in two main cases:

1. When the model has some prior distribution which is normal, then we can use autoregressive proposals to target that prior distribution which may give some computational benefit. This is the approach taken in e.g. elliptical slice sampling (Murray et al. 2010).

2. When we have some approximation of the target distribution where this approximation is a normal distribution, then we can use autoregressive proposals which target that approximation. We used these sorts of autoregressive

Algorithm 37 ClusterBPS, with bounces for rejected clusters

Given state $z = (x, v) \in \mathbb{R}^d \times \mathbb{R}^d$, return new state z'.

function ClusterBPS(x, v)
  $\mathcal{C} \leftarrow$ InitClusters(d).
  for $i \in [m]$ where $|S_i| \geq 2$ **do**
    Sample

$$s_i \sim \text{Bernoulli}\left(1 - \frac{\min_{b_{S_i}} \rho_i(\Phi_{b_{S_i}}(z_{S_i}))}{\rho_i(z_{S_i})}\right).$$

    if $s_i = 1$ **then**
      Union($\mathcal{C}$, S_i)
    end if
  end for
  Initialize next state $z' = (x', v') \leftarrow (x, v)$.
  for $C \in$ Clusters($\mathcal{C}$) **do**
    with probability

$$\min\left\{1, \prod_{i:S_i \cap C \neq \emptyset, s_i = 1} \frac{\eta_i(\Phi_{\mathbf{1}_{S_i}}(z_{S_i})), s_i)}{\eta_i(z_{S_i}, s_i)}\right\}$$

      $z'_C \leftarrow \Phi_{\mathbf{1}_C}(z)$.
    otherwise
      Compute $g \leftarrow \sum_{i:S_i \subseteq C, s_i = 1} \nabla U_i(x_{S_i})$.
      Compute $v^* \leftarrow -R(v; g)$, and take $z^* = (x, v^*)$.
      with probability

$$\min\left\{1, \frac{\left[\prod_{i:S_i \cap C \neq \emptyset} \eta_i(z^*, s_i) - \prod_{i:S_i \cap C \neq \emptyset} \eta_i(\Phi_{\mathbf{1}_{S_i}}(z^*), s_i)\right]_+}{\left[\prod_{i:S_i \cap C \neq \emptyset} \eta_i(z_{S_i}, s_i) - \prod_{i:S_i \cap C \neq \emptyset} \eta_i(\Phi_{\mathbf{1}_{S_i}}(z), s_i)\right]_+}\right\}$$

        Set $v'_C \leftarrow v^*_C$.
  end for
  return x', v'.
end function

proposals in Chapter 4 to target a normal approximation to a large dataset posterior.

We shall focus primarily on the first case. For this case we will consider potentials of the form

$$U(x) = \frac{1}{2} x^T \Lambda x + \sum_i V_i(x_i) \tag{5.4}$$

where Λ is usually a precision matrix (but not necessarily) which is sparse: non-zero

entries indicate a relationship between coordinates, i.e. $\Lambda_{ij} \neq 0$ implies that x_i and x_j are dependent, conditional on all other indices $\{1, \ldots, d\} \setminus \{i, j\}$. We will make efforts to use sparse operations whenever possible. This forbids, for example, directly using the covariance matrix $\Sigma = \Lambda^{-1}$ as this will generally not be sparse even if Λ is. For further reference on such models see Rue and Held (2005).

5.7.1 A velocity representation of autoregressive proposals

In order to use autoregressive proposals in our cluster random walk algorithm, we derive an equivalent "velocity" representation of an autoregressive proposal which will allow us to express these proposals in our usual terms. It will likely be straightforward to re-derive the entire algorithm in terms of x and x^* using proposal densities directly over these states, but we leave this to future work.

As before, we take for an autoregressive proposal targeting $\mathcal{N}(\mu, \Sigma)$,

$$x^* = \rho(x - \mu) + \sqrt{1 - \rho^2}\xi + \mu \qquad\qquad \xi \sim N(0, \Sigma).$$

We want to have this proposal as the result of taking a step of a velocity v, i.e. $x^* = x + v$, so solving for v we have

$$v = x^* - x = (\rho - 1)(x - \mu) + \sqrt{1 - \rho^2}\xi$$

implying that we can equivalently sample

$$v \sim \mathcal{N}((\rho - 1)(x - \mu), (1 - \rho^2)\Sigma).$$

The corresponding potential for this velocity is

$$\begin{aligned} K(v) &= \frac{1}{2} \left[v - (\rho - 1)(x - \mu)\right]^T \left[(1 - \rho^2)\Sigma\right]^{-1} \left[v - (\rho - 1)(x - \mu)\right] \\ &= \frac{1}{2} \frac{1}{1 - \rho^2} \sum_{ij} \Lambda_{ij} \left[v_i - (\rho - 1)(x_i - \mu_i)\right] \left[v_j - (\rho - 1)(x_j - \mu_j)\right] \end{aligned} \tag{5.5}$$

where $\Lambda = \Sigma^{-1}$.

5.7.2 Rejections in the factorized autoregressive move

One advantage of using autoregressive moves is that this proposal "cancels" with the target (or the Gaussian factors of the target) when it targets the same Gaussian distribution as the target. As such, it incurs no rejections of the proposal (or no rejections due to the Gaussian factors of the target). Here we show that this is also true when using factorized acceptance probabilities, so long as we pair together corresponding potential and kinetic terms into the same factors.

Assume the target distribution has a potential $U(x) = \frac{1}{2}x^T\Lambda x$, and assuming $K(v)$ of the form (5.5) with $\mu = 0$; the velocity is thus adapted to the target in the sense that $x^* = x + v$ will be distributed according to $\mathcal{N}(0, \Lambda^{-1})$ when x is. The total energy $H(x, v)$ is

$$U(x) + K(v) = \left(\frac{1}{2} + \frac{1}{2}\frac{(\rho-1)^2}{1-\rho^2}\right)x^T\Lambda x + \frac{1}{2}\frac{1}{1-\rho^2}v^T\Lambda v - \frac{\rho-1}{1-\rho^2}v^T\Lambda x \tag{5.6}$$

and applying the linear update to the state $(x + v, -v)$ gives a total energy

$$\begin{aligned} U(x+v) + K(-v) &= \frac{1}{2}(x+v)^T\Lambda(x+v) \\ &\quad + \frac{1}{2}(-v-(\rho-1)(x+v))^T\frac{1}{1-\rho^2}\Lambda(-v-(\rho-1)(x+v)) \\ &= U(x) + K(v) \end{aligned}$$

which is equal to (5.6) so the energies are the same before and after the move, and the autoregressive proposal would be accepted with probability 1.

Note that the above is true for any matrix Λ; this implies that any factorization which pairs corresponding x and v factors will accept autoregressive moves with probability 1. In general, we can say that for some partition of pairs of indices $\bigcup_k F_k = \{1, \ldots, d\} \times \{1, \ldots, d\}$, $F_i \cap F_j = \emptyset$ for $i \neq j$, then if the factors are defined such that

$$H(x, v) = \sum_k H_k(x, v) = \sum_k \sum_{ij \in F_k} \Lambda_{ij}\left[x_i x_j + \frac{1}{1-\rho^2}(v_i - (\rho-1)x_i)(v_j - (\rho-1)x_j)\right]$$

we will have no rejections. We will typically take factors corresponding to each possible combination of variables, so for a d-dimensional state will be decomposed

to $d + \frac{1}{2}(d^2 - d)$ factors as follows:

$$H(x,v) = \sum_{i=1}^{d} \frac{1}{2}\Lambda_{ii}\left[x_i^2 + \frac{1}{1-\rho^2}(v_i - (\rho-1)x_i)^2\right] \tag{5.7}$$

$$+ \sum_{i=1}^{d}\sum_{j=i+1}^{d} \Lambda_{ij}\left[x_i x_j + \frac{1}{1-\rho^2}(v_i - (\rho-1)x_i)(v_j - (\rho-1)x_j)\right]. \tag{5.8}$$

We derive the probability of forming a bond in this model. For a two-variable Gaussian factor, we have the density

$$\rho_{ij}(x,v) = \exp\left(-\Lambda_{ij}\left[x_i x_j + \frac{1}{1-\rho^2}(v_i - (\rho-1)x_i)(v_j - (\rho-1)x_j)\right]\right).$$

A bond will be formed with probability

$$\eta(s_{ij} = 1|z) = 1 - \frac{\min_{b\in\{0,1\}^2}\rho_{ij}(\Phi_b(x,v))}{\rho_{ij}(x,v)} = 1 - \exp\left(-\left[\frac{\Lambda_{ij}}{\rho-1}v_1 v_2\right]_+\right)$$

which can be evaluated efficiently.

Using autoregressive proposals, algorithm 36 takes on an interesting interpretation if we have a potential of the form (5.4). In this case, the clustering step will only involve the factors of the latent field and will not consider the observation terms, since the observations do not involve multiple coordinates. The subsequent accept-reject step will however only consider observation terms, since any update of coordinates leaves invariant the energy of the Gaussian factors.

5.7.3 Per-coordinate autoregression

Using autoregressive moves which target the latent space is quite appealing as computing rejections and clusters in this case can be highly efficient. However, such a model may still struggle in poorly conditioned problems where some coordinate is highly constrained by the observations, requiring that the autoregressive coefficient ρ be near to one, while other coordinates may be less constrained and would be better explored by a smaller autoregressive coefficient.

As such, we aim to derive a per-coordinate autoregression coefficient. That is, we will take as our proposal

$$x^* = P(x - \mu) + \xi \tag{5.9}$$

where P is a matrix and ξ is Gaussian noise with an appropriate covariance matrix, chosen so that x^* is distributed according to some Gaussian distribution if x is as well. Conceptually we might expect that P should be a diagonal matrix with each element interpreted as an autoregressive coefficient for each coordinate. However, we note that P cannot be set arbitrarily; if strong correlations exist between the coordinates of x then these must not be broken. As an example, imagine for a two-dimensional model that we took a matrix $P = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$; any correlation between x_1 and x_2 would not be recoverable using the form (5.9).

Our approach will be to design a P which first takes a projection of x into a space where the coordinates are uncorrelated, then applies a per-coordinate autoregressive parameter, and finally transforms back into the original space. In the space where the coordinates are uncorrelated we may indeed choose arbitrary per-coordinate autoregressive coefficients. Thus we restrict the user to select autocorrelation coefficients in a different space, but we suggest that as this space is unconstrained it will be easy to select this to fulfil some objective in the constrained space as closely as possible.

One method to project x into an uncorrelated space would be to use the eigenvectors of the covariance matrix, however this will be inappropriate in situations where we have a sparse Λ and wish to use sparse operations wherever possible. As such, we suggest that a more appropriate choice is to use projections based on an LDL^T decomposition of the precision matrix which allows us to project to and from the uncorrelated space using only sparse operations.

Henceforth assume that x is zero-mean. Instead of a per-coordinate autoregressive coefficient, take an autoregressive *matrix* P, so our autoregressive updates take the form

$$x^* = Px + Qu$$

where Q is a lower triangular matrix with non-negative diagonal entries (so QQ^T is semi-positive definite) and u is uncorrelated Gaussian noise with unit variance.

If $x \sim \mathcal{N}(0, \Sigma)$, then Px has covariance matrix $P\Sigma P^T$, and for x^* to have covariance matrix Σ we require that

$$QQ^T = \Sigma - P\Sigma P^T.$$

Take $\Lambda = LDL^T$ where L is lower triangular and D is a diagonal matrix. Thus $\Sigma = \Lambda^{-1} = L^{-T}D^{-1}L^{-1}$. We assume that Λ and L are sparse; Λ is often sparse as defined by the model and L will be typically be sparse, but dependent on the *bandwidth* of Λ, see Rue and Held (2005) for details.

Assume a P of the form

$$P = L^{-T}\rho L^T$$

where ρ is a diagonal matrix of arbitrarily selected autoregression coefficients with $\rho_i \in (-1, 1)$. Given this choice of P, QQ^T must now satisfy

$$\begin{aligned} QQ^T &= L^{-T}D^{-1}L^{-1} - L^{-T}\rho L^T L^{-T}D^{-1}L^{-1}L\rho L^{-1} \\ &= L^{-T}D^{-1/2}\left[I - \rho^2\right]D^{-1/2}L^{-1}. \end{aligned}$$

Thus it is sufficient that

$$Q = L^{-T}D^{-1/2}\sqrt{I - \rho^2}$$

and QQ^T is positive definite so long as $1 - \rho_i^2 > 0$ for all i.

It remains to show that we can efficiently compute x^* by relying only on sparse operations. We assume that L is sparse; this means we can efficiently compute expressions of the form $x = Ly$ by direct multiplication, and expressions of the form $p = L^{-T}q$ by solving $L^T p = q$ for p.

So, to compute $Px = L^{-T}\rho L^T x$, we take

1. $x' = \rho L^T x$ which can be done efficiently as ρ is diagonal and L^T is sparse, then
2. $Px = L^{-T}x'$, obtained by solving $L^T(Px) = x'$ for Px.

Similarly, to compute $Qu = L^{-T}D^{-1/2}\sqrt{I - \rho^2}u$ we take

1. $z = D^{-1/2}\sqrt{I - \rho^2}u$ by direct multiplication. This is efficient as $D^{-1/2}\sqrt{I - \rho^2}$ is diagonal.

2. $Qu = L^{-T}z$, obtained by solving $L^T Qu = z$ for Qu.

5.8 Partial updates for the leapfrog integrator

This far we have considered partial updates only for a linear transformation which is the simplest possible transformation. Here we show that partial updates are also possible for a single iteration of the leapfrog algorithm as used in Hamiltonian Monte Carlo.

For some $b \in \{0, 1\}^d$, which here $b_i = 1$ also indicates that a coordinate is updated, the partial leapfrog transformation from state x_t, p_t is

$$\begin{aligned} x_{t+1/2} &= x_t + \frac{\epsilon}{2} p_t \\ p_{t+1} &= (\mathbf{1} - b) \odot p_t - b \odot (p_t - \epsilon \nabla U(x_{t+1/2})) \\ x_{t+1} &= x_{t+1/2} - \frac{\epsilon}{2} p_{t+1}. \end{aligned}$$

For $b = \mathbf{1}$ this reduces to the typical leapfrog scheme (with subsequent momentum reversal) and for $b = \mathbf{0}$ the state is unchanged and $(x_{t+1}, p_{t+1}) = (x_t, p_t)$.

The partial leapfrog transformation is the inverse of itself (for a given b). Also, since the three steps of the leapfrog are volume preserving, the entire transformation is volume preserving.

The partial leapfrog can be used as the transformation Φ in any of the clustering algorithms described above. We suggest that it can work well when used in a piecewise-deterministic fashion, where the partial update is followed in each iteration by reversing the momentum, and the momentum is only refreshed periodically. It is not immediately obvious whether it is feasible to extend partial updates to a multiple-step leapfrog integrator.

For the linear transformation, a rejection of some coordinates would cause those coordinates to move back towards previously visited states. This motivated the use of an event kernel, such as the bounce or tunnelling procedures of Chapter

3, allowing the possibility that those coordinates will move towards states not visited in the preceding iterations. This is however *not* the case for the leapfrog transformation, since the update to the momentum relies on the potential gradient for *all* coordinates. As such, even with a reversal of the momentum for a subset of coordinates, the momentum updates will be novel and the trajectory will not immediately revisit states.

It may be interesting to connect this method to its continuous-time counterpart, which would be a modification of the local bouncy particle sampler to use Hamiltonian dynamics. We have not previously considered Hamiltonian dynamics for the local bouncy particle sampler, as the interdependence of coordinates in the Hamiltonian flow means that events cannot have a local effect.

5.9 Experimental results

We give preliminary simulation results for some of the algorithms discussed in this chapter, and compare these to a standard Hamiltonian Monte Carlo sampler.

We use a "Gaussian-Poisson" model comprising a two-dimensional latent Gaussian field with Poisson observations at each site. The sites are arranged in a $d \times d$ grid, giving d^2 sites, and each site as its four neighbours the sites to its left, top, right, and bottom. More specifically, the site at coordinates (m, n) is neighbour to sites $(m-1, n)$, $(m, n-1)$, $(m+1, n)$, and $(m, n+1)$ if they exist (note that henceforth we number sites by a single index in $\{1, \ldots, d^2\}$). The density of the model is

$$\pi(x) \propto \mathcal{N}(x; 0, \Lambda^{-1}) \prod_{i=1}^{d^2} \text{Poisson}(y_i; \exp(x_i))$$

and Λ has diagonal elements $\Lambda_{ii} = 1$ and off-diagonal elements either $\Lambda_{ij} = 0$ if sites i and j are not neighbours or $\Lambda_{ij} = -0.25$ if the sites are neighbours. The observations were generated from the model, i.e. by first sampling x from $\mathcal{N}(0, \Lambda^{-1})$ and then taking $y_i \sim \text{Poisson}(\exp(x_i))$.

For each algorithm we have plotted the integrated autocorrelation time (IACT) as described in Section 4.9 of the second moment of x_1, the first coordinate of the latent

field, computed using overlapping batch means. The integrated autocorrelation time is given in terms of iterations and also in terms of seconds.

All algorithms were run five times starting from the mode of the distribution. We plot the average integrated autocorrelation time across these five runs and give error estimates based on these five replications. Each run was for 2×10^4 iterations. For the local bouncy particle sampler we simulated 2×10^6 events, and for the purpose of computing the integrated autocorrelation time we subsampled the path to 2×10^4 discrete-time samples. The overlapping batch means was used with a batch length of $(2 \times 10^4)^{2/3}$. The mean and variance of each posterior were estimated from a Hamiltonian Monte Carlo run of 10^5 samples using the best-performing parameter combination among those tested.

In each plot, we show one grid size per pane, testing all models for $d \times d$ grids with $d \in \{4, 8, 16, 32, 64, 128\}$. We ran the following algorithms:

- Hamiltonian Monte Carlo, Figure 5.1. We tested for a range of leapfrog step sizes and numbers of leapfrog steps.

- The continuous-time local bouncy particle sampler (Algorithm 33), Figure 5.2. The integrated autocorrelation time was computed by first subsampling the trajectory at regular intervals at an average rate of 10 samples per piecewise-deterministic segment. Thus the integrated autocorrelation time in iterations is not comparable with the others, but the integrated autocorrelation time expressed terms of seconds can be compared with that of the other algorithms.

- The cluster random walk algorithm (Algorithm 36) with proposals drawn from a Gaussian distribution with a scaled identity precision matrix, Figure 5.3.

- The cluster random walk algorithm (Algorithm 36) using autoregressive proposals, Figure 5.4.

For all algorithms we compared performance across design parameters. We found that Hamiltonian Monte Carlo was most sensitive to tuning parameters. For both versions of cluster random walk, the parameter tuning was relatively insensitive

to the dimension, and parameters that worked well in low dimension also worked well in higher dimension. The local bouncy particle sampler algorithm seemed to work equally well regardless of the refresh rate chosen.

Ultimately, we found that the clustering algorithms gave similar performance to Hamiltonian Monte Carlo in terms of integrated autocorrelation time in seconds. Unfortunately, this quantity is highly dependent on implementation details and it may be the case that a suboptimal implementation of these algorithms may unfairly penalize their results. Our Hamiltonian Monte Carlo implementation is built on the `Eigen` matrix library for C++ (Guennebaud et al. 2010), which we expect to give highly optimized performance for calculating gradients and potential values in Hamiltonian Monte Carlo.

It may be difficult to find a useful and fair metric by which these algorithms can be compared. In many other settings one may compare potential or gradient evaluations, however the disparate mechanisms of these algorithms preclude such a simple comparison. We conjecture, in particular, that the implementation of the local bouncy particle sampler used here may be suboptimal. We leave for future work a thorough investigation into the efficient implementation of the novel algorithms.

The simulations presented here require further exploration. While the integrated autocorrelation time of a single coordinate may be representative of an algorithm's performance, a practitioner may typically be more interested in models including hyperparameters such as the interaction strength between sites. Hamiltonian Monte Carlo can be applied without extension to such situations. For the local models the situation is less straightforward as the hyperparameters have an effect on all sites and increases dependencies between remote parts of the graph.

Figure 5.1: Gaussian-Poisson sampling results for Hamiltonian Monte Carlo. Each panel corresponds to a model grid size.

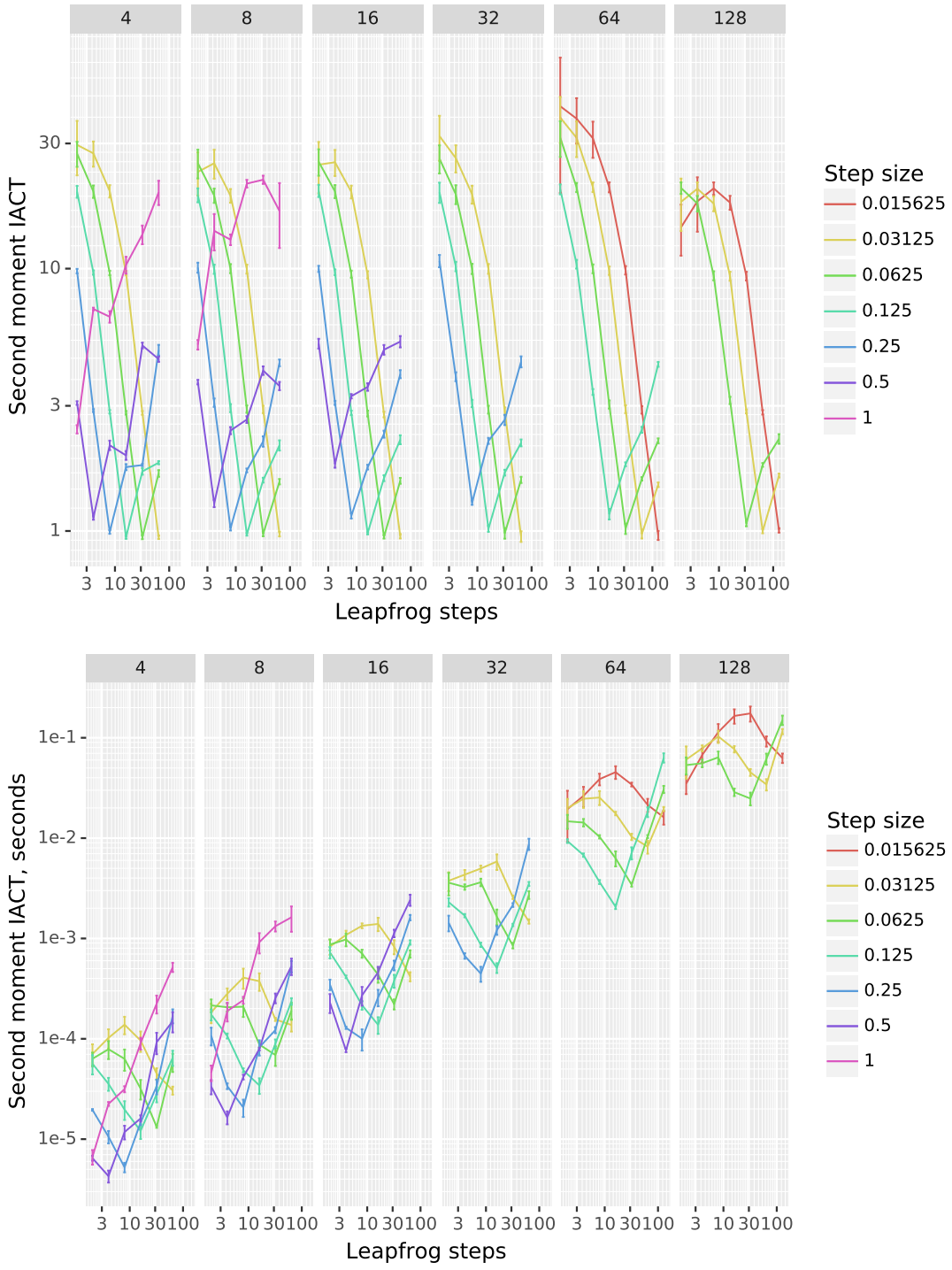


Figure 5.2: Gaussian-Poisson sampling results for the local bouncy particle sampler. Each panel corresponds to a grid size. Refresh rate is λ_{ref} in the text.

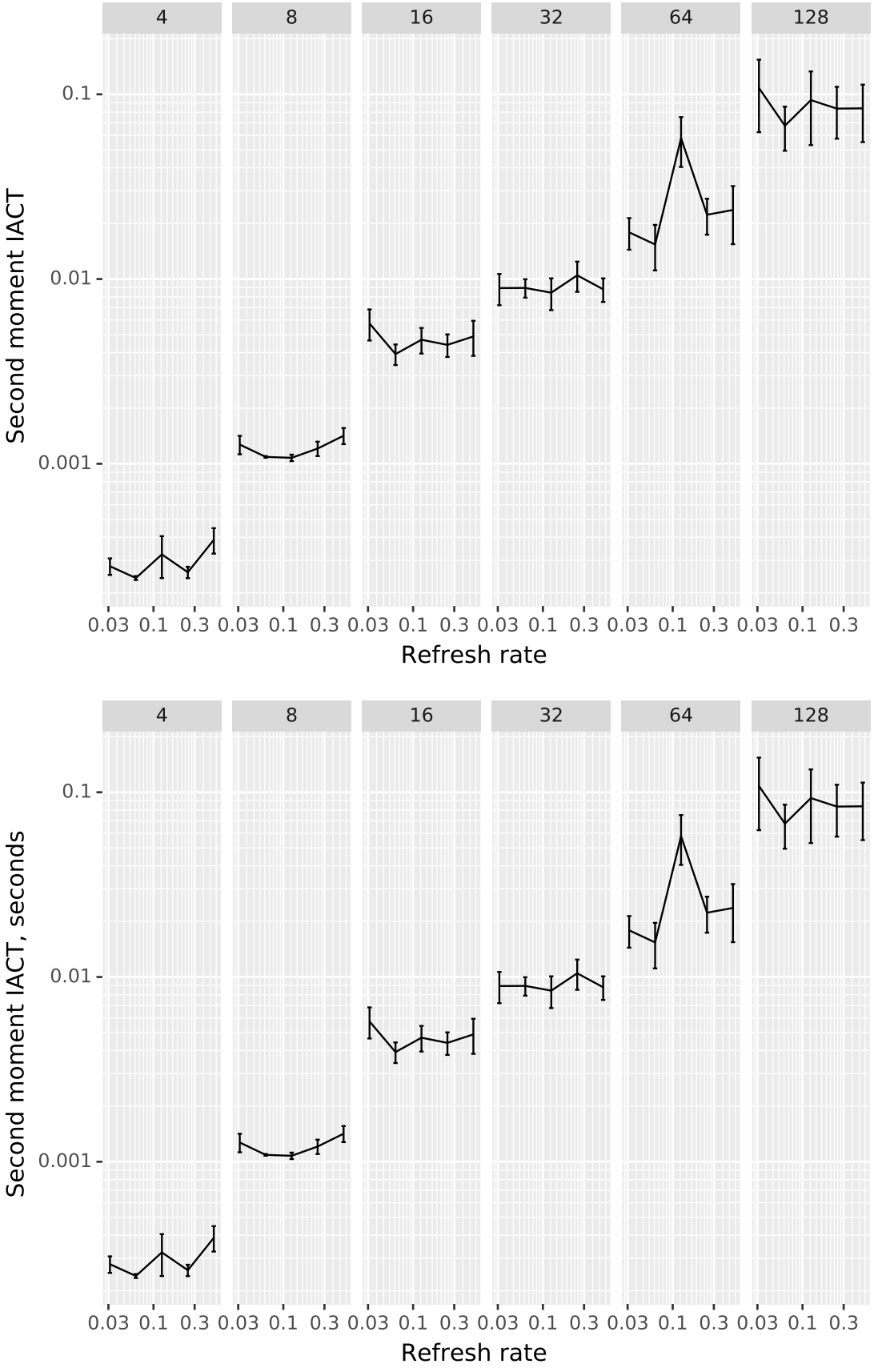


Figure 5.3: Gaussian-Poisson sampling results for the Cluster Random Walk algorithm with Gaussian proposals drawn from $\mathcal{N}(0, \sigma I)$ where σ the velocity scale.

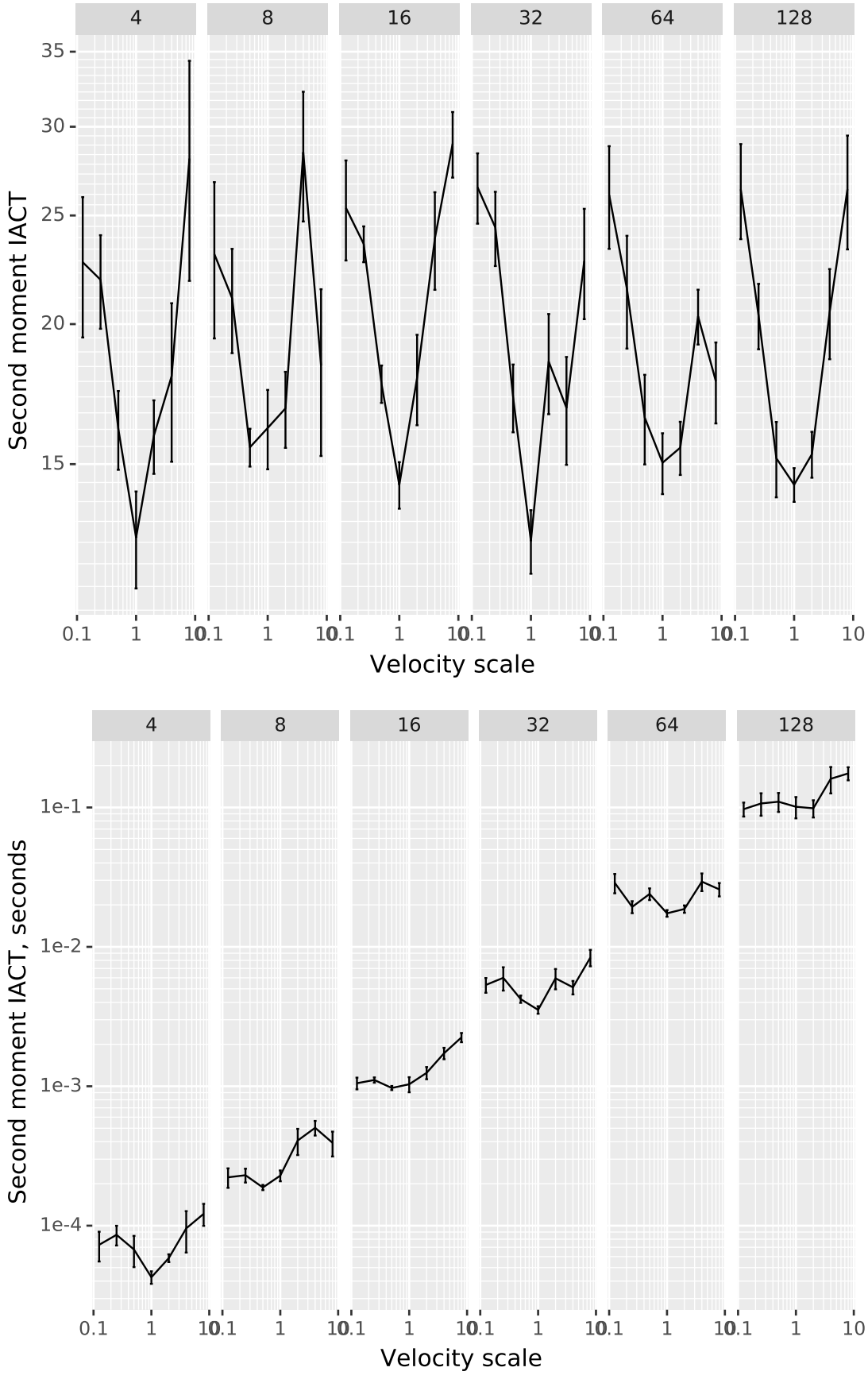
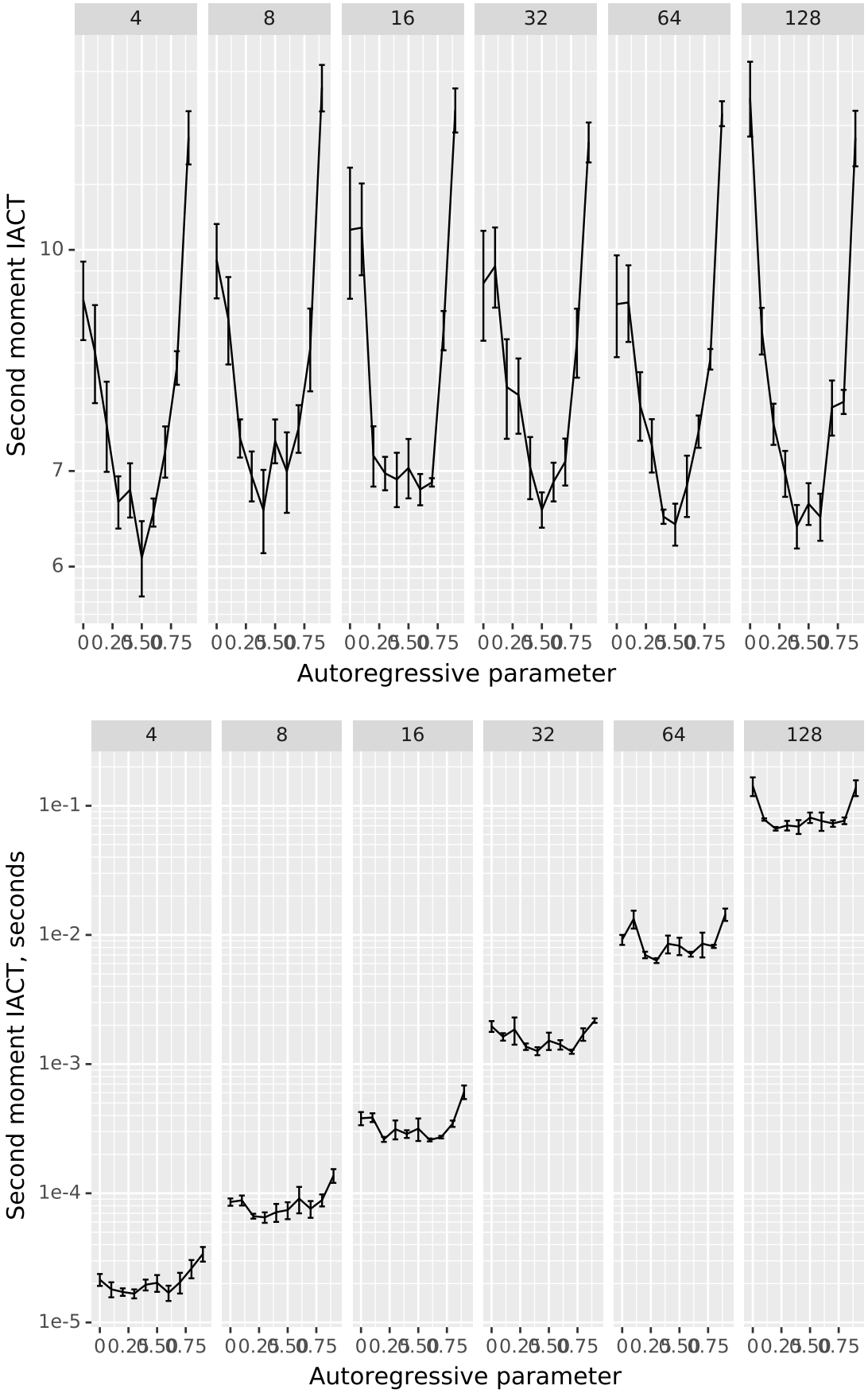


Figure 5.4: Gaussian-Poisson sampling results for the Cluster Random Walk algorithm with autoregressive proposals. The autoregressive parameter is ρ in the text.



6
Conclusion

To conclude, we summarize the main contributions of this thesis and suggest avenues for future work, considering each chapter in turn.

- In Chapter 2, we reviewed the basic conditions for a continuous-time piecewise-deterministic Markov process to leave invariant a target distribution ρ. We presented conditions for so-called *decomposable rates*. The conditions on the event kernel were expressed as an invariance condition for a lifted chain targeting the jump chain of the piecewise-deterministic Markov process.

  The conditions on the event kernels were shown to be connected to simulation methods widely deployed in the rarefied gas literature. Using a decomposition of the velocity into a component parallel to the gradient and a component perpendicular to the gradient, we derive a factorization of the jump chain distribution which allows a simple construction of event kernels using Markov chain Monte Carlo transitions for each of these components independently. All known proposals for event kernels in the literature can be described in this framework. Empirical studies show that among these event kernels the best performance is seen for methods which fix (or slightly perturb) the gradient-perpendicular component while randomizing the gradient-parallel

component. The reasons for this are unclear. We suggest that a thorough analysis of these event kernels may yield useful insights.

A generalization of the flow to use Hamiltonian dynamics was derived. As compared with the exact Hamiltonian Monte Carlo sampler (Pakman and Paninski 2014), this offers a means to account for potentials beyond constrained Gaussian distributions. This Hamiltonian flow represents the only alternative to the linear flow; other flows suitable for continuous-time algorithms are elusive.

Finally, we described a new type of continuous-time event kernel, which we called the *tunnelling* kernel. The method requires finding an exact potential along a flow and may often be computationally inefficient. In Chapter 3 we review the several existing tunnelling algorithms that have appeared as discrete-time algorithms, and which require only that one can evaluate the potential at the points of the discrete-time trajectory — which is necessary to implement such an algorithm anyway. Thus we conjecture that bounce kernels may be more efficient in continuous-time, and tunnelling kernels more efficient in discrete-time.

- In Chapter 3, we developed discrete-time algorithms analogous to the continuous-time algorithms of Chapter 2. Simulating multiple event-rates in continuous-time has a discrete-time counterpart in factorized acceptance rates, a method previously seen in algorithms such as delayed acceptance. In Chapters 4 and 5 we emphasize the importance of these multiple event-rate constructions, and conclude that the desirable scaling properties of these constructions are shared by factorized acceptance rates.

  In the process of deriving discrete-time versions of the decomposable rate, we drew connections between a wide range of algorithms. We showed how the discrete-time algorithm can be implemented by simulating a Poisson process in latent space, and showed how the latent Poisson process can also be viewed as a generalization of a slice variable. Furthermore, we showed how the

introduction of this Poisson process connects these algorithms with several existing algorithms involving latent processes. Straightforward adaptations to the case of an atomic measure allowed for connections to multiple slice sampling and sparsity-inducing constructions such as Firefly Monte Carlo.

One advantage of continuous-time methods is that the selection of any sort of scale or step size is avoided. We showed that the information used to compute event times in continuous-time can also be used in discrete-time which we suggest can ameliorate, to some extent, the problem of selecting too small a step size.

We discussed how bounces might be implemented in discrete-time, noting that these were first implemented in the reflective slice sampler.

Finally, we reviewed a class of event kernels providing an alternative to bounces. We termed this class of algorithms *tunnelling*. Comparing these with the proposed bounce schemes, we suggest that tunnelling will typically be more effective due to the simplicity of its implementation. A thorough comparison of empirical performance for these schemes would be necessary to justify this stance.

- In Chapter 4, we addressed simulation for posteriors of large datasets. By leveraging an approximation to the posterior we ware able to bound the influence of individual observations. Extending the continuous-time mechanisms of Poisson process thinning for multiple event-rates we realize a discrete-time algorithm with desirable scaling properties.

  While we can apply piecewise-deterministic ideas to a discrete-time sampler in this case, we suggest that the necessity of developing an approximation to the posterior also yields a proposal mechanism superior to dynamics of a piecewise-deterministic algorithm. We presented the scalable Metropolis-Hastings algorithm, allowing for arbitrary proposal distributions.

  We showed how the Poisson process thinning procedure can be applied to subset selection in the firefly Monte Carlo algorithm. It was found empirically

that the firefly algorithm yielded slightly worse performance than the scalable Metropolis-Hastings algorithm. We conjectured that this was due to the reliance of firefly Monte Carlo on a lower bound to the posterior density, rather than an approximation as in scalable Metropolis-Hastings; a detailed analysis of this discrepancy would be an interesting avenue for future research.

Several aspects of the algorithms presented in Chapter 4 merit further attention. The bounds (4.14) do not take into account the distance between the current state x and the proposed state x' and thus appear overly coarse; we thus conjecture that there is an opportunity to bound the rejection probability more tightly and improve the efficiency of the algorithm, even if only by a constant-time factor.

Additionally, these algorithms require quite restrictive conditions on the likelihood, relying heavily on the ability to uniformly bound derivatives of the likelihood. This precludes the possibility of extending the framework even to very similar models such as a probit likelihood. It may be interesting to explore what sort of extensions may be necessary to allow greater applicability of the method.

- In Chapter 5 we developed a discrete-time method based on clustering, inspired by the local effects of events in the local bouncy particle sampler. By conditioning on the value of the proposal state we derived an valid algorithm in which the proposal state is accepted for only a subset of coordinates. We demonstrated that the method can be implemented by using a factorized acceptance probability and then building clusters around those factors which are rejected.

  We described how the algorithm can be specialized when targeting a model with a latent Gaussian field. In this case, an autoregressive proposal yields an algorithm where the accept-reject step only depends on the observation factors.

As the algorithm is closely connected to random cluster methods, we can perhaps derive extensions from simulation methods used in that field. For example, the clustering algorithm of Wolff (1989) provides an interesting alternative which may be worth exploring; we conjecture that an algorithm based on this could yield more efficient updates by sampling larger clusters on average, as in the original Wolff algorithm.

We described how the problem of partial updates may be seen as a boolean satisfiability problem and we overconstrained the problem to yield a tractable solution. It may be possible in some settings to consider the larger space of solutions, which may still be tractable in some interesting scenarios. For example, in time series models where the parameter dependencies form a one-dimensional chain, we conjecture that a more flexible approach may be feasible.

A major theme of this thesis has been to establish connections between a wide range of algorithms. We argue that this has enabled a new perspective on the more recently developed continuous-time algorithms, many aspects of which which prove to be closely related to ideas from older and more established algorithms. Innovations realized in continuous-time algorithms have inspired novel discrete-time algorithms demonstrating similar properties, suggesting that the advantages found in continuous-time are sometimes attributable to mechanisms transcending continuous-time simulation. By unifying and comparing these algorithms we are able to understand the fundamental merits of each algorithm and guide the development of new and interesting methods.

Bibliography

R. P. Adams, I. Murray, and D. J. MacKay. Tractable nonparametric Bayesian inference in Poisson processes with Gaussian process intensities. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 9–16. ACM, 2009.

H. M. Afshar and J. Domke. Reflection, refraction, and Hamiltonian Monte Carlo. In *Advances in Neural Information Processing Systems*, pages 3007–3015, 2015.

J. H. Albert and S. Chib. Bayesian analysis of binary and polychotomous response data. *Journal of the American statistical Association*, 88(422):669–679, 1993.

C. Andrieu and S. Livingstone. Peskun-Tierney ordering for Markov chain and process Monte Carlo: beyond the reversible scenario. *arXiv preprint arXiv:1906.06197*, 2019.

M. Banterle, C. Grazian, A. Lee, and C. P. Robert. Accelerating Metropolis-Hastings algorithms by delayed acceptance. *arXiv preprint arXiv:1503.00996*, 2015.

R. Bardenet, A. Doucet, and C. Holmes. On Markov chain Monte Carlo methods for tall data. *The Journal of Machine Learning Research*, 18(1):1515–1557, 2017.

A. A. Barker. Monte Carlo calculations of the radial distribution functions for a proton-electron plasma. *Australian Journal of Physics*, 18(2):119–134, 1965.

A. Beskos, O. Papaspiliopoulos, G. O. Roberts, and P. Fearnhead. Exact and computationally efficient likelihood-based estimation for discretely observed diffusion processes (with discussion). *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(3):333–382, 2006.

J. Bierkens, G. Roberts, et al. A piecewise deterministic scaling limit of lifted Metropolis-Hastings in the Curie-Weiss model. *The Annals of Applied Probability*, 27(2):846–882, 2017.

J. Bierkens, P. Fearnhead, G. Roberts, et al. The zig-zag process and super-efficient sampling for Bayesian analysis of big data. *The Annals of Statistics*, 47(3): 1288–1320, 2019a.

J. Bierkens, G. O. Roberts, P.-A. Zitt, et al. Ergodicity of the zigzag process. *The Annals of Applied Probability*, 29(4):2266–2301, 2019b.

B. Bloem-Reddy and J. Cunningham. Slice sampling on Hamiltonian trajectories. In *International Conference on Machine Learning*, pages 3050–3058, 2016.

A. Bouchard-Côté, S. J. Vollmer, and A. Doucet. The bouncy particle sampler: A nonreversible rejection-free Markov chain Monte Carlo method. *Journal of the American Statistical Association*, 113(522):855–867, 2018.

S. Brooks, A. Gelman, G. Jones, and X.-L. Meng. *Handbook of Markov chain Monte Carlo*. CRC press, 2011.

H. Cai. A random cluster algorithm for some continuous Markov random fields. *Journal of Statistical Computation and Simulation*, 79(11):1287–1299, 2009.

C. M. Campos and J. Sanz-Serna. Extra chance generalized hybrid Monte Carlo. *Journal of Computational Physics*, 281:365–374, 2015.

D. M. Ceperley. Path integrals in the theory of condensed helium. *Reviews of Modern Physics*, 67(2):279, 1995.

J. A. Christen and C. Fox. Markov chain Monte Carlo using an approximation. *Journal of Computational and Graphical statistics*, 14(4):795–810, 2005.

T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT press, 2009.

R. Cornish, P. Vanetti, A. Bouchard-Côté, G. Deligiannidis, and A. Doucet. Scalable Metropolis-Hastings for exact Bayesian inference with large datasets. *arXiv preprint arXiv:1901.09881*, 2019.

O. L. d. V. Costa. Stationary distributions for piecewise-deterministic Markov processes. *Journal of Applied Probability*, 27(1):60–73, 1990.

M. H. Davis. 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, 1984.

L. Devroye. Nonuniform random variate generation. *Handbooks in operations research and management science*, 13:83–121, 2006.

P. Diaconis, S. Holmes, and R. M. Neal. Analysis of a nonreversible Markov chain sampler. *Annals of Applied Probability*, pages 726–752, 2000.

O. B. Downs, D. J. MacKay, and D. D. Lee. The nonnegative Boltzmann machine. In *Advances in Neural Information Processing Systems*, pages 428–434, 2000.

R. G. Edwards and A. D. Sokal. Generalization of the Fortuin-Kasteleyn-Swendsen-Wang representation and Monte Carlo algorithm. *Physical review D*, 38(6):2009, 1988.

S. N. Ethier and T. G. Kurtz. *Markov processes: characterization and convergence*. John Wiley & Sons, 2005.

P. Fearnhead, J. Bierkens, M. Pollock, G. O. Roberts, et al. Piecewise deterministic Markov processes for continuous-time Monte Carlo. *Statistical Science*, 33(3): 386–412, 2018.

J. M. Flegal, G. L. Jones, et al. Batch means and spectral variance estimators in Markov chain Monte Carlo. *The Annals of Statistics*, 38(2):1034–1070, 2010.

K. Fukui and S. Todo. Order-n cluster Monte Carlo method for spin systems with long-range interactions. *Journal of Computational Physics*, 228(7):2629–2642, 2009.

N. Galbraith. *On Event-Chain Monte Carlo Methods*. PhD thesis, Master's thesis, Department of Statistics, Oxford University, 2016.

G. Guennebaud, B. Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

W. K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. 1970.

J. Haviland. The solution of two molecular flow problems by the Monte Carlo method. *Methods in computational physics*, 4:109–209, 1965.

R. Herbrich, T. Graepel, and C. Campbell. Bayes point machines. *Journal of Machine Learning Research*, 1(Aug):245–279, 2001.

M. D. Hoffman and A. Gelman. The no-u-turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research*, 15 (1):1593–1623, 2014.

T. Jaakkola and M. Jordan. A variational approach to Bayesian logistic regression models and their extensions. In *Sixth International Workshop on Artificial Intelligence and Statistics*, volume 82, page 4, 1997.

J. F. C. Kingman. *Poisson processes*, volume 3 of *Oxford Studies in Probability*. The Clarendon Press Oxford University Press, New York, 1993. ISBN 0-19-853693-3. Oxford Science Publications.

R. A. Kronmal and A. V. Peterson Jr. On the alias method for generating random variables from a discrete distribution. *The American Statistician*, 33(4):214–218, 1979.

I. Kuščher. Reciprocity in scattering of gas molecules by surfaces. *Surface Science*, 25(2):225–237, 1971.

J. S. Liu. *Monte Carlo strategies in scientific computing*. Springer Science & Business Media, 2008.

D. Maclaurin and R. P. Adams. Firefly Monte Carlo: Exact MCMC with subsets of data. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

C. Mak. Stochastic potential switching algorithm for Monte Carlo simulations of complex systems. *The Journal of chemical physics*, 122(21):214110, 2005.

J. C. Maxwell. IV. on the dynamical theory of gases. *Philosophical transactions of the Royal Society of London*, (157):49–88, 1867.

A. R. Mesquita and J. P. Hespanha. Jump control of probability densities with applications to autonomous vehicle motion. *IEEE Transactions on Automatic Control*, 57(10):2588–2598, 2012.

N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

M. Michel, S. C. Kapfer, and W. Krauth. Generalized event-chain Monte Carlo: Constructing rejection-free global-balance algorithms from infinitesimal steps. *The Journal of chemical physics*, 140(5):054116, 2014.

M. Michel, A. Durmus, and S. Sénécal. Forward event-chain Monte Carlo: Fast sampling by randomness control in irreversible Markov chains. *arXiv preprint arXiv:1702.08397*, 2017.

M. Michel, X. Tan, and Y. Deng. Clock Monte Carlo methods. *Physical Review E*, 99(1):010105, 2019.

J. Moriarty, J. Vogrinc, and A. Zocca. The skipping sampler: A new approach to sample from complex conditional densities. *arXiv preprint arXiv:1905.09964*, 2019.

I. Murray, D. MacKay, and R. P. Adams. The Gaussian process density sampler. In *Advances in Neural Information Processing Systems*, pages 9–16, 2009.

I. Murray, R. Adams, and D. MacKay. Elliptical slice sampling. In Y. W. Teh and M. Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 541–548, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL `http://proceedings.mlr.press/v9/murray10a.html`.

R. M. Neal. Slice sampling. *The Annals of Statistics*, 31(3):705–767, 2003.

R. M. Neal. New Monte Carlo methods based on Hamiltonian dynamics. In *31st International Workshop on Bayesian Inference and Maximum Entropy Methods in Science and Engineering*, 2011. URL `https://www.cs.toronto.edu/~radford/ftp/maxent2011.pdf`.

R. M. Neal et al. MCMC using Hamiltonian dynamics. *Handbook of markov chain monte carlo*, 2(11):2, 2011.

A. Nishimura, D. Dunson, and J. Lu. Discontinuous Hamiltonian Monte Carlo for sampling discrete parameters. *arXiv preprint arXiv:1705.08510*, 2017.

E. Nummelin. *General irreducible Markov chains and non-negative operators*, volume 83. Cambridge University Press, 2004.

A. Pakman and L. Paninski. Exact Hamiltonian Monte Carlo for truncated multivariate Gaussians. *Journal of Computational and Graphical Statistics*, 23(2): 518–542, 2014.

A. Pakman, D. Gilboa, D. Carlson, and L. Paninski. Stochastic bouncy particle sampler. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2741–2750. JMLR. org, 2017.

J. Park and Y. F. Atchadé. Markov chain Monte Carlo algorithms with sequential proposals. *arXiv preprint arXiv:1907.06544*, 2019.

P. H. Peskun. Optimum Monte-Carlo sampling using Markov chains. *Biometrika*, 60(3):607–612, 1973.

E. A. Peters et al. Rejection-free Monte Carlo sampling for general potentials. *Physical Review E*, 85(2):026703, 2012.

V. Rao, L. Lin, and D. B. Dunson. Data augmentation for models based on rejection sampling. *Biometrika*, 103(2):319–335, 2016.

V. A. Rao. *Markov chain Monte Carlo for continuous-time discrete-state systems*. PhD thesis, UCL (University College London), 2012.

H. Rue and L. Held. *Gaussian Markov random fields: theory and applications*. Chapman and Hall/CRC, 2005.

J. Shanthikumar. Discrete random variate generation using uniformization. *European Journal of Operational Research*, 21(3):387–398, 1985.

C. Shen. *Rarefied gas dynamics: fundamentals, simulations and micro flows*. Springer Science & Business Media, 2006.

C. Sherlock and A. H. Thiery. A discrete bouncy particle sampler. *arXiv preprint arXiv:1707.05200*, 2017.

J. Skilling. Bayesian computation in big spaces - nested sampling and Galilean Monte Carlo. In *AIP Conference Proceedings 31st*, volume 1443, pages 145–156. AIP, 2012.

J. Sohl-Dickstein, M. Mudigonda, and M. R. DeWeese. Hamiltonian Monte Carlo without detailed balance. *arXiv preprint arXiv:1409.5191*, 2014.

R. H. Swendsen and J.-S. Wang. Nonuniversal critical dynamics in Monte Carlo simulations. *Physical review letters*, 58(2):86, 1987.

A. Terenin and D. Thorngren. A piecewise deterministic Markov Process via (r, θ) swaps in hyperspherical coordinates. *arXiv preprint arXiv:1807.00420*, 2018.

L. Tierney. Markov chains for exploring posterior distributions. *the Annals of Statistics*, pages 1701–1728, 1994.

L. Tierney. A note on Metropolis-Hastings kernels for general state spaces. *The Annals of Applied Probability*, 8(1):1–9, 1998.

L. Tierney and A. Mira. Some adaptive Monte Carlo methods for Bayesian inference. *Statistics in medicine*, 18(17-18):2507–2515, 1999.

K. S. Turitsyn, M. Chertkov, and M. Vucelja. Irreversible Monte Carlo algorithms for efficient sampling. *Physica D: Nonlinear Phenomena*, 240(4-5):410–414, 2011.

P. Vanetti, A. Bouchard-Côté, G. Deligiannidis, and A. Doucet. Piecewise-deterministic Markov chain Monte Carlo. *arXiv preprint arXiv:1707.05296*, 2017.

A. J. Walker. An efficient method for generating discrete random variables with general distributions. *ACM Transactions on Mathematical Software (TOMS)*, 3 (3):253–256, 1977.

Q. Wang, V. Rao, and Y. W. Teh. An exact auxiliary variable Gibbs sampler for a class of diffusions. *arXiv preprint arXiv:1903.10659*, 2019.

U. Wolff. Collective Monte Carlo updating for spin systems. *Physical Review Letters*, 62(4):361, 1989.

C. Wu and C. P. Robert. Generalized bouncy particle sampler. *arXiv preprint arXiv:1706.04781*, 2017.

K. Yi and F. Doshi-Velez. Roll-back Hamiltonian Monte Carlo. *arXiv preprint arXiv:1709.02855*, 2017.