Piecewise Deterministic Markov Processes for Continuous-Time Monte Carlo

Paul Fearnhead[1,†], Joris Bierkens[2], Murray Pollock[3] and Gareth O Roberts[3]

[1]Department of Mathematics and Statistics, Lancaster University

[2]DIAM, TU Delft

[3]Department of Statistics, University of Warwick

[†]Correspondence: p.fearnhead@lancaster.ac.uk

February 16, 2018

Abstract

Recently there have been conceptually new developments in Monte Carlo methods through the introduction of new MCMC and sequential Monte Carlo (SMC) algorithms which are based on continuous-time, rather than discrete-time, Markov processes. This has led to some fundamentally new Monte Carlo algorithms which can be used to sample from, say, a posterior distribution. Interestingly, continuous-time algorithms seem particularly well suited to Bayesian analysis in big-data settings as they need only access a small sub-set of data points at each iteration, and yet are still guaranteed to target the true posterior distribution. Whilst continuous-time MCMC and SMC methods have been developed independently we show here that they are related by the fact that both involve simulating a piecewise deterministic Markov process. Furthermore we show that the methods developed to date are just specific cases of a potentially much wider class of continuous-time Monte Carlo algorithms. We give an informal introduction to piecewise deterministic Markov processes, covering the aspects relevant to these new Monte Carlo algorithms, with a view to making the development of new continuous-time Monte Carlo more accessible. We focus on how and why sub-sampling ideas can be used with these algorithms, and aim to give insight into how these new algorithms can be implemented, and what are some of the issues that affect their efficiency.

Keywords: Bayesian Statistics, Big data, Bouncy particle sampler, Continuous-time importance sampling, Control variates, SCALE, Zig-zag sampler.

1 Introduction

Monte Carlo methods, such as MCMC and SMC, have been central to the application of Bayesian statistics to real-world problems (Robert and Casella, 2011; McGrayne, 2011). These established Monte Carlo methods are based upon simulating discrete-time Markov processes. For example MCMC algorithms simulate a discrete-time Markov chain constructed to have a target distribution of interest, the posterior distribution in Bayesian inference, as its stationary distribution. Whilst SMC methods involve propagating and re-weighting particles so that a final set of weighted particles approximate a target distribution. The propagation step here also involves simulating from a discrete-time Markov chain.

In the past few years there have been new developments in MCMC and SMC methods based on continuous-time versions of these Monte Carlo methods. For example, continuous-time MCMC algorithms have been proposed (Peters and de With, 2012; Bouchard-Côté et al., 2017; Bierkens et al., 2017b; Bierkens et al., 2016) that involve simulating a continuous-time Markov process that has been designed to have a target distribution of interest as its stationary distribution. These continuous-time MCMC algorithms were originally motivated as they are examples of non-reversible Markov processes. There is substantial evidence that non-reversible MCMC algorithms will be more efficient than standard MCMC algorithms that are reversible (Neal, 1998; Diaconis et al., 2000; Neal, 2004; Bierkens, 2015), and there is empirical evidence that these continuous-time MCMC algorithms are more efficient than their discrete-time counterparts (see e.g. Bouchard-Côté et al., 2017). Similarly a continuous-time version of SMC has also been recently proposed (Fearnhead et al., 2016), which involves propagating particles using a continuous-time Markov process. The original motivation for this was to be able to target distributions related to infinite-dimensional stochastic processes, such as diffusions, without resorting to any time-discretisation approximations. However, we show below that one application of this methodology is to generate weighted-samples from a target distribution of interest, giving an alternative interpretation of the recently proposed SCALE algorithm of Pollock et al. (2016a).

The purpose of this paper is to show that continuous-time MCMC and continuous-time SMC methods are linked through the fact that they both are based upon simulating continuous-time processes called piecewise-deterministic Markov processes. These are processes that evolve deterministically between a countable set of random event times. The stochasticity in the process is due to the randomness regarding when these events occur, and possibly random dynamics at the event times. These processes are natural building blocks of continuous-time Monte Carlo methods, as they involve a finite amount of computation to simulate as only a finite number of events and transitions are simulated in any fixed time-interval.

Furthermore we aim to show that the methods that have been developed to date are just specific examples of a much wider class of potential continuous-time MCMC and SMC methods that are based on piecewise-deterministic Markov processes. By giving an informal introduction to theory of piecewise-deterministic Markov processes, with emphasis on aspects most relevant to the development of valid Monte Carlo methods, we hope to make the development of new continuous-time Monte Carlo methods more accessible and help stimulate work in this area.

One aspect of continuous-time Monte Carlo that is particularly relevant for modern applications of Bayesian statistics is that they seem well-suited to big-data problems. If we denote our target distribution by $\pi(\mathbf{x})$ then the dynamics of these methods depend on the target through $\nabla \log \pi(\mathbf{x})$. Now if $\pi(\mathbf{x})$ is a posterior distribution, then it will often be in product-form, where each factor relates to a data-point or set of data-points. This means that $\nabla \log \pi(\mathbf{x})$ is a sum, and hence is easy to approximate unbiasedly using sub-samples of the data. It turns out we can use these unbiased estimators within the continuous-time Monte Carlo methods without affecting their validity. That is, the algorithms will still target $\pi(\mathbf{x})$. This is in comparison to other discrete-time MCMC methods that use sub-sampling (Welling and Teh, 2011; Bardenet et al., 2017; Ma et al., 2015; Quiroz et al., 2015), where the approximation in the sub-sampling means that the algorithms will only target an approximation to $\pi(\mathbf{x})$. It also compares favourably with big-data methods that independently run MCMC on batches of data, and then combines the MCMC samples in some way (Neiswanger et al., 2014; Scott et al., 2016; Srivastava et al., 2015; Li et al., 2017). As the combination steps involved will also introduce some approximation error.

The outline of the paper is as follows. In the next section we give an informal introduction to Piecewise Deterministic Markov processes. Our aim is to cover key relevant concepts linked to these processes whilst avoiding technical details. Those interested in a more rigorous introduction should see Davis (1984) and Davis (1993). Sections 3 and 4 then cover continuous-time versions of MCMC and SMC respectively. These have been written so that either section could be read independently of the other. Our aim for each section is to introduce the continuous-time Monte Carlo algorithm, show how it relates to a piecewise deterministic Markov process, and how we can use the theory for these processes to see that the Monte Carlo algorithms target the correct distribution. We also cover how these algorithms can be implemented using sub-sampling ideas, and highlight the importance of low-variance sub-sampling estimators for obtaining highly efficient samplers for big-data.

2 Piecewise Deterministic Markov Processes

The continuous-time versions of SMC, or sequential importance sampling, and MCMC that we will consider later are all examples of time-homogeneous piecewise-deterministic Markov processes. We will henceforth call these piecewise deterministic processes or PDPs.

A PDP is a continuous-time stochastic process. Throughout we will denote the state of a PDP at time t by $\mathbf{Z}_t$. The dynamics of the PDP involves random events, with deterministic dynamics between events and possibly random transitions at events. These dynamics are thus defined through specifying three quantities:

(i) **The deterministic dynamics.** We will assume that these are specified through an ordinary differential equation:

$$\frac{\mathrm{d}z_t^{(i)}}{\mathrm{d}t} = \Phi_i(\mathbf{z}_t), \tag{1}$$

for $i = 1, \ldots, d$, for some known vector-valued function $\Phi = (\Phi_1(\mathbf{z}), \ldots, \Phi_d(\mathbf{z}))$. This will lead to a deterministic transition function, so that the solution of the differential equation starting from value $\mathbf{z}_t$ and run for a time interval of length s will give

$$\mathbf{z}_{s+t} = \Psi(\mathbf{z}_t, s)$$

for some function Ψ.

(ii) **The event rate.** Events will occur singularly at a rate, $\lambda(\mathbf{z}_t)$, that depends only on the current position of the process. The probability of an event in interval in $[t, t+h]$ given the state at time t, $\mathbf{z}_t$, is thus $\lambda(\mathbf{z}_t)h + o(h)$.

(iii) **The transition distribution at events.** At each event the state of the process will change, according to some transition kernel. For an event at time τ, if $\mathbf{z}_{\tau-}$ denotes the state immediately prior to the event, then the state at time τ will be drawn from a distribution with density $q(\cdot|\mathbf{z}_{\tau-})$.

To define a PDP process we will also need to specify its initial condition. We will assume that $\mathbf{Z}_0$ is drawn from some known distribution with density function $p_0(\cdot)$.

2.1 Simulating a PDP

To be able to use a PDP as the basis of an importance sampling or MCMC algorithm, we will need to be able to simulate from it. A general approach to simulating a PDP is to iterate the following steps:

(S1) Given the current time, t, and state of the PDP, $\mathbf{z}_t$, simulate the next event time, τ say.

(S2) Calculate the value of the process immediately before the next event time

$$\mathbf{z}_{\tau-} = \Psi(\mathbf{z}_t, \tau - t).$$

(S3) Simulate the new value of the process, immediately after the event, from $q(\mathbf{z}_\tau|\mathbf{z}_{\tau-})$.

The simulation algorithm is initiated with a current time $t = 0$ and with $\mathbf{Z}_0$ drawn from the initial distribution of the process. To simulate the process for a time interval T these steps can be iterated until the first event time after T. If we wish to then simulate the value of the process at a time, s say, between events we just find the event time, τ, immediately prior to s; the value of the process immediately after the event, $\mathbf{z}_\tau$; and then set

$$\mathbf{z}_s = \Psi(\mathbf{z}_\tau, s - \tau).$$

If s is a time before the first event we would use $\tau = 0$.

Below we will assume that our PDP has been chosen so that $\Psi(\cdot, \cdot)$ is known analytically and that the proposal distribution at events, $q(\cdot|\cdot)$, can be easily simulated from. Thus the only challenging step to simulating a PDP will be simulating the next event in step S1. This involves simulating the next event in a time-inhomogeneous Poisson process.

The first thing to note is that the event rate in (S1) can be written as a deterministic function of time, as the state dynamics are deterministic until the next event. If we are currently at time t with state $\mathbf{z}_t$, then for any future time $t + s$ before the next event, the state will be $\mathbf{z}_{t+s} = \Psi(\mathbf{z}_t, s)$. Thus the event rate will be

$$\lambda(\mathbf{z}_{t+s}) = \lambda(\Psi(\mathbf{z}_t, s)) = \tilde{\lambda}_{\mathbf{z}_t}(s),$$

for a suitably defined function $\tilde{\lambda}_{\mathbf{z}_t}(\cdot)$. We can simulate the time until the next event, s, as the time of the first event in a Poisson process of rate $\tilde{\lambda}_{\mathbf{z}_t}(u)$.

If the event rate is a simple function, then we can simulate events directly. Define $\Lambda_{\mathbf{z}}(s) = \int_0^s \tilde{\lambda}_{\mathbf{z}}(u)\mathrm{d}u$. We simulate a the time of an event, s say, by (i) simulating u, the realisation of an exponential random variable with rate 1, and (ii) finding $s > 0$ the solution of $\Lambda_{\mathbf{z}}(s) = u$ (e.g. Cinlar, 2013).

For more complicated rate functions either calculating $\Lambda_{\mathbf{z}}(s)$ or solving the equation in step (ii) may not be tractable. In such cases the most general approach to simulating event times is by thinning, or adaptive thinning (e.g. Lewis and Shedler, 1979).

If we can upper-bound the event rate, $\tilde{\lambda}_{\mathbf{z}_t}(u) < \lambda^+(u)$, then thinning works by simulating possible events of a Poisson process of rate $\lambda^+(u)$ and accepting a possible event at time u as an actual event with probability $\tilde{\lambda}_{\mathbf{z}_t}(u)/\lambda^+(u)$. The time of the first accepted event will be the time until the next event for our PDP. This requires the upper bound $\lambda^+(u)$ to be such that simulating events from a Poission process of rate $\lambda^+(u)$ is straightforward – for example $\lambda^+(u)$ is constant or linear in u, or piecewise constant or piecewise linear. Obviously the lower the bound λ^+ the more computationally efficient this approach will be.

2.2 Analysing a PDP

We now give informal introductions to some of the mathematical tools for analysing a PDP. These are introduced as they are used later to show that the continuous-time Monte Carlo methods we review have appropriate properties. For example, we introduce the generator of a PDP in the following section, and this is used to show that the continuous-time importance samplers of Section 4 produce properly weighted samples (Liu and Chen, 1998). We then introduce the Fokker-Planck equation for a PDP, which can be used to showed that the continuous-time MCMC methods of Section 3 have the correct invariant distribution. Understanding both the generator of a PDP and its Fokker-Planck equation is important if we wish to develop new versions of these continuous-time Monte Carlo methods. For further details on generators see Section 14 of Davis (1993), and for further information on calculating the invariant distribution if a PDP see Section 34 of Davis (1993).

2.2.1 The Generator

The generator of a continuous-time, time-homogeneous, Markov process is an operator that acts on functions of the state variable. We will denote the generator by $\mathcal{A}$. For suitable functions $f(\mathbf{z})$, the generator is defined by

$$\mathcal{A}f(\mathbf{z}) = \lim_{\delta t \to 0} \frac{\mathrm{E}(f(\mathbf{Z}_{t+\delta t})|\mathbf{Z}_t = \mathbf{z}) - f(\mathbf{z})}{\delta t}.$$

The set of suitable functions, which are the functions for which this limit exists for all $\mathbf{z}$, is called the domain of the generator.

The fact that the process is time-homogeneous means that the right-hand side does not depend on t. We can interpret the generator applied to a function $f(\mathbf{z})$, as giving the derivative of the expectation of $f(\mathbf{Z}_t)$ conditional on the current value of $\mathbf{Z}_t$. As the generator specifies how the expectation of any suitable function $f(\cdot)$ changes over time, it uniquely defines the dynamics of the underlying continuous-time stochastic process, in a similar way that knowing the moment generating function of a random variable will uniquely determine its distribution (Ethier and Kurtz, 2005).

If we are interested in the derivative of the expectation of a function of our PDP at a time t, then we can write this as

$$\frac{\mathrm{dE}(f(Z_t))}{\mathrm{d}t} = \lim_{\delta t \to 0} \frac{\mathrm{E}(f(\mathbf{Z}_{t+\delta t}) - \mathrm{E}(f(\mathbf{Z}_t))}{\delta t} = \lim_{\delta t \to 0} \mathrm{E}_t \left(\frac{\mathrm{E}_{t+\delta t|t}(f(\mathbf{Z}_{t+\delta t})|\mathbf{Z}_t) - f(\mathbf{Z}_t)}{\delta t} \right),$$

where in the last expression the inner expectation is with respect to $\mathbf{Z}_{t+\delta t}$ given $\mathbf{Z}_t$ and the outer expectation with respect to $\mathbf{Z}_t$. Assuming we can exchange the outer expectation and the limit we get

$$\frac{\mathrm{dE}(Z_t)}{\mathrm{d}t} = \mathrm{E}_t\left(\mathcal{A}f(\mathbf{Z}_t)\right). \tag{2}$$

Thus the derivative of the expectation of our function is the expectation of the generator applied to the function.

Davis (1984) gives the generator for a piecewise deterministic process:

$$\mathcal{A}f(\mathbf{z}) = \Phi(z) \cdot \nabla f(\mathbf{z}) + \lambda(\mathbf{z}) \int q(\mathbf{z}'|\mathbf{z})[f(\mathbf{z}') - f(\mathbf{z})]\mathrm{d}\mathbf{z}', \tag{3}$$

for functions $f(\cdot)$ such that $t \mapsto f(\Psi(z,t))$ is absolutely continuous. The form of the generator has a simple interpretation. The first term on the right-hand side relates to the deterministic dynamics. For deterministic dynamics the generator is just the time-derivative of $f(\mathbf{z}_t)$, which by the product rule is

$$\frac{\mathrm{d}f(\mathbf{z}_t)}{\mathrm{d}\mathbf{z}_t} = \sum_{i=1}^{d} \frac{\partial f(\mathbf{z}_t)}{\partial \mathbf{z}_t^{(i)}} \frac{\partial \mathbf{z}_t^{(i)}}{\partial t} = \Phi(\mathbf{z}) \cdot \nabla f(\mathbf{z}),$$

where $\Phi(\mathbf{z})$ is defined in (1). The second term on the right-hand side is the change in expectation at events. The probability of an event in time $[t, t+h]$ is $\lambda(\mathbf{z}_t)h + o(h)$ and the change in expectation conditional on event occuring, up to terms of $o(h)$, is given by the integral on the right-hand side.

2.2.2 The Forward Operator and Fokker-Planck Equation

We can define the adjoint of a generator of a continuous-time Markov process, $\mathcal{A}^*$, such that for suitable functions $g(\mathbf{z})$ and $f(\mathbf{z})$

$$\int g(\mathbf{z})\mathcal{A}f(\mathbf{z})\mathrm{d}\mathbf{z} = \int f(\mathbf{z})\mathcal{A}^*g(\mathbf{z})\mathrm{d}\mathbf{z}.$$

Now if we define the density function of our continuous-time Markov process at time t to be $p_t(\mathbf{z})$ then from (2) we get that, for suitable function f, the derivate of the expectation of $f(Z_t)$ with respect to t is

$$\frac{\mathrm{dE}(Z_t)}{\mathrm{d}t} = \mathrm{E}_t\left(\mathcal{A}f(\mathbf{Z}_t)\right) = \int p_t(\mathbf{z})\mathcal{A}f(\mathbf{z})\mathrm{d}\mathbf{z} = \int f(\mathbf{z})\mathcal{A}^*p_t(\mathbf{z})\mathrm{d}\mathbf{z}.$$

However we can equally write this derivative as

$$\frac{\mathrm{dE}(Z_t)}{\mathrm{d}t} = \frac{\mathrm{d}}{\mathrm{d}t}\int p_t(\mathbf{z})f(\mathbf{z})\mathrm{d}\mathbf{z} = \int \frac{\partial p_t(\mathbf{z})}{\partial t}f(\mathbf{z})\mathrm{d}\mathbf{z},$$

again assuming we can interchange differentiation and integration. This gives that

$$\int \frac{\mathrm{d}p_t(\mathbf{z})}{\mathrm{d}t}f(\mathbf{z})\mathrm{d}\mathbf{z} = \int f(\mathbf{z})\mathcal{A}^*p_t(\mathbf{z})\mathrm{d}\mathbf{z}.$$

As this holds for sufficiently many functions $f(\mathbf{z})$ we get

$$\frac{\partial p_t(\mathbf{z})}{\partial t} = \mathcal{A}^*p_t(\mathbf{z}).$$

This is a partial differential equation for the distribution of the stochastic process, known as the Fokker-Planck or Forward Kolmogorov equation.

It is straightforward to show that the adjoint of the generator of a PDP (3) is

$$\mathcal{A}^*g(\mathbf{z}) = -\sum_{i=1}^{d}\frac{\partial\Phi_i(\mathbf{z})g(\mathbf{z})}{\partial\mathbf{z}^{(i)}} + \int g(\mathbf{z}')\lambda(\mathbf{z}')q(\mathbf{z}|\mathbf{z}')\mathrm{d}\mathbf{z}' - g(\mathbf{z})\lambda(\mathbf{z}) \tag{4}$$

The first term equates to the adjoint of the first term of the generator, and is obtained using integration by parts. The second term equates to the adjoint of the second-term of the generator, and can be obtained by using a change of variables within the integral.

If $p(\mathbf{z})$ is an invariant distribution of our PDP than it will satisfy $\mathcal{A}^* p(\mathbf{z}) = 0$, which gives

$$-\sum_{i=1}^{d} \frac{\partial \Phi_i(\mathbf{z}) p(\mathbf{z})}{\partial \mathbf{z}^{(i)}} + \int p(\mathbf{z}')\lambda(\mathbf{z}')q(\mathbf{z}|\mathbf{z}')\mathrm{d}\mathbf{z}' - p(\mathbf{z})\lambda(\mathbf{z}) = 0.$$

The first term here relates to the change in probability mass caused by the deterministic dynamics, the second term relates to the probability flow into state $\mathbf{z}$ and the final term the probability flow out of state $\mathbf{z}$. For an invariant distribution these will cancel for all states $\mathbf{z}$.

3 Continuous-Time MCMC

We first consider continuous-time versions of MCMC. These algorithms involve simulating a PDP process which has a given target distribution, $\pi(\mathbf{x})$, as its stationary distribution. Such algorithms were originally of interest as they are non-reversible processes. As mentioned in the introduction, there is substantial evidence that non-reversible MCMC algorithms are more efficient than standard, reversible MCMC. Intuitively this is because non-reversible MCMC suppresses the random-walk behaviour of reversible MCMC and thus can more rapidly explore the state-space. Furthermore it has been shown that continuous-time MCMC is suitable for using sub-sampling ideas, similar to those in Section 4.3. Thus these methods are also promising for big-data applications of MCMC.

3.1 The Continuous-time limit of MCMC

To help build intuition for continuous-time MCMC, and to see how it links to discrete-time MCMC algorithms, we will first derive a continuous-time algorithm as a limiting form for a simple non-reversible discrete-time MCMC algorithm (Gustafson, 1998; Diaconis et al., 2000). This MCMC algorithm will target a joint distribution of $(\mathbf{x}, \mathbf{v})$, where $\mathbf{v}$ can be viewed as a velocity. For our specific algorithm we will consider only velocities of a fixed, say unit, speed, and hence $\mathbf{v}$ could equally be defined as a direction. Our MCMC will target a distribution $\pi(\mathbf{x})p_u(\mathbf{v})$ where $p_u(\mathbf{v})$ will be the uniform distribution over all velocities with unit speed.

The MCMC algorithm will have two types of move. The first involves two deterministic proposals

(1a) Propose a move from $(\mathbf{x}, \mathbf{v})$ to $(\mathbf{x} + h\mathbf{v}, -\mathbf{v})$. Accept this with the standard Metropolis-Hastings accept probability, which simplifies to

$$\min\left\{1, \frac{\pi(\mathbf{x} + h\mathbf{v})}{\pi(\mathbf{x})}\right\}$$

(1b) Move from $(\mathbf{x}', \mathbf{v}')$ to $(\mathbf{x}', -\mathbf{v}')$.

Both the moves in (1a) and (1b) are reversible, and can be shown to satisfy detailed balance. To make step (1a) reversible we have to propose a move which flips the velocity, and hence in (1b) we flip the velocity back again. So the net affect of applying both (1a) and (1b) is that the velocity is unchanged if we accept the proposed move in step (1a) but flips if we reject the move. This is a standard approach in Hamiltonian Monte Carlo (Neal et al., 2011). In fact this algorithm can be viewed as a type of Hamiltonian Monte Carlo move, but based on the dynamics of an approximate potential for $\mathbf{x}$ which is uniform (and hence the velocity is not changed other than by the flip).

Whilst this move keeps $\pi(\mathbf{x})p_u(\mathbf{v})$ invariant, it leads to a reducible Markov chain if the dimension of $\mathbf{x}$ is greater than 1, as it only proposes moves along the direction given by $\mathbf{v}$. Thus we need a second type of move to produce an irreducible MCMC algorithm with the required asymptotic distribution. The second move we use is an update of $\mathbf{v}$, from some transition kernel that has $p_u(\mathbf{v})$ as its stationary distribution. We will imagine applying N transitions of type 1 between each of these updates just of $\mathbf{v}$.

Under this framework we can then consider letting $h \to 0$ while keeping $s = hN$ a constant. We will scale time so that the ith MCMC transition will occur at time ih, and define $(\mathbf{x}_t, \mathbf{v}_t)$ to be the value of the state after the ith MCMC transition for $ih \leq t < (i+1)h$.

Now for each move in step (1a) the rejection probability for small h is

$$\max\{0, 1-\exp[\log\pi(\mathbf{x}+h\mathbf{v})-\log\pi(\mathbf{x})]\} = \max\{0, 1-\exp[\mathbf{v}\cdot\nabla\log\pi(\mathbf{x})h+o(h)]\} = \max\{0, -\mathbf{v}\cdot\nabla\log\pi(\mathbf{x})h\}+o(h),$$

assuming that, for example, $\pi(\mathbf{x})$ is twice differentiable.

Thus in our limit as $h \to 0$, rejections in step (1a) will occur as events in a Poisson process of rate $\lambda(\mathbf{x}_t, \mathbf{v}_t) = \max\{0, -\mathbf{v}_t \cdot \nabla\pi(\mathbf{x}_t)\}$. The dynamics between these events will be deterministic, with $\mathbf{v}_t$ being constant and $\mathbf{x}_t$ changing as in a constant velocity model with velocity $\mathbf{v}_t$. At each event the velocity will just flip. Note that while the process is moving to areas of higher probability density, as defined by $\pi(\mathbf{x})$, the rate of the Poisson process will be 0. Thus events will only occur if the process is moving to areas of lower probability mass.

This limiting process is just a PDP with constant velocity dynamics between events, with the velocity changing at event times.

It is natural to consider a general class of PDP processes with these dynamics, and see what flexibility there is in choosing the distribution of the event times, and the distribution of the change of velocity at events, so that we still have a process whose marginal stationary distribution for $\mathbf{X}_t$ is $\pi(\mathbf{x})$. To do this, denote the state of our PDP by $\mathbf{Z}_t = (\mathbf{X}_t, \mathbf{V}_t)$, and assume our PDP has the following dynamics:

(i) **The deterministic dynamics.** For $i = 1, \dots, d$

$$\frac{\mathrm{d}x_t^{(i)}}{\mathrm{d}t} = v_t^{(i)}, \text{ and } \frac{\mathrm{d}v_t^{(i)}}{\mathrm{d}t} = 0.$$

The solution of these dynamics is given by $(\mathbf{x}_{t+s}, \mathbf{v}_{t+s}) = (\mathbf{x}_t + s\mathbf{v}_t, \mathbf{v}_t)$ for any $s > 0$.

(ii) **The event rate.** Events will occur at a rate, $\lambda(\mathbf{z}_t)$.

(iii) **The transition distribution at events.** At an event at time τ, $\mathbf{x}_\tau = \mathbf{x}_{\tau-}$ and $\mathbf{v}_\tau$ is drawn from some transition density $q(\cdot|\mathbf{x}_{\tau-}, \mathbf{v}_{\tau-})$.

We now need to consider how to choose the event rate and the transition density so that $\pi(\mathbf{x})$ is the marginal stationary distribution.

3.2 The Stationary Distribution of the PDP

A necessary condition for $\pi(\mathbf{x})$ to be the marginal stationary distribution of our PDP is that it is the marginal of an invariant distribution for the PDP. We will use the adjoint of the generator of our PDP to derive a condition on both the event rate and the transition distribution at events for the PDP to have $\pi(\mathbf{x})$ as the marginal of an invariant distribution.

As above, let $\mathbf{z} = (\mathbf{x}, \mathbf{v})$. Denote the invariant distribution of our PDP by $p(\mathbf{z})$. We can factorise this as the product of the marginal stationary distribution for $\mathbf{x}$ times the conditional for $\mathbf{v}$ given $\mathbf{x}$, and we wish to have $p(\mathbf{z}) = \pi(\mathbf{x})p(\mathbf{v}|\mathbf{x})$. If $\mathcal{A}^*$ is the adjoint of the generator of our PDP, as $p(\mathbf{z})$ in an invariant distibution we have $\mathcal{A}^* p(\mathbf{z}) = 0$. This gives

$$-\pi(\mathbf{x})p(\mathbf{v}|\mathbf{x})[\mathbf{v} \cdot \nabla_{\mathbf{x}} \log \pi(\mathbf{x}) + \mathbf{v} \cdot \nabla_{\mathbf{x}} \log p(\mathbf{v}|\mathbf{x}) + \lambda(\mathbf{z})] + \int \lambda(\mathbf{x}, \mathbf{v}')q(\mathbf{v}|\mathbf{x}, \mathbf{v}')\pi(\mathbf{x})p(\mathbf{v}'|\mathbf{x})\mathrm{d}\mathbf{v}' = 0. \tag{5}$$

In the above $\nabla_{\mathbf{x}}$ denotes the vector of first partial derivative with respect to the components of $\mathbf{x}$.

To date, all continuous-time MCMC algorithms have been designed so that under the invariant distribution $\mathbf{v}$ is independent of $\mathbf{x}$, and thus all components of $\nabla_{\mathbf{x}} \log \pi(\mathbf{v}|\mathbf{x})$ will be 0. If we wish to design such a process we need to choose $\lambda(\mathbf{x}, \mathbf{v})$ and $q(\mathbf{v}'|\mathbf{x}, \mathbf{v})$ such that, by rearranging (5),

$$p_v(\mathbf{v})\lambda(\mathbf{x}, \mathbf{v}) - \int \lambda(\mathbf{x}, \mathbf{v}')q(\mathbf{v}|\mathbf{x}, \mathbf{v}')p_v(\mathbf{v}')\mathrm{d}\mathbf{v}' = -p_v(\mathbf{v})\mathbf{v} \cdot \nabla_{\mathbf{x}} \log \pi(\mathbf{x}), \tag{6}$$

for some distribution $p_v(\mathbf{v})$ for the velocity. The left-hand side is measuring the net probability flow out of states with velocity $\mathbf{v}$, this must offset the change in probability mass for $\mathbf{V}$ caused by the deterministic dynamics, which is the term on the right-hand side.

Note that if we integrate (6) with respect to $\mathbf{v}$, the left-hand side is 0. So we get $\mathrm{E}(\mathbf{V}) \cdot \nabla \log \pi(\mathbf{x}) = 0$, where the expectation is with respect to the invariant distribution for the velocity. As this will need to hold for all $\mathbf{x}$, we can see that the invariant distribution for all components of the velocity must have zero mean.

The actual processes we describe in the next section all allow velocities within some symmetrical set, and are designed so that $p_v(\mathbf{v})$ is uniform on this set. They ensure (6) holds through deterministic dynamics at events. They introduce a "flip" operator, $F_{\mathbf{x}}$ say, that can depend on $\mathbf{x}$ and which satistifies $F_{\mathbf{x}}(F_{\mathbf{x}}(\mathbf{v})) = \mathbf{v}$. They then only allow transitions between pairs of velocities, $\mathbf{v}$ and $\mathbf{v}'$ that satisfy $\mathbf{v}' = F_{\mathbf{x}}(\mathbf{v})$ and, by definition of $F_{\mathbf{x}}$, $\mathbf{v} = F_{\mathbf{x}}(\mathbf{v}')$. Under this constraint on the transitions at events we get a simple set of equations that we need the event rates to satisfy. For any $\mathbf{v}$, and with $\mathbf{v}' = F_{\mathbf{x}}(\mathbf{v})$, it is straightforward to show that (6) holds if and only if

$$\lambda(\mathbf{x}, \mathbf{v}) - \lambda(\mathbf{x}, \mathbf{v}') = -\mathbf{v} \cdot \nabla_{\mathbf{x}} \log \pi(\mathbf{x}). \tag{7}$$

for all $\mathbf{x}$. Note that as this equation must also hold for $\mathbf{v}'$, we immediately see that only flip operators for which $F_{\mathbf{x}}(\mathbf{v}) \cdot \nabla_{\mathbf{x}} \log \pi(\mathbf{x}) = -\mathbf{v} \cdot \nabla_{\mathbf{x}} \log \pi(\mathbf{x})$ are allowable. The rates only depend on the target through the term $\nabla_{\mathbf{x}} \log \pi(\mathbf{x})$, which means that $\pi(\mathbf{x})$ is only needed to be known up to proportionality. Also note that (7) does not uniquely define the rates. If we have a set of rates, $\lambda(\mathbf{x}, \mathbf{v})$ that satisfy (7), then $\lambda(\mathbf{x}, \mathbf{v}) + \gamma(\mathbf{x}, \mathbf{v})$ will also satisfy (7) for any positive function $\gamma(\mathbf{x}, \mathbf{v})$ for which $\gamma(\mathbf{x}, \mathbf{v}) = \gamma(\mathbf{x}, F_{\mathbf{x}}(\mathbf{v}))$.

A natural choice of rates which satisfy (7) are those which are smallest. This will give $\lambda(\mathbf{x}, \mathbf{v}) = \max\{0, -\mathbf{v} \cdot \nabla_{\mathbf{x}} \log \pi(\mathbf{x})\}$. We will call these the *canonical rates.* Theoretical justification for the canonical rates when $d = 1$ is given in Bierkens and Duncan (2017), who show that the asymptotic variance of Monte Carlo estimators is minimised when using these rates.

3.2.1 Different continuous-time MCMC algorithms

We now describe a number of choices of flip operator, and the corresponding PDPs. We start with the limiting process we derived in Section 3.1, and then describe two continuous MCMC processes that have been recently proposed. These three choices all lead to identical PDPs for a one-dimensional target, but differ in terms of how they extend to higher dimensions. Each assume the target is defined on an unbounded domain; for extensions of these methods to bounded domains see Bierkens et al. (2017a). We will finish with some discussion of alternative schemes that are possible.

Pure Reflection and Refresh

The continuous-time limit we derived in Section 3.1 corresponds to $F_{\mathbf{x}}(\mathbf{v}) = -\mathbf{v}$, with the canonical rates. Such a process we call a pure reflection process. For a multi-dimensional target distribution, this process would be reducible, as it can only explore positions $\mathbf{x}$ that lie on a straight-line defined by the initial velocity. As such this is an example where $\pi(\mathbf{x})$ would be a marginal invariant distribution but not the marginal stationary distribution. To overcome this we would need an additional move, which refreshes $\mathbf{v}$. Such a refresh move would need to have $p_u(\mathbf{v})$ as its stationary distribution. The times of refreshing could be either deterministic or random.

Bouncy Particle Sampler

The Bouncy Particle Sampler of Bouchard-Côté et al. (2017), based on an algorithm of Peters and de With (2012), is an adaption of the pure reflection process, which minimises the change in velocity, $||F_{\mathbf{x}}(\mathbf{v}) - \mathbf{v}||$, at each event.

It does this by defining $F_{\mathbf{x}}(\mathbf{v})$ to be

$$F_{\mathbf{x}}(\mathbf{v}) = \mathbf{v} - 2\frac{\mathbf{v} \cdot \nabla \log \pi(\mathbf{x})}{\nabla \log \pi(\mathbf{x}) \cdot \nabla \log \pi(\mathbf{x})} \nabla \log \pi(\mathbf{x}). \tag{8}$$

This flips the component of $\mathbf{v}$ that is in the direction of $\nabla \log \pi(\mathbf{x})$ but leaves the components of $\mathbf{v}$ that are orthogonal to $\nabla \log \pi(\mathbf{x})$ unchanged. They again use the canonical rates. As with the pure reflection process this means that events only occur if the PDP is moving to areas of lower probability mass according to $\pi(\mathbf{x})$.

The original sampler of Peters and de With (2012) just simulates this PDP. However Bouchard-Côté et al. (2017) shows that, for some targets, such a sampler can be reducible. This means that, depending on how the process is initiated, there may be parts of the state-space that the Bouncy Particle Sampler cannot reach. As a result the invariant distribution, $\pi(\mathbf{x})$, of the PDP may not be its unique asymptotic distribution.

The Bouncy Particle Sampler introduces refresh events. Refresh events occur as events of an independent Poisson process of constant rate, and at a refresh event we simulate a new velocity from $p_v(\mathbf{v})$. Bouchard-Côté et al. (2017) show that for any non-zero rate of this refresh process, the resulting sampling will have $\pi(\mathbf{x})$ as its unique asymptotic distribution.

Zig-Zag Sampler

The Zig-Zag sampler (Bierkens et al., 2017b; Bierkens et al., 2016) considers a discrete set of velocities. If $\mathbf{x}$ is d-dimensional, then $\mathbf{v} = \sum_{i=1}^{d} \theta_i \mathbf{e}_i$, where each $\theta_i \in \{-1, 1\}$ and $\mathbf{e}_1, \ldots, \mathbf{e}_d$ are a set of orthogonal basis vectors for $\mathbb{R}^d$. The invariant distribution for $\mathbf{v}$ is defined as the uniform distribution over this set of 2^d possible values.

The Zig-Zag sampler can be viewed as having d-distinct event types, each with its own rate, and each with its own deterministic change to the velocity. The ith event will have a flip, $F^{(i)}$, that switches θ_i to $-\theta_i$, but keeps the velocity in the other $d-1$ directions unchanged. If we denote $\lambda_i(\mathbf{x}, \mathbf{v})$ to be the rate of events of type i, then this corresponds to our general formulation of a PDP but with $\lambda(\mathbf{x}, \mathbf{v}) = \sum_{i=1}^{d} \lambda_i(\mathbf{x}, \mathbf{v})$, and with the transition distribution at an event being a discrete distribution over the d transitions that correspond to the d different flips. Flip i occurs with probability $\lambda_i(\mathbf{x}, \mathbf{v})/\lambda(\mathbf{x}, \mathbf{v})$. Subsituting this into (6) shows that we need to choose $\lambda_i(\mathbf{x}, \mathbf{v})$ so that

$$\sum_{i=1}^{d} \left\{ \lambda_i(\mathbf{x}, \mathbf{v}) - \lambda_i(\mathbf{x}, F^{(i)}(\mathbf{v})) \right\} = -\sum_{i=1}^{d} \theta_i \frac{\partial \log \pi(\mathbf{x})}{\partial x^{(i)}}.$$

Here we have assumed that $x^{(i)}$ is the component of $\mathbf{x}$ in direction $\mathbf{e}_i$. This can be achieved if we choose the rates such that

$$\lambda_i(\mathbf{x}, \mathbf{v}) - \lambda_i(\mathbf{x}, F^{(i)}(\mathbf{v})) = -\theta_i \frac{\partial \log \pi(\mathbf{x})}{\partial \mathbf{x}^{(i)}}.$$

As above, this does not uniquely define the rates, only the difference between rates for velocities that differ in terms of their component in the $\mathbf{e}_i$ direction.

It is a challenging goal to show that the Zig-Zag process is ergodic, that is that its invariant distribution is also its unique asymptotic distribution, in full generality. So far it is established rigorously in Bierkens et al. (2016)

that the Zig-Zag process is ergodic in any of the following cases: (i) one-dimensional target distributions, (ii) factorized target distributions, and (iii) switching rates that are positive everywhere (which can be obtained by adding a constant $\varepsilon > 0$ to the canonical switching rates). Experiments suggest that ergodicity holds in much more generality.

Note that the above argument easily generalises to allowing velocities of the form $\mathbf{v} = \sum_{i=1}^{m} \theta_i \mathbf{e}_i$, where the $\mathbf{e}_i$ are not constrained to be orthogonal, and we can even allow $m > d$ directions. Whether there are advantages in using such a set of possible velocities is not clear.

Alternatives

There is substantial extra flexibility in choosing the event rates and the type of transition at events beyond the three examples we have detailed. For example we could consider transitions at an event that does not depend on the current velocity. If we allow $\mathbf{v}$ to be any unit vector, then it is straightforward to show that choosing $\lambda(\mathbf{x}, \mathbf{v}) = \max\{0, -\mathbf{v} \cdot \nabla_{\mathbf{x}} \log \pi(\mathbf{x})\}$, and, at each event, sampling a new velocity from the distribution

$$q(\mathbf{v}'|\mathbf{x}, \mathbf{v}) \propto \max\{0, \mathbf{v}' \cdot \nabla_{\mathbf{x}} \log \pi(\mathbf{x})\},$$

will lead to a PDP with invariant distribution that has $\pi(\mathbf{x})$ as its marginal.

More substantial alternatives are also possible. For example, we could consider processes which allow the invariant distribution of $\mathbf{V}$ to depend on $\mathbf{x}$ – something that Girolami and Calderhead (2011) has shown to be beneficial for Hamiltonian Monte Carlo methods. For a proposed distribution $\pi(\mathbf{v}|\mathbf{x})$ we would then need to find a set of event rates and transitions that satisfy (5).

3.3 Simulation and Use of Skeletons for Continuous MCMC

So far we have described a number of different PDPs that will have $\pi(\mathbf{x})$ as their marginal invariant distribution. For these to be useful in practice, we need to be able to simulate them efficiently. How to do this in practice will depend on the form of $\pi(\mathbf{x})$, but is likely to use the ideas briefly described at the end of Section 2.1. For further detail see the discussion of this, and suggestions, in Bouchard-Côté et al. (2017) and Bierkens et al. (2016).

The output of simulating a PDP will be a set of event times and the values of the state at those event times. We wish to use this output to obtain Monte Carlo estimates of expectations of functions of $\mathbf{X}$, where $\mathbf{X}$ is distributed according to $\pi(\mathbf{x})$. Assume we have simulated the PDP for some time-interval T. We will discard the value of the process in some burn-in period of length t_b. Assume there were N events in the time-interval $[t_b, T]$. Denote these as τ_i for $i = 1, \dots, N$, and let $\tau_0 = t_b$ and $\tau_{N+1} = T$.

There are two approaches to obtain a Monte Carlo estimate of $\int \pi(\mathbf{x}) g(\mathbf{x}) \mathrm{d}\mathbf{x}$ for some function $g(\mathbf{x})$ of interest.

The first is to calculate the average of this function along the path of the PDP:

$$\frac{1}{\tau_{N+1}-\tau_0}\sum_{i=0}^{N}\int_0^{\tau_{i+1}-\tau_i} g(\mathbf{x}_{\tau_i}+s\mathbf{v}_{\tau_i})\mathrm{d}s.$$

Here each integral corresponds to the integrals of $g(\mathbf{x}_t)$ for t in $[\tau_i,\tau_{i+1}]$, and uses the fact that for such a t, $\mathbf{x}_t=\mathbf{x}_{\tau_i}+(t-\tau_i)\mathbf{v}_{\tau_i}$.

The above approach is difficult if the integrals are not easy to evaluate. In this case we can resort to a standard Monte Carlo approximation. Choose an integer $M>0$, define $h=(\tau_{N+1}-\tau_0)/M$ and then use the Monte Carlo estimator

$$\frac{1}{M}\sum_{j=1}^{M} g(\mathbf{x}_{\tau_0+jh}),$$

where we can trivially calculate $\mathbf{x}_{\tau_0+jh}$ using the set of event times and the values of the PDP at those event times.

3.4 Example: Robust Regression

To demonstrate the difference between the Bouncy Particle Sampler and the Zig-Zag sampler, and to compare these methods with more traditional MCMC methods, we will consider their application to a robust regression model. We model the mean of each data point as a linear function of $d-1$ covariates and an intercept, but model the errors as a mixture of a standard normal random variable and a normal random variable with mean equal to 0 but a variance equal to 10^2. Appendix A gives details of the log-posterior for this model and how we can bound the event rate of, and hence simulate events for, either the Bouncy Particle or Zig-Zag sampler.

We first compare the dynamics of the Zig-Zag algorithm and the Bouncy Particle Sampler. To do this we consider the $d=2$ case, so that we have a bivariate target distribution whose contours we can plot. We simulated $n=500$ data points, with the covariates values being independent draws from a standard normal distribution. We simulated half the data points with parameter values $(2,1)$ and half with parameter values $(6,1)$, with the simulated residuals being from a standard normal distribution. This choice was made so as to produce a posterior distribution with multiple modes – corresponding to the intercept term being either 2, 6, or 4, the average of these values.

Example output from the Bouncy Particle Sampler and the Zig-Zag Algorithm are shown in Figure 1. We tried three implementations of the Bouncy Particle Sampler, each with substantially different rates at which we refresh the velocity. These show the potential pitfalls of choosing this rate either too low or too high. In the former case (top-left plot) either all (if the refresh rate is 0 as here) or almost all of the changes of velocity will be at bounce events. If the posterior has contours that are close to elliptical, as they are in the tails of our model, then this will produce dynamics with strong structure which can slow down mixing. We see this here with the PDP dynamics circling around in the tails of the posterior. As mentioned above, Bouchard-Côté et al. (2017) give examples

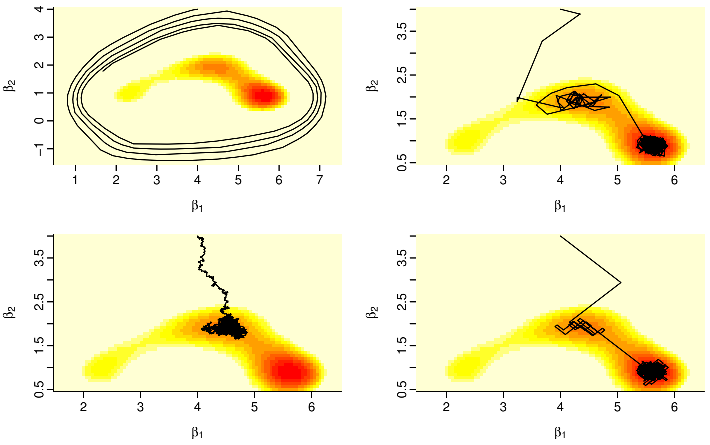

Figure 1: Plots of output from the Bouncy Particle Sampler (top-row and bottom-left) and Zig Zag algorithm (bottom-right) for the robust regression model with $d = 2$. In each case we plot the continuous-time output of the position component of each sampler on top of a heat-map of the posterior for the two parameters (which are denoted β_1, β_2). The Bouncy Particle sample output is for different refresh rates: no refresh event (top-left) refresh rate 1 (top right) and refresh rate 100 (bottom left).

where setting the refresh rate to 0 leads to a sampler which is reducible, and, depending on the initial conditions, will not be able to reach some parts of the state-space. By comparison if we use a refresh-rate that is too high (bottom-left) then the resulting process resembles a reversible MCMC algorithm and thus loses the potential advantages of non-reversible dynamics that we obtain with a more reasonable choice of refresh rate (top-right plot). Notice that the Zig-Zag sampler's dynamics (bottom-right) are qualitatively similar to the Bouncy Particle Sampler's dynamics when a reasonable refresh rate is chosen. Though the Zig-Zag sampler is restricted to move in certain directions, with the resulting "zig-zag" nature of the output giving the algorithm its name.

We now compare the Bouncy Particle Sampler to a Metropolis adjusted Langevin Algorithm (MALA, Roberts and Rosenthal, 1998), and consider how the two methods compare in both a low-dimensional, $d = 8$, and high-dimensional, $d = 128$, setting. We simulated the covariates for each observation from an AR(1) process with lag-1 correlation of 0.5. In both cases we set all co-efficients in the linear model to 0 except for those associated with the intercept and first covariate. We simulated 500 observations, 300 from a model with $(\beta_1, \beta_2) = (2, 1)$ and 200 from a model with $(\beta_1, \beta_2) = (6, 1)$, and with standard normal residuals in each case. This produces a complex log-posterior whilst ensuring that the posterior has a single main mode, which means that auto-correlation summaries of MCMC output are more reliable for estimating the efficiency of the algorithm. We tuned the MALA algorithm to have an acceptance probability close to 0.5 (Roberts and Rosenthal, 1998), and run both MALA and the Bouncy Particle Sampler so that they each had 50,000 iterations (where an iteration for the Bouncy Particle Sampler corresponds to a proposed event-time). The resulting samplers had similar computational costs, with MALA taking slightly longer. For each sampler we removed the first 40% of the output as burn-in. For the Bouncy Particle Sampler we then sampled 30,000 values of the parameters at equally spaced time-points, so that both algorithms gave an identical form of output.

Auto-correlation plots for for the intercept parameter are shown in Figure 2. Both samplers mix quickly for

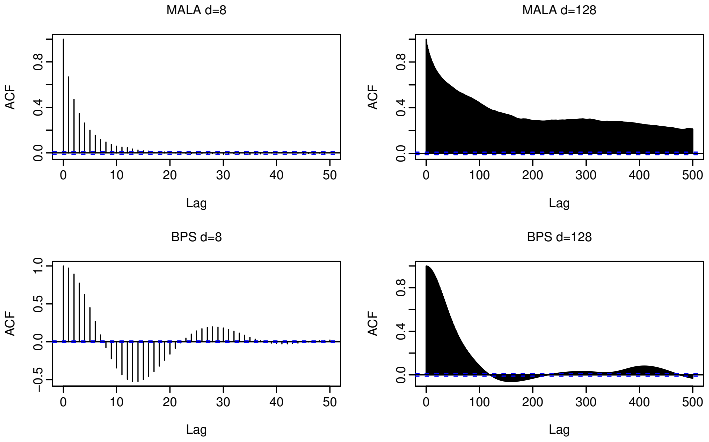

Figure 2: Auto-correlation plot for the intercept parameter from a run of MALA (top-row) and the Bouncy Particle Sampler (bottom-row), for $d = 8$ (left-column) and $d = 128$ (right-column).

the low-dimesional case (see left-hand column). However the Bouncy Particle Sampler shows negative auto-correlation. We believe this is caused by the sampler's dynamics which tends to move from one tail of the posterior to the other (behaviour that is particularly pronounced for 1-dimensional unimodal target distributions; see Bierkens and Duncan, 2017). As a result of this negative correlation, estimates of the auto-correlation time for the Bouncy Particle Sampler are slightly small than for MALA. However, as the dimension increases, we tend to see bigger advantages from using the Bouncy Particle Sampler – perhaps due to its non-reversible dynamics. This can be seen from the auto-correlation plots for $d = 128$ (see right-hand column). For this run of the two algorithms the estimated auto-correlation times are approximately 670 for MALA and 130 for the Bouncy Particle Sampler, suggesting a 5-fold gain in efficiency from using the latter algorithm. Key to the strong performance of the Bouncy Particle Sampler for this example is the fact that we can efficienctly simulate the event times using thinning – with the method described in Appendix A; with around 30% of proposed event times being accepted.

3.5 Exact Approximation versions and Subsampling

Exact approximate algorithms (Andrieu and Roberts, 2009) are MCMC algorithms that use estimators of the target distribution within the accept-reject step. If implemented correctly, and if these estimators are both positive and unbiased, then it can be shown that the resulting MCMC algorithms are exact: in the sense they still have the target distribution as their stationary distribution. It turns out that exact approximate versions of the continuous-time MCMC algorithms detailed in the previous section are also possible.

3.5.1 Exact Approximation for Pure Reflection and Zig Zag

For concreteness and ease of presentation we will consider an exact approximate version of the Pure Reflection process. Though the ideas we detail extend trivially to the Zig-Zag sampler (and see Bierkens et al., 2016, for more details of an exact approximate version of Zig-Zag).

For the Pure Reflection process the requirement on the rates of events is that for any velocity $\mathbf{v}$

$$\lambda(\mathbf{x},\mathbf{v})-\lambda(\mathbf{x},-\mathbf{v})=-\mathbf{v}\cdot\nabla\log\pi(\mathbf{x}).$$

For a given choice of rates, such as the canonical rates $\lambda(\mathbf{x},\mathbf{v})=\max\{0,-\mathbf{v}\cdot\nabla\log\pi(\mathbf{x})\}$, we would often use thinning to simulate the event times (see Section 2.1). Thus if our current state is $(\mathbf{x}_t,\mathbf{v}_t)$ we would introduce a bound on the event rate for $s>0$

$$\tilde{\lambda}^{+}(s)\geq\lambda(\mathbf{x}_t+s\mathbf{v}_t,\mathbf{v}_t),$$

simulate potential events at rate $\tilde{\lambda}^{+}(s)$ and accept them with probability $\lambda(\mathbf{x}+s\mathbf{v},\mathbf{v})/\tilde{\lambda}^{+}(s)$. The time until the next event is just the time until the first accepted event.

Now assume we have a estimator of $-\nabla\log\pi(\mathbf{x})$, which we will denote $\mathbf{U}(\mathbf{x})$. This estimator is a random variable, and examples of how it could be constructed are given below. We further introduce a random rate function

$$\hat{\lambda}(\mathbf{x},\mathbf{v})=\max\{0,\mathbf{v}\cdot\mathbf{U}(\mathbf{x})\}.$$

This is just the canonical event rate, but replacing $-\nabla\log\pi(\mathbf{x})$ with its unbiased estimator. The idea of an exact-approximate version is to simulate events using thinning, with a bound on the event rate that satisfies

$$\tilde{\lambda}^{+}(s)\geq\hat{\lambda}(\mathbf{x}_t+s\mathbf{v}_t,\mathbf{v}_t)$$

almost surely, and where we accept points using the random acceptance probability $\hat{\lambda}(\mathbf{x}_t+s\mathbf{v}_t,\mathbf{v}_t)/\tilde{\lambda}^{+}(s)$.

As the overall acceptance probability will be the expectation of the random acceptance probability, it is straightforward to show that simulating events in this way is equivalent to simulating events at a rate

$$\lambda(\mathbf{x},\mathbf{v})=\mathrm{E}\left(\max\left\{0,\mathbf{v}\cdot\mathbf{U}(\mathbf{x})\right\}\right),\tag{9}$$

where expectation is with respect to the random variable $\mathbf{U}(\mathbf{x})$. Furthermore if we now calculate the difference in rates, $\lambda(\mathbf{x},\mathbf{v})-\lambda(\mathbf{x},-\mathbf{v})$, we have

$$\begin{aligned}\lambda(\mathbf{x},\mathbf{v})-\lambda(\mathbf{x},-\mathbf{v}) &= \mathrm{E}\left(\max\left\{0,\mathbf{v}\cdot\mathbf{U}(\mathbf{x})\right\}\right)-\mathrm{E}\left(\max\left\{0,-\mathbf{v}\cdot\mathbf{U}(\mathbf{x})\right\}\right)\\ &= \mathrm{E}\left(\max\left\{0,\mathbf{v}\cdot\mathbf{U}(\mathbf{x})\right\}-\max\left\{0,-\mathbf{v}\cdot\mathbf{U}(\mathbf{x})\right\}\right)\\ &= \mathrm{E}\left(\max\left\{0,\mathbf{v}\cdot\mathbf{U}(\mathbf{x})\right\}+\min\left\{0,\mathbf{v}\cdot\mathbf{U}(\mathbf{x})\right\}\right)=\mathrm{E}\left(\mathbf{v}\cdot\mathbf{U}(\mathbf{x})\right).\end{aligned}$$

Thus provided $\mathbf{U}(\mathbf{x})$ is an unbiased estimator of $-\nabla\log\pi(\mathbf{x})$, the resulting process will have $\pi(\mathbf{x})$ as its marginal invariant distribution.

Note that using an unbiased estimator of $-\nabla\log\pi(\mathbf{x})$ does not come without cost, as the resulting process will, in general, be less efficient. This loss of efficiency comes first as the rates that are used for events, $\mathrm{E}(\max\{0, \mathbf{v}\cdot\mathbf{U}(\mathbf{x})\})$, will, in general, be larger than the canonical rates. The only exception being if $\mathbf{U}(\mathbf{x})$ has the same sign as $-\nabla \log \pi(\mathbf{x})$ with probability one. Using larger rates appears to reduce the rate of mixing of the process (see Bierkens and Duncan, 2017; Bierkens et al., 2016). A related issue is that the bound on the rates, used when implementing thinning, will also tend to be larger. This will increase the cost of simulating the process. Intuitively we would expect both these losses of efficiency to increase as the variability of our estimator increases.

3.5.2 Exact Approximation for the Bouncy Particle Sampler

The idea of an exact approximation version of the Bouncy Particle Sampler is slightly more complicated, due to the fact that the flip operator used also depends on $\nabla \log \pi(\mathbf{x})$. To implement such an exact approximation version we need to use the same estimate of $\nabla \log \pi(\mathbf{x})$ in the flip operator as was used in deciding to accept the event. So, using the notation of the previous section, at a potential event at time s, simulated from a Poisson process with rate $\tilde{\lambda}^{+}(s)$ we will now:

(1) Simulate $\mathbf{u}$, a realisation of $\mathbf{U}(\mathbf{x}_t + s\mathbf{v}_t)$.

(2) Accept the event with probability

$$\frac{\hat{\lambda}(\mathbf{x}_t + s\mathbf{v}_t, \mathbf{v}_t)}{\tilde{\lambda}^{+}(s)} = \frac{\max\{0, \mathbf{u}\cdot\mathbf{v}_t\}}{\tilde{\lambda}^{+}(s)}$$

(3) If we accept the potential event, this corresponds to an event at time $t+s$, with new state being $\mathbf{x}_{t+s} = \mathbf{x}_t + s\mathbf{v}_t$ and

$$\mathbf{v}_{t+s} = \mathbf{v}_t - 2\frac{\mathbf{v}_t\cdot\mathbf{u}}{\mathbf{u}\cdot\mathbf{u}}\mathbf{u}.$$

Note that the same realisation, $\mathbf{u}$, is used in both steps (2) and (3).

This leads to an algorithm where the transition at an event is random. We will assume that $\mathbf{U}$ is a discrete random variable, which is consistent with the use of sub-sampling discussed in Section 3.5.3. It is straightforward to show that the resulting PDP has events occurring at rate (9) as before, but with a transition probability mass function at an event that is

$$q(\mathbf{v}'|\mathbf{x}, \mathbf{v}) = p(\mathbf{u}|\mathbf{x})\frac{\max\{0, \mathbf{u}\cdot\mathbf{v}\}}{\mathrm{E}\left(\max\left\{0, \mathbf{v}\cdot\mathbf{U}(\mathbf{x})\right\}\right)}$$

where $p(\mathbf{u}|\mathbf{x})$ is the probability mass of simulating $\mathbf{u}$, and $\mathbf{v}' = \mathbf{v} - 2\mathbf{u}(\mathbf{v}\cdot\mathbf{u})/(\mathbf{u}\cdot\mathbf{u})$. Now substituting these values into the equation for the stationary distribution of the PDP (5), it is simple to show that the resulting process will have an invariant distribution $\pi(\mathbf{x})p_v(\mathbf{v})$, where $p_v(\mathbf{v})$ is uniform on the set of velocities of fixed speed.

3.5.3 Use of Subsampling

An example of an exact-approximate version of these continuous-time MCMC algorithms arises if we use subsampling of data points at each iteration when performing Bayesian inference in a big-data setting. This was first suggested for the Zig Zag algorithm by Bierkens et al. (2016). Bouchard-Côté et al. (2017) has shown that sub-sampling can be used within the Bouncy Particle Sampler (see also Pakman et al., 2017), though they derive this as a special case of what they call the local Bouncy Particle Sampler. This local Bouncy Particle Sampler also allows efficient implementation when the target can be written as a product of factors, each of which only depends on a few components of $\mathbf{x}$.

We will consider a target density of the form

$$\pi(\mathbf{x}) \propto \prod_{i=1}^{n} \pi_i(\mathbf{x}).$$

In this case, the simplest unbiased estimator of $-\nabla \log \pi(\mathbf{x})$ is just

$$-n\nabla \log \pi_I(\mathbf{x}), \tag{10}$$

where I is drawn uniformly from $1, \dots, n$. However, to increase the efficiency of sub-sampling methods we would want to try and minimise the variance of our estimator, for example by using control variates. One approach that is commoly used (e.g. Bardenet et al., 2017; Dubey et al., 2016; Baker et al., 2017) is to have a pre-processing step that finds $\hat{\mathbf{x}}$, a value of $\mathbf{x}$ that is near the posterior mode. We then calculate and store $\nabla \log \pi(\hat{\mathbf{x}})$, and use the following estimator

$$-\nabla \log \pi(\hat{\mathbf{x}}) + n\left(\nabla \log \pi_I(\hat{\mathbf{x}}) - \nabla \log \pi_I(\mathbf{x})\right). \tag{11}$$

If $\mathbf{x}$ is within a distance of $O(1/\sqrt{n})$ of $\hat{\mathbf{x}}$ and if $\pi(\mathbf{x})$ is sufficiently smooth we would expect the variance of this estimator to only increase linearly with n. By comparison the variance of the simple estimator (10) will increase like $O(n^2)$.

3.6 Example: Mixture Model

To demonstrate some of the properties of the use of sub-sampling within continuous-time MCMC algorithms we apply them to a simple mixture model. Assume we have IID data from a mixture distribution, where for $j = 1 \dots, n$

$$Y_j \sim \begin{cases} \mathrm{N}(0, 10^2) & \text{with probability } p, \\ \mathrm{N}(x, 1^2) & \text{otherwise.} \end{cases}$$

Our interest is inference for x, and we assume a Gaussian prior with mean 0 and variance 4. In the following we simulate data with $x = 4$.

This involves inference for a univariate parameter, and for this case each of the three versions of continuous-time MCMC introduced earlier are equivalent. Furthermore we do not need to introduce any refreshing of the velocity, as used within the Bouncy Particle Sampler, to ensure ergodicity (Bierkens and Duncan, 2017). Our aim is to show how the continuous-time MCMC algorithms can be implemented, how the choice of the bounding process that simulates potential event times can affect efficiency, and give some insight into how and when the subsampling ideas can lead to gains in efficiency.

We will first look at implementing continuous-time MCMC using a global bound on the event rates. For this model, if we write $\pi(x) \propto \prod_{i=1}^{n} \pi_i(x)$, where π_i is the likelihood of the ith observation times the $1/n$th root of the prior, then

$$\log \pi_i(x) = \log\left(\frac{p}{10}\exp\left\{-\frac{1}{200}y_i^2\right\} + (1-p)\exp\left\{-\frac{1}{2}(x-y_i)^2\right\}\right) - \frac{1}{8}x^2.$$

We will fix $p = 0.95$ in the simulations we present. We can bound $|\nabla \log \pi_i(x)|$ for each i, and this bound increases with $|y_i|$. If we let j be the observation with largest absolute value, then the simplest global bound on the event rates will be

$$\lambda^+ = n \max_x |\nabla \log \pi_j(x)|, \tag{12}$$

which we can calculate numerically.

We can then simulate the path of our continous-time MCMC algorithm by iterating the following steps. Assuming we are currently at time t with state (x_t, v_t):

(1) Simulate the time until the next putative event, s, a realisation of an exponential distribution with rate λ^+.

(2) Calculate $x_{t+s} = x_t + sv_t$, and the actual rate of an event at position x_{t+s}:

$$\lambda(x_{t+s}, v_t) = \max\{0, -v_t \nabla \log \pi(x_{t+s})\}.$$

(3) With probability $\lambda(x_{t+s}, v_t)/\lambda^+$ switch the sign of the velocity, $v_{t+s} = -v_t$ and store the value (x_{t+s}, v_{t+s}). Otherise $v_{t+s} = v_t$.

This simulates using the canonical rates. To use the simplest version of sub-sampling we just replace the calculation of the actual rate in step (2) by

$$\lambda(x_{s+t}, v_t) = n \max\{0, -v_t \nabla \log \pi_I(x_{s+t})\},$$

for I sampled uniformly from $\{1, \dots, n\}$. Note that our choice of λ^+ can still be used with sub-sampling, as it bounds the above rate for all I and x_{s+t}.

The above algorithm has similarities to one iteration of a standard MCMC algorithm. Steps (1) and (2) can be viewed in terms of simulating a new state, and step (3) is a form of accept-reject step. However there are fundamental differences. Firstly, the probability in step (3) depends on the target through $\nabla \log \pi(x)$, as compared to the acceptance probability in MCMC which depend on $\pi(x)$. Secondly, the algorithm moves from x_s to x_{s+t} regardless of the outcome in step (3). Step (3) only affects the velocity component. Finally, as mentioned in Section 3.3 the use of the output is different. For continuous-time MCMC we have to take averages with respect to the continuous-time path, or with respect the value of the process at equally-spaced time-points. By comparison MCMC would average with respect to the value of the chain at the end of each iteration.

We now turn to how the use of sub-sampling impacts on the efficiency of continuous-time MCMC. For our above implementation the average number of iterations needed to simulate a path over a time-interval of length T will be T/λ^+ for both the canonical and sub-sampling versions. Sub-sampling will involve a smaller cost in step (2) as the rate depends on just a single data point rather than all n data point. However this computational saving comes at the cost of an overall increased rate of switching velocity. This is shown in Figure 3, where we give examples of $\nabla \log \pi(x)$ for two simulated data sets, of size $n = 150$ and $n = 1,500$ respectively, and each simulated with the true value of $x = 4$. We also show the canonical rate of switching from a negative to a positive velocity, and the expected rate of switching when we use sub-sampling. The canonical implementation has uniformly lower rates.

The impact of these different rates can be seen in Figure 4, where we show trace autocorrelation plots for analysing the data set with $n = 150$. Using subsampling leads to paths of the sampler that switch velocity substantially more frequently. As a result, the canonical implementation is more efficient in terms of suppressing random walk behaviour, and this is seen in terms of better mixing and lower autocorrelation. The autocorrelation plots suggest we need to run the sub-sampling version for roughly 5 times as long to obtain the same accuracy as using the canonical implementation.

We can improve the computational efficiency of both these implementations of continuous-time MCMC through using a lower bounding rate, λ^+. The possibility for lowering λ^+ is greater, however, for the canonical version, and this is a second advantage it has over using sub-sampling. For example, with an additional pre-processing cost, we could choose

$$\lambda^+ = \sum_{i=1}^{n} \max_x |\nabla \log \pi_i(x)|. \tag{13}$$

Such a choice can be used with sub-sampling if our estimate of the rate uses non-uniform sampling of data points:

$$\lambda(x, v) = \lambda^+ \max\left\{0, -v\frac{\nabla \log \pi_I(x)}{\max_x |\nabla \log \pi_I(x)|}\right\},$$

where we sample I from $1, \ldots, n$, with value i having probability proportionally to $\max_x |\nabla \log \pi_i(x)|$. For the canonical implementation we can reduce the rate further if we are able to use the actual maximum of $|\nabla \log \pi(x)|$, but such a choice is not valid with sub-sampling. For our example, using (13) rather than (12) will reduce the number of iterations required by a factor of 5.3. If we used the actual maximum of $|\nabla \log \pi(x)|$ for the canonical version, we would reduce the number of iterations by a factor of nearly 30 when compared to using (12). Note

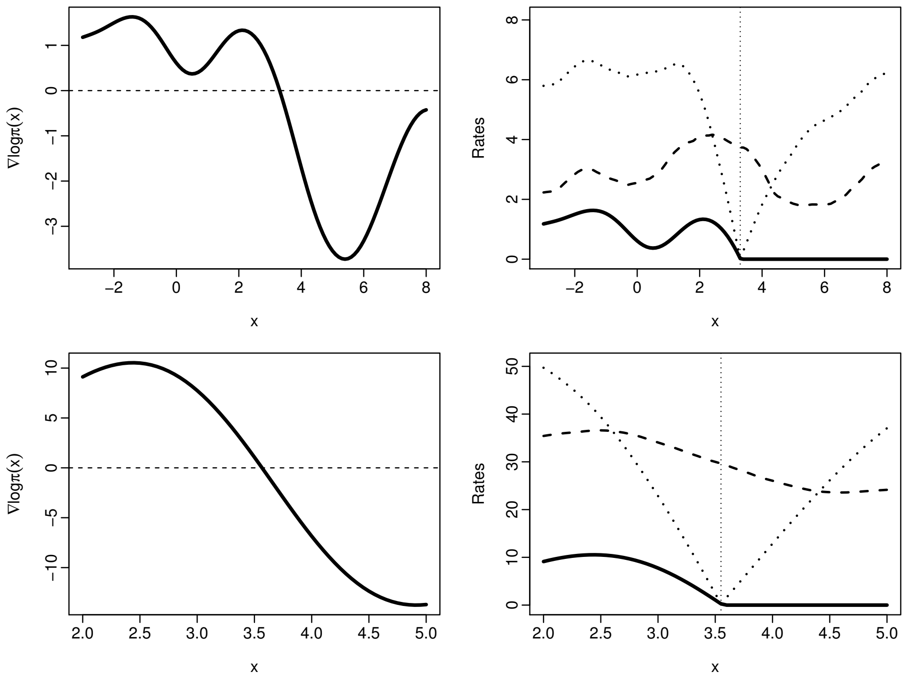

Figure 3: Plots of $\nabla \log \pi(x)$ for the mixture example (left-hand column), and rates at which the continuous-time MCMC algorithm will switch from a negative to a positive velocity (right-hand column). For the latter plots we show rates for the canonical process (full lines), simple sub-sampling (dashed lines) and sub-sampling with control variates (dotted lines). The vertical dotted line shows the value of $\hat{x}$. Top row is for 150 data points, and the bottom row for 1,500 data points. Plots are restricted to areas of non-negligible posterior mass.

that for all these options for choosing λ^+, the underlying stochastic process we are simulating is unchanged – it is just the efficiency of the simulation algorithm that is affected.

If we compare the best implementation of continous-time MCMC with subsampling, using (13), to the best version of the canonical implementation we get that for the same accuracy the we would need just over 25 times as many iterations using subsampling. Each iteration would be quicker, however, as it would need access to just one, out of 150, data-points.

To see any substantial gains from using subsampling, we need to have a lower variance estimator of $\nabla \log \pi(x)$, using, for example, control variates (11). To implement this we need to upper bound our estimator of the rate. This is possible for this application as the absoluate value of the second derivate of $\log \pi_i(x)$ is bounded. Assume we can find a bound, C say, then we use a bounding rate

$$\lambda^+(x) = |\nabla \log \pi(\hat{x})| + nC|x - \hat{x}|. \tag{14}$$

To implement the resulting algorithm we can again iterate the three steps given above. The only changes are that in step (1) we need to simulate the inter-event time from a point process with rate

$$\tilde{\lambda}^+(s) = |\nabla \log \pi(\hat{x})| + nC|x_t + v_t s - \hat{x}|,$$

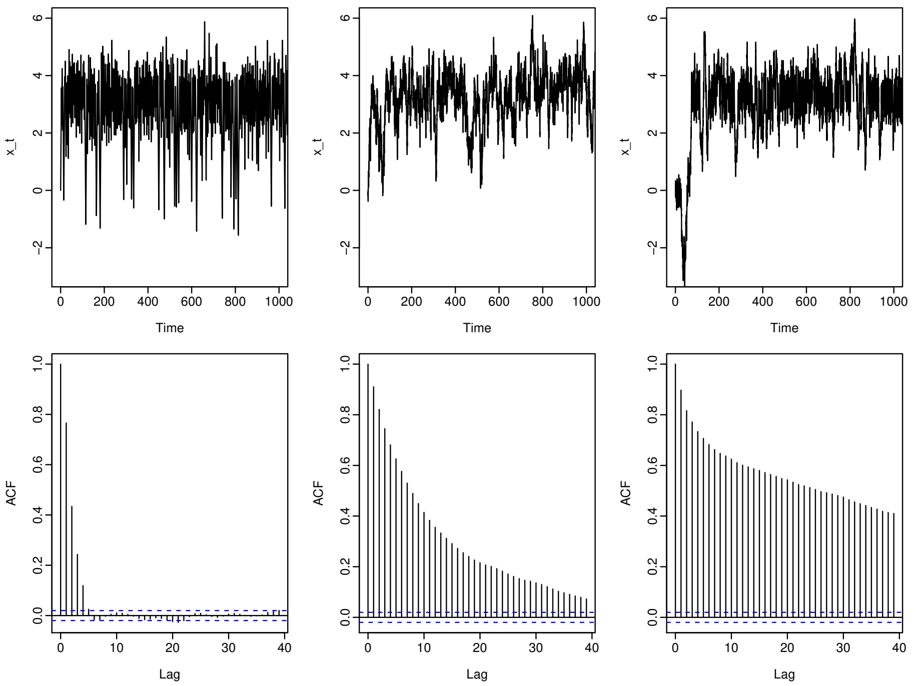

Figure 4: Trace plots (top row) and autocorrelation plots (bottom row) for three implementations of continuous-time MCMC: canonical process (left-hand column); simple sub-sampling (middle column) and subsampling with control variates (right-hand column). Auto correlation plots are values sampled every unit time-step from the continuous sample paths.

and in step (3) the probability of switching the velocity is

$$\frac{\max\{0, -v[\nabla \log \pi(\hat{x}) + n(\nabla \log \pi_I(x_{t+s}) - \nabla \log \pi_I(\hat{x}))]\}}{|\nabla \log \pi(\hat{x})| + nC|x_{t+s} - \hat{x}|}.$$

We can get some insight into the advantage of using control variates by calculating the expected rate of switching the velocity for the resulting algorithm and comparing with this rate for the other two implementations. This comparison is shown in Figure 3. We see a much lower rate of switching when we use control variates if x is close to $\hat{x}$ as compared to the simple sub-sampling approach. However the rate is actually larger if x is far from $\hat{x}$. Thus we see the importance of $\hat{x}$ being close to the mode of the posterior. This picture is the same for both small, $n = 150$, and larger, $n = 1,500$, data sets. However for larger data sets the posterior mass close to the posterior mode increases. As such the amount of time that algorithm will be in regions where using control variates is better will increase as we analyse larger data sets.

In Figure 4 we see output from the algorithm using control variates for $n = 150$. For such a small sample size, there appears to be little advantage in using control variates. The mixing in the tails is poor, due to the large variability of our esimators of the switching rate when x is not close to $\hat{x}$. Note that we can avoid this issue by using a hybrid scheme that estimates the rates using control variates when $|x - \hat{x}|$ is small, and uses simple sub-sampling otherwise.

We see advantages from using control variates as we analyse larger data sets. A comparison of our three implementations of continuous-time MCMC is shown in Table 1, where we look at their computational cost per effective sample size (ESS), a standard measure of MCMC performance. Firstly note that for the canonical implementation, the amount of PDP time, t, we need to run the continuous-time MCMC for decreases with sample size. This is as described in the scaling limits discussed in Bierkens et al. (2016). The intuition is that for larger n the posterior is more concentrated, and thus the underlying PDP process needs less time to explore the posterior. This property is also seen if we use subsampling with control variates. Without control variates, the actual switching rates of the underlying PDP increase quickly with n which slows down the mixing of the algorithm, and the amount of time, t, that we need to simulate the underlying PDP for does not change much with n.

The computational cost of each algorithm also depends on the number of iterations, that is the number of proposed event-times, of the algorithm per t; and the cost of each iteration. The former increases with n for all implementations. Overall, the number of iterations per ESS remains roughly constant when we use control variates. As the computation cost per iterations is $O(1)$ we see evidence that this algorithm has a computational cost that does not increase with n. By comparison the number of iterations per ESS appears to increase roughly linearly with n if we use subsampling without control variates. (See Bouchard-Côté et al., 2017, for further empirical evidence of these scalings when we use sub-sampling with or without control variates).

For the canonical implementation, even using the best possible global bound on the event rate, we have the number of iterations per ESS remaining constant but the computational cost per iteration is $O(n)$. Thus its overall computational cost will increase linearly with n. We see some evidence of these scalings if we look at the number of gradient evaluations associated with each observation that needs to be calculated per ESS. In situations where these gradients are expensive this would be a good proxy for the overall computational cost – and these results suggest using sub-sampling with control variates will be particularly useful for models where this is the case.

4 Continuous-Time Sequential Importance Sampling

We now consider continuous-time versions of sequential importance sampling. Such algorithms were first developed to solve the problem of simulating from a diffusion (see Oksendahl, 2007, for an introduction to diffusions). In this situation we have a diffusion process, $\mathbf{X}_t$, defined as the solution to an SDE

$$\mathrm{d}\mathbf{X}_t = \mathbf{b}(\mathbf{X}_t)\mathrm{d}t + \sigma(\mathbf{X}_t)\mathrm{d}\mathbf{B}_t,$$

where $\mathbf{b}(\mathbf{x})$ is the d-dimensional drift, $\mathbf{B}_t$ is d dimensional Brownian motion, and $\sigma(\mathbf{x})$ is a d by d matrix that defines the instantaneous variance of the process. We have an initial distribution $p_0(\mathbf{x})$ for the diffusion, and wish to sample from the distribution of the process at some future time or times. If we denote the density of this process at time t, by $p_t(\mathbf{x})$, then the challenge is to sample from $p_t(\mathbf{x})$ for diffusion processes where we cannot write down what $p_t(\mathbf{x})$ is.

		Canonical	Canonical	Subsampling	Control Variate
	Bounding Rate	(13)	$\max_x \lvert\nabla \log \pi(x)\rvert$	(13)	(14)
$n = 150$	t per ESS	3.3	3.3	14	22
	Iterations per t	57	11	57	210
	Iterations per ESS	190	36	800	4,600
	Observation-gradient evaluations per ESS	28,000	5,400	800	4,600
$n = 1,500$	t per ESS	0.43	0.43	8.6	2.1
	Iterations per t	570	15	570	1,000
	Iterations per ESS	245	6.4	4,900	2,100
	Observation-gradient evaluations per ESS	370,000	9,600	4,900	2,100
$n = 15,000$	t per ESS	0.13	0.13	9.1	0.91
	Iterations per t	5,600	100	5,600	3,800
	Iterations per ESS	730	13	51,000	3,500
	Observation-gradient evaluations per ESS	11,000,000	195,000	51,000	3,500

Table 1: Comparison of different implementations of continuous-time MCMC: canonical, subsampling, and subsampling with control variates; and how they vary as sample size, n, increases. Both canonical and subsampling use a global bound on the event rate to simulate possible events, we give results for canonical using both (13) and $\max_x |\nabla \log \pi(x)|$ as this bound. We give estimates of both the stochastic process time-length, t, that the MCMC algorithm needs to be run per effective sample size (ESS); and the average number of iterations (proposed event times) per t. The product of these is then the number of iterations needed per ESS. The subsampling and control variate versions require calculating the gradient associated with a single data point per iteration, whereas standard implementation requires n such evaluations; for each n we also give the average number of such evaluations per ESS.

The Exact Algorithm (Beskos and Roberts, 2005; Beskos et al., 2006) and its variants (Beskos et al., 2008; Pollock et al., 2016b) have given a number of algorithms for simulating from such a diffusion process, but only under strong conditions on the drift and instantaneous variance. For example it is commonly required that $\sigma(\mathbf{x})$ is constant, and that the drift can be written as the gradient of some potential function. Almost all uni-variate diffusion processes can be transformed to satisfy these requirements, but few multivariate diffusion processes can.

Whilst we do not know what $p_t(\mathbf{x})$ is for any $t > 0$, we do know that it solves the Fokker-Planck equation for the diffusion

$$\frac{\partial p_t(\mathbf{x})}{\partial t} = -\sum_{i=1}^{d} \frac{\partial b_i(\mathbf{x}) p_t(\mathbf{x})}{\partial x_i} + \frac{1}{2} \sum_{i=1}^{d} \sum_{j=1}^{d} \frac{\partial^2 \Sigma_{ij}(\mathbf{x}) p_t(\mathbf{x})}{\partial x_i x_j},$$

where $\Sigma = \sigma^T \sigma$. This motivates the following question: can we use our knowledge of the Fokker-Planck equation for the process of interest in order to develop a valid importance sampling algorithm to sample from $p_t(\mathbf{x})$?

The continuous-time importance sampling (CIS) procedure of Fearnhead et al. (2016), which we describe below, will in fact enable us to do so. We will present it in a slightly more general form, in that we will use CIS to sample from a distribution $p_t(\mathbf{x})$ that is the solution to a partial differential equation

$$\frac{\partial p_t(\mathbf{x})}{\partial t} = \mathcal{L}^* p_t(\mathbf{x})$$

for some known operator $\mathcal{L}^*$ and subject to a known initial condition $p_0(\mathbf{x})$. This then allows for sampling from more general continuous-time Markov processes, where $\mathcal{L}^*$ would be the adjoint of the generator for that process. In the following we assume that this partial differential equation has a solution, and the formal justification of the CIS algorithm requires that this solution is unique.

4.1 The CIS Algorithm

The idea of CIS is similar in spirit to a standard importance sampler. We will choose a tractable proposal process, for the problem of sampling from a diffusion this is most naturally chosen to be Brownian motion. This proposal process must have a know transition density which is simple to sample from. We will simulate paths from this proposal process up to time t, and then construct an importance sampling weight. The challenge is that we need to calculate an importance sampling weight without knowing $p_t(\mathbf{x})$. The property we want from our importance sampler is that if we simulate a value and weight at time t, $(\mathbf{X}, W)$, then for suitable functions $f(\mathbf{x})$ we will have that $Wf(\mathbf{X})$ will be an unbiased estimator of the expectation of $f(\mathbf{X}_t)$ for our target process,

$$\mathrm{E}(f(\mathbf{X}_t)) = \int f(\mathbf{x})p_t(\mathbf{x})\mathrm{d}\mathbf{x}.$$

The original CIS algorithm can be derived by taking a limit of a discrete-time sequential importance sampler (Fearnhead et al., 2016). Below we give the general form of the resulting CIS process. The key to making this a valid importance sampler is choosing the incremental weight (see step 2 below) appropriately. We will show how we can derive the form of the incremental weight by viewing the CIS process as a PDP, and using the generator of the PDP to calculate expectations with respect to the CIS process.

Denote the transition density of the proposal process over time interval s as $q_s(x|y)$, and assume we have chosen an event-rate $\tilde{\lambda}(s) > 0$ for $s > 0$. The CIS algorithm is of the form:

(0) Set $\tau = 0$, $W_0 = 1$ and simulate $\mathbf{X}_0$ from the initial distribution of the target process, $p_0(\mathbf{x})$.

(1) Simulate a new event time $\tau' > \tau$, with the inter-event time $s = \tau' - \tau$ being drawn from a Poisson process with rate $\tilde{\lambda}(s)$.

(2) If $\tau' > t$ then simulate $\mathbf{X}_t$ from $q_{t-\tau}(\cdot|\mathbf{X}_{\tau'})$, and set $W_t = W_{\tau'}$. Otherwise simulate $\mathbf{X}_{\tau'}$ from $q_s(\cdot|\mathbf{X}_\tau)$ and update the weight. The update of the weight will take the form

$$W_{\tau'} = W_\tau \left[1 + \frac{\rho(\mathbf{X}_{\tau'}, \mathbf{X}_\tau, s)}{\tilde{\lambda}(s)}\right], \tag{15}$$

where ρ is a function that will be derived in Section 4.1.2. If $\tau' < t$, set $\tau = \tau'$ and go to (1).

Step (0) is just an initialisation step. The idea of the algorithm is that we use random event times, simulated in step (1), at which we evaluate the proposal process. Based on the value of the process at both this event and the preceeding event we then update the importance sampling weight. As is standard in sequential importance sampling, the new weight (calculated in step 2) is just the old weight multiplied by an incremental weight. Figure 5 gives an example of the output of this algorithm.

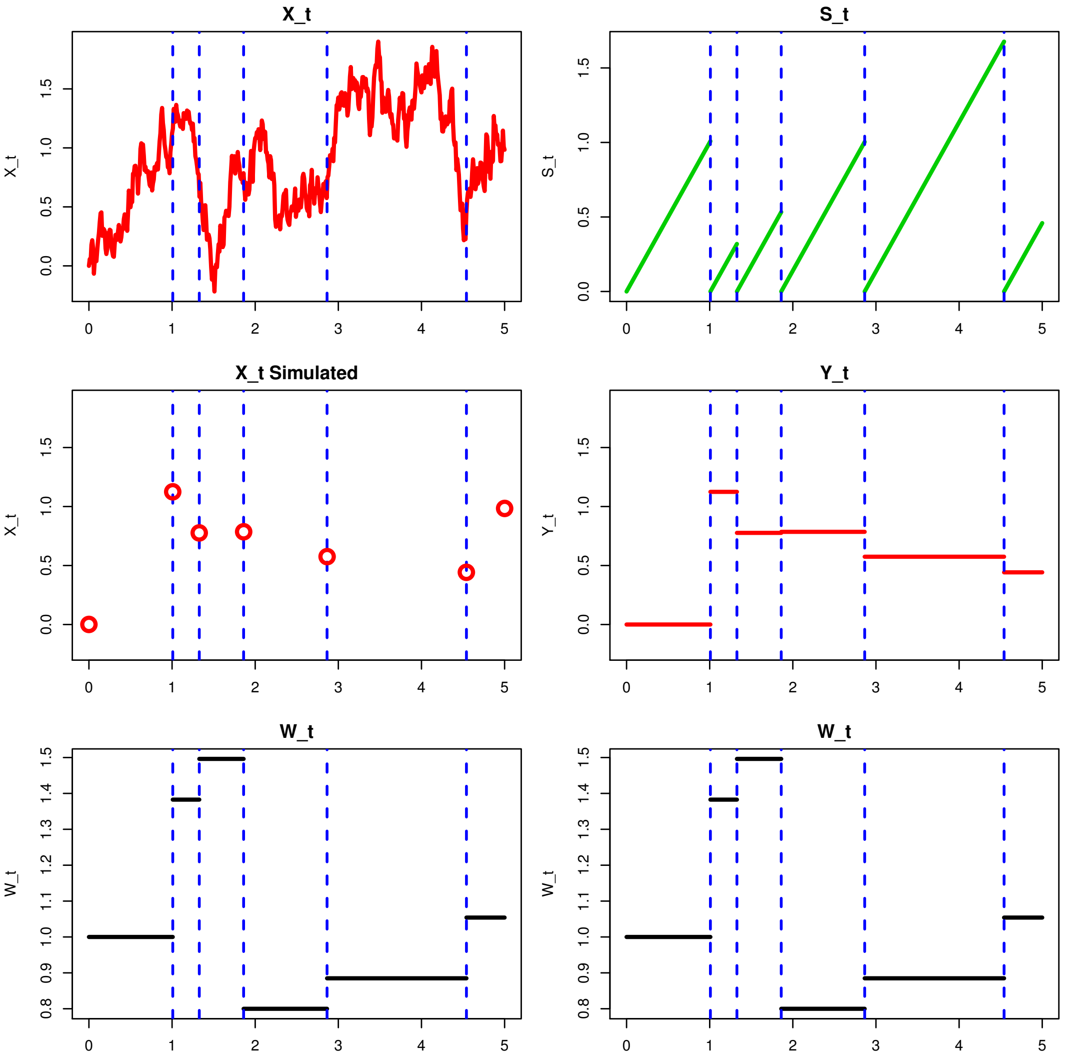

Figure 5: Example of the CIS process. Left-hand column shows the original importance sampling process. Conceptually we think of a proposal process, X_t (top plot), simulated in continuous-time. However we need only simulate this process at event times (denoted by vertical dashed lines) and at times we are interested in. The middle plot shows these simulated values. The weight process is shown in the bottom plot, and the weight changes only at event times. The right-hand column shows the corresponding components of our PDP process: S_t (top); Y_t (middle) and W_t (bottom).

The key to making this a valid importance sampling algorithm is working out an appropriate form of the incremental weight. Without loss of generality we have written the incremental weight in the form given inside the square brackets in (15), as this will simplify the derivation later. To specify the incremental weight, we need to appropriately choose the function ρ.

4.1.1 CIS as a PDP

It can be seen that the CIS process is just a piecewise deterministic Markov process. The only randomness in the process is the timing of the events and the transitions at the events. We can formalise this by defining a PDP with state $\mathbf{Z}_t = (\mathbf{Y}_t, W_t, S_t)$ where $\mathbf{Y}_t$ is the value of the CIS process simulated at the most recent event prior to t, W_t is the importance sampling weight at time t and S_t is the time since the most recent event. See Figure 5 for an example.

This PDP has deterministic dynamics given by differential equations

$$\frac{\mathrm{d}\mathbf{Y}_t}{\mathrm{d}t} = 0, \qquad \frac{\mathrm{d}W_t}{\mathrm{d}t} = 0, \quad \text{and} \quad \frac{\mathrm{d}S_t}{\mathrm{d}t} = 1.$$

Events occur at a rate $\lambda(\mathbf{Z}_t) = \tilde{\lambda}(S_t)$. At an event at time τ we simulate $\mathbf{Y}_\tau$ from $q_{S_{\tau-}}(\cdot|\mathbf{Y}_{\tau-})$, set $S_\tau = 0$ and update W_τ as described in (15).

The generator of $\mathbf{Z}_t$, given by (3), is

$$\mathcal{A}h(\mathbf{y}, w, s) = \frac{\partial h(\mathbf{y}, w, s)}{\partial s} + \tilde{\lambda}(s) \int \left\{ h\left(\mathbf{y}', w\left(1 + \frac{\rho(\mathbf{y}', \mathbf{y}, s)}{\tilde{\lambda}(s)}\right), 0\right) - h(\mathbf{y}, w, s) \right\} q_s(\mathbf{y}' \mid \mathbf{y})\, d\mathbf{y}'. \tag{16}$$

Given the state at time t, $\mathbf{Z}_t = (\mathbf{Y}_t, W_t, S_t)$, our estimate of the expectation of $f(\mathbf{X}_t)$ under our target process will be $W_t f(\mathbf{X})$, where $\mathbf{X}$ is drawn from $q_{S_t}(\cdot \mid \mathbf{Y}_t)$. The requirement that our algorithm is a valid importance sampler then becomes that

$$\mathrm{E}_{\mathbf{X},\mathbf{Z}}\left(W_t f(\mathbf{X})\right) := \mathrm{E}_{\mathbf{Z}}\left(W_t \int f(\mathbf{x}) q_{S_t}(\mathbf{x}|\mathbf{Y}_t)\mathrm{d}\mathbf{x}\right) = \int f(\mathbf{x}) p_t(\mathbf{x})\mathrm{d}t.$$

Here the first expectation is with respect to $\mathbf{Z}$, the PDP, and $\mathbf{X}$ the simulated value for $\mathbf{X}_t$, and the second expectation is just with respect to $\mathbf{Z}$.

As this must hold for all appropriate functions f, we need that for almost all $\mathbf{x}$

$$\mathrm{E}\left(W_t q_{S_t}(\mathbf{x}|\mathbf{Y}_t)\right) = p_t(\mathbf{x}).$$

We will denote the left-hand side by $\tilde{p}_t(\mathbf{x})$. Due to the initialisation of the CIS algorithm we have $\tilde{p}_0(\mathbf{x}) = p_0(\mathbf{x})$. Thus we need to choose the incremental weights such that $\tilde{p}_t(\mathbf{x})$ satisfies the Fokker-Planck equation for the target process

$$\frac{\partial \tilde{p}_t(\mathbf{x})}{\partial t} = \mathcal{L}^* \tilde{p}_t(\mathbf{x}). \tag{17}$$

4.1.2 Obtaining the Incremental Weight

We will give an informal outline of how to derive the function ρ. Throughout this argument we will assume that we can interchange expectation with various operators. See Fearnhead et al. (2016), and the discussion below, for conditions under which this is valid.

We need to choose ρ so that (17) holds. The right-hand side of (17) is just

$$\mathcal{L}^* \tilde{p}_t(\mathbf{x}) = \mathrm{E}\left(\mathcal{L}^* W_t q_{S_t}(\mathbf{x}|\mathbf{Y}_t)\right).$$

The left hand side of (17) is the derivative of an expectation, and thus can be written in terms of the generator of the PDP, $\mathcal{A}$,

$$\frac{\partial \tilde{p}_t(\mathbf{x})}{\partial t} = \frac{\partial \mathrm{E}(W_t q_{S_t}(\mathbf{x}|\mathbf{Y}_t))}{\partial t} = \mathrm{E}\left(\mathcal{A} W_t q_{S_t}(\mathbf{x}|\mathbf{Y}_t)\right)).$$

So both the left and right hand sides of (17) can be written as an expectation of a function of the PDP. For these expectations to be equal it is sufficient for the two functions to be equal to each other.

Using the form of the generator for the CIS process (16) we get that for current state $\mathbf{z} = (\mathbf{y}, w, s)$,

$$\begin{aligned}
\mathcal{A} w q_s(\mathbf{x}|\mathbf{y}) &= \frac{\partial w q_s(\mathbf{x}|\mathbf{y})}{\partial s} + \tilde{\lambda}(s) \int \left\{ w \left(1 + \frac{\rho(\mathbf{y}', \mathbf{y}, s)}{\tilde{\lambda}(s)}\right) q_0(\mathbf{x}|\mathbf{y}') - w q_s(\mathbf{x}|\mathbf{y}) \right\} q_s(\mathbf{y}' \mid \mathbf{y}) \mathrm{d}\mathbf{y}' \\
&= w \frac{\partial q_s(\mathbf{x}|\mathbf{y})}{\partial s} + w \tilde{\lambda}(s) \left\{ \int \left(1 + \frac{\rho(\mathbf{y}', \mathbf{y}, s)}{\tilde{\lambda}(s)}\right) q_0(\mathbf{x}|\mathbf{y}') q_s(\mathbf{y}' \mid \mathbf{y}) \mathrm{d}\mathbf{y}' - q_s(\mathbf{x}|\mathbf{y}) \int q_s(\mathbf{y}' \mid \mathbf{y}) \mathrm{d}\mathbf{y}' \right\} \\
&= w \frac{\partial q_s(\mathbf{x}|\mathbf{y})}{\partial s} + \tilde{\lambda}(s) \left[w q_s(\mathbf{x}|\mathbf{y}) \left(1 + \frac{\rho(\mathbf{x}; \mathbf{y}, s)}{\tilde{\lambda}(s)}\right) - w q_s(\mathbf{x}|\mathbf{y}) \right] \\
&= w \frac{\partial q_s(\mathbf{x}|\mathbf{y})}{\partial s} + w q_s(\mathbf{x}|\mathbf{y}) \rho(\mathbf{x}, \mathbf{y}, s).
\end{aligned}$$

The above argument is informal, as for $s = 0$ (a state visited every time a jump occurs) the function $q_s(x|y)$ does not exist. However it acts, informally, like a dirac delta function. So for any function $h(\mathbf{y}')$ we have $\int q_0(\mathbf{x}|\mathbf{y}') h(\mathbf{y}') \mathrm{d}\mathbf{y}' = h(\mathbf{x})$, and this is used above. We refer the reader to Fearnhead et al. (2016) for formal justification of this step and others.

Thus for the two functions of the PDP that we are taking expectations of to be equal we need

$$w \frac{\partial q_s(\mathbf{x}|\mathbf{y})}{\partial s} + w q_s(\mathbf{x}|\mathbf{y}) \rho(\mathbf{x}, \mathbf{y}, s) = \mathcal{L}^* w q_s(\mathbf{x}|\mathbf{y}),$$

which can be re-arranged to give

$$\rho(\mathbf{x}, \mathbf{y}, s) = \frac{\mathcal{L}^* q_s(\mathbf{x}|\mathbf{y}) - \frac{\partial q_s(\mathbf{x}|\mathbf{y})}{\partial s}}{q_s(\mathbf{x}|\mathbf{y})}. \tag{18}$$

The incremental weight is then $1 + \rho(\mathbf{x}, \mathbf{y}, s)/\tilde{\lambda}(s)$.

The form of this incremental weight is quite intuitive. It is based on the difference between how transition densities change under the target process and under the proposal process. The optimal proposal process would, obviously, be the target process. For this choice $\partial q_s(\mathbf{x}|\mathbf{y})/\partial s = \mathcal{L}^* q_s(\mathbf{x}|\mathbf{y})$, $\rho(\mathbf{x}, \mathbf{y}, s) = 0$, and the importance sampling weights would always be equal to 1. As expected, a proposal process that more closely mimics the target process will have less variable incremental weights. Furthermore we see a trade-off in the choice of the event rate $\tilde{\lambda}(s)$, as larger values of this rate will lead to more events and a higher computational cost, but reduce the variance in the incremental weight. If we were to double the event rate, we would double the expected number of events, but the variance of the incremental weight at an event would reduce by a factor of 4. We expect that the net effect is an overall reduction of Monte Carlo variance in the weights but an increase in computational cost.

One issue with the CIS algorithm is that, for certain combinations of target and proposal processes, it can be possible to get negative weights (similar, for example, to the Russian roulette sampler of Lyne et al., 2015). This can occur if it is possible for $\rho(\mathbf{x},\mathbf{y},s)$ to be smaller than $-\tilde{\lambda}(s)$. For the applications of CIS to simulating from a general diffusion process that are described in Fearnhead et al. (2016), $\rho(\mathbf{x},\mathbf{y},s)$ cannot be bounded below, and thus negative weights are possible. Obviously the probability of getting negative weights can be controlled, with larger $\tilde{\lambda}(s)$ values reducing this probability. This in turn leads to important theoretical and practical issues. The above argument for deriving the incremental weight required the interchange of expectation and operators. The main condition for justifying this is that the we need the importance sampling weights to be such that $\mathbb{E}(|W_t|)$ is finite for any $t > 0$. This condition may not hold due to the possibility of negative weights. In fact, Fearnhead et al. (2016), show that some naive implementations of CIS will not have importance sampling weights that satisfy this condition. Furthermore, for problems where the target process is a diffusion, they give sufficient conditions on both the proposal process and the event rate that will ensure $\mathbb{E}(|W_t|)$ is finite. In general, and within the more specific context of diffusion proposals and targets, Fearnhead et al. (2016) provide conditions under which W_t has p-th moments for $p \geq 1$. The paper also gives practically implementable and intuitive strategies for ensuring that these moments exist.

4.2 Continuous-time Sequential Monte Carlo

To date we have reviewed a continuous-time importance sampling algorithm. This is most naturally viewed as an extension of sequential importance sampling (Liu and Chen, 1998) to continuous-time. However it is then possible to extend this to a continuous-time version of SMC. All we need to do is to jointly simulate multiple CIS processes, and then introduce resampling steps. The simplest implementation of this is to choose a set of resampling times, say $h, 2h, 3h, \ldots, Kh$, and a number of "particles", N. The continuous-time SMC algorithm will then proceed as follows:

(0) **Initiate**. Simulate $\mathbf{x}_0^{(1)}, \ldots, \mathbf{x}_0^{(N)}$ independently from $p_0(\mathbf{x})$. Set $w_0^{(i)} = 1$ for $i = 1, \ldots, N$. Set $t = 0$

(1) **Propagate and Reweight**. For $i = 1, \ldots, N$ run the CIS process for a time interval of length h, with initial values $\mathbf{x}_t^{(i)}$ and $w_t^{(i)}$. Denote the output of the CIS process at time $u = t + h$ by $\tilde{\mathbf{x}}_u^{(i)}$ and $\tilde{w}_u^{(i)}$.

(2) **Resample**. For $i = 1, \ldots, N$ sample k_i from $1, \ldots, N$ with probabilities proportional to $|\tilde{w}_u^{(1)}|, \ldots, |\tilde{w}_u^{(N)}|$. Set $\mathbf{x}_u^{(i)} = \tilde{\mathbf{x}}_u^{(k_i)}$ and

$$w_u^{(i)} = \frac{\tilde{w}_u^{k_i}}{|\tilde{w}_u^{k_i}|} \frac{1}{N} \sum_{j=1}^{N} |\tilde{w}_u^{(j)}|.$$

(3) **Iterate**. Set $t = u$. If $t < K\tau$ go to step (1).

The resampling step is different from standard resampling used in SMC to allow for the possibility of negative weights. The form of the weight after resampling is defined so that resampling is unbiased, and this requires the sign of the weight assigned to any particle value to be unchanged. It is simple to extend the above use of resampling to allow for lower-variance resampling schemes (Kitagawa, 1996; Carpenter et al., 1999; Doucet et al.,

2000) in step (2), and to allow resampling times to depend on the SMC output, for example to be times when the effective sample size of the weights drops below some threshold (Liu and Chen, 1995).

The beneficial effect of resampling will be less when we have negative weights. For example, it is easy to show that $\mathbb{E}(|W_t|)$ is unchanged by resampling. Thus resampling cannot counteract any increase in $\mathbb{E}(|W_t|)$ with t, and this increase will necessarily imply deterioration of Monte Carlo performance with t. Thus the good long-term stability properties that SMC with resampling often has (Del Moral and Guionnet, 2001; Douc et al., 2014) will not be possible in the presence of negative weights. This issue with negative weights is well-known within related quantum Monte Carlo methods (Foulkes et al., 2001), where it is termed the fermion-sign problem.

Fearnhead et al. (2016) suggest alternative resampling approaches for step (2) that can reduce $\mathbb{E}(|W_t|)$ at resampling times, and thus may lead to long-term stability. Alternatively, we need to choose event rates in CIS to be sufficiently large that negative weights are rare over the time-scales that we wish to run an SMC algorithm for.

4.3 CIS for Big Data: the SCALE Algorithm

Recently Pollock et al. (2016a) presented SCALE, an algorithm for sampling from a posterior distribution. The original derivation of SCALE was based on constructing a killed Brownian-motion process whose quasi-stationary distribution is the posterior distribution. The SCALE algorithm then samples from this quasi-stationary distribution. A key property of SCALE is that it only needs to use a small sub-sample of the data at each iteration of the algorithm, and thus it is suitable for large data applications.

Whilst the original derivation is very different, we show here that SCALE can be viewed as a CIS, or continuous-time SMC, algorithm. Our setting is that we wish to sample from a posterior distribution which we will assume can be written as

$$\pi(\mathbf{x}) \propto \prod_{i=1}^{n} \pi_i(\mathbf{x}),$$

where, to keep notation consistent with our presentation of CIS, $\mathbf{x}$ is the parameter vector. Here $\pi_i(\mathbf{x})$ is the likelihood for the ith observation multiplied by the $1/n$th power of the prior. As is common to Bayesian inference, the posterior distribution is only known up to a constant of proportionality. We wish to develop a Monte Carlo algorithm for sampling from this posterior that has good computational properties for large n.

The idea of SCALE and its link to CIS is as follows. We will use CIS to target a stochastic process that has $\pi(\mathbf{x})$ as its stationary distribution. To implement CIS we only need to know the Fokker-Planck equation for this process. If we run CIS (or in practice a continuous-time SMC version) then after a suitable burn-in period this will give us weighted samples from $\pi(\mathbf{x})$. A key property of the CIS algorithm is that the incremental weights depend on the posterior only through $\log \pi(\mathbf{x})$. This is a sum, and it is easy to use sub-sampling to unbiasedly estimate this sum. Thus to deal with potentially large data we will implement a random weight version of CIS (Fearnhead et al., 2008), where we use sub-sampling to estimate the incremental weights. Using unbiased random weights leads to a valid importance sampler, but one with an increased Monte Carlo error. The next ingredient to the SCALE algorithm is to use control variates to reduce the variance of our sub-sampled estimates of the incremental weights, which in turn helps to control the overall Monte Carlo error of the algorithm. Finally Pollock et al. (2016a) use ideas from the Exact Algorithm to avoid the possibility of negative weights. We now detail each of these steps.

The first step to SCALE is the choice of a stochastic process that has $\pi(\mathbf{x})$ as its asymptotical distribution. We need specify this process through its Fokker-Planck equation. SCALE uses the stochastic process for which

$$\frac{\partial p_t(\mathbf{x})}{\partial t} = \frac{1}{2}\sum_{i=1}^{d} \frac{\partial^2 p_t(\mathbf{x})}{\partial x_i^2} - \frac{1}{2\pi(\mathbf{x})}\left(\sum_{i=1}^{d} \frac{\partial^2 \pi(\mathbf{x})}{\partial x_i^2}\right) p_t(\mathbf{x}).$$

It is trivial to see that $\pi(\mathbf{x})$ is an invariant distribution for this stochastic process, as on substituting $p_t(\mathbf{x}) = \pi(x)$ the two terms on the right-hand side cancel. The actual underlying stochastic process can be interpreted as Brownian motion with killing, conditioned on survival: see Pollock et al. (2016a) for more details.

If we implement CIS for this target process, and use Brownian motion as the proposal distribution, we have

$$q_s(\mathbf{x}|\mathbf{y}) = \left(\frac{1}{\sqrt{2\pi s}}\right)^d \exp\left\{-\sum_{i=1}^{d} \frac{(x_i - y_i)^2}{2s}\right\},$$

and

$$\frac{\partial q_s(\mathbf{x}|\mathbf{y})}{\partial s} = \frac{1}{2}\sum_{i=1}^{d} \frac{\partial^2 q_s(\mathbf{x}|\mathbf{y})}{\partial x_i^2}.$$

Thus from (18), the function that determines the incremental weights is

$$\rho(\mathbf{x}, \mathbf{y}, s) = -\frac{1}{2\pi(\mathbf{x})}\sum_{i=1}^{d} \frac{\partial^2 \pi(\mathbf{x})}{\partial x_i^2} = -\frac{1}{2}\sum_{i=1}^{d}\left[\frac{\partial^2 \log \pi(\mathbf{x})}{\partial x_i^2} + \left(\frac{\partial \log \pi(\mathbf{x})}{\partial x_i}\right)^2\right].$$

The right-hand side depends on $\pi(\mathbf{x})$ only through derivatives of

$$\log \pi(\mathbf{x}) = \text{constant} + \sum_{i=1}^{n} \log \pi_i(\mathbf{x}).$$

Importantly these derivatives do not depend on the unknown normalising constant of $\pi(\mathbf{x})$. Furthermore as they are sums, it is simple to unbiasedly estimate the derivatives. For example given j and k, two independent draws from a uniform distribution on $1, \ldots, n$, we can estimate $\rho(\mathbf{x}, \mathbf{y}, s)$ by

$$-\frac{1}{2}\sum_{i=1}^{d}\left[n\frac{\partial^2 \log \pi_j(\mathbf{x})}{\partial x_i^2} + n^2\left(\frac{\partial \log \pi_j(\mathbf{x})}{\partial x_i}\right)\left(\frac{\partial \log \pi_k(\mathbf{x})}{\partial x_i}\right)\right]. \tag{19}$$

We can reduce the variance of our estimate of ρ, and hence of the incremental weights, using control variates. Pollock et al. (2016a) suggest using a pre-processing step that finds a values $\hat{\mathbf{x}}$ close to the posterior mode. We precalculate the first and second derivates of $\log \pi(\mathbf{x})$ at $\hat{\mathbf{x}}$, and calculate the value, $\hat{\rho}$, of ρ at $\hat{\mathbf{x}}$. We then estimate of ρ at $\mathbf{x}$ as

$$-\frac{n}{2}\sum_{i=1}^{d}\left\{\left[\frac{\partial^2 \log \pi_j(\mathbf{x})}{\partial x_i^2}-\frac{\partial^2 \log \pi_j(\hat{\mathbf{x}})}{\partial x_i^2}\right]+n\left[\frac{\partial \log \pi_j(\mathbf{x})}{\partial x_i}-\frac{\partial \log \pi_j(\hat{\mathbf{x}})}{\partial x_i}\right]\left[\frac{\partial \log \pi_j(\mathbf{x})}{\partial x_i}-\frac{\partial \log \pi_j(\hat{\mathbf{x}})}{\partial x_i}+2\frac{1}{n}\frac{\partial \log \pi(\hat{\mathbf{x}})}{\partial x_i}\right]\right\}+\hat{\rho} \tag{20}$$

where, as before, j and k are independent draws from an uniform distribution on $1,\dots,n$. The idea behind this control-variate approach is that if $\hat{\mathbf{x}}$ is within $1/\sqrt{n}$ of the posterior mode, then with high-probability, at stationarity the CIS process will be at an $\mathbf{x}$ value that is within $1/\sqrt{n}$ of $\hat{\mathbf{x}}$. If the first and second derivatives of the $\log \pi_j$s are well-behaved this means that the terms in the square brackets will be $O_p(1/\sqrt{n})$, and thus ρ will be $O(n)$ with high probability. This compares well with the naive subsampling estimator of ρ (19), which will be $O_p(n^2)$.

Finally, to avoid negative weights, the SCALE algorithm then uses ideas from Beskos et al. (2008) and Burq and Jones (2008), to simulate the proposal process in such a way that we know an upper and lower bound the process takes within a given time-interval. With such bounds we can then choose our event-rate λ sufficiently high that negative weights do not occur. This approach does come with a computational cost, as simulating Brownian motion together with such a bound can be an order of magnitude, or more, slower than just simulating Brownian motion. Below we investigate the feasibility of implementing a version of SCALE that allows for negative weights, but that chooses event-rates to be sufficiently large that they are rare.

4.4 Extensions

Here we give two examples of how we can use the theory for PDPs to easily obtain generalisations of the basic CIS and SCALE algorithms.

4.4.1 Alternative Proposal Distributions

With any importance sampling approach, the choice of proposal distribution can have a substantial impact on the resulting Monte Carlo properties. The original derivation of CIS used the idea of a continuous-time stochastic process that was being used as a proposal process. However our derivation of the CIS algorithm has not actually required us to specify a proposal process, just a suitable family of transition densities, $q_s(\mathbf{x}|\mathbf{y})$. This family needs to have certain properties, such as being differentiable with respect to s, so that the incremental weight (18) can be calculated. This appears to make it easy to consider alternative proposals, as we only need to specify an appropriate family of densities. For example, in standard importance sampling applications it is often recommended to use heavy-tailed proposals, so for CIS it is natural to consider transition densities that are t-distributed, such as

$$q_s(\mathbf{x}|\mathbf{y}) \propto s^{-d/2}\frac{1}{(1+\frac{1}{\nu}\sum_{i=1}^{d}(x_i-y_i)^2/s)^{(\nu+d)/2}}, \tag{21}$$

for some appropriately chosen degrees of freedom $\nu > 0$.

4.4.2 Alternative SCALE Processes

We can also develop alternatives to the SCALE algorithm that differ in terms of the underlying stochastic process that they target. For example we can target the Langevin diffusion, for which the Fokker-Planck equation is

$$\frac{\partial p_t(\mathbf{x})}{\partial t} = \frac{1}{2}\sum_{i=1}^{d} \frac{\partial^2 p_t(\mathbf{x})}{\partial x_i^2} - \frac{1}{2}\sum_{i=1}^{d}\left[\frac{\partial}{\partial x_i}\left(p_t(\mathbf{x})\frac{\partial \log \pi(\mathbf{x})}{\partial x_i}\right)\right].$$

Again, it is simple to see, by substituting $p_t(\mathbf{x}) = \pi(\mathbf{x})$ on the right-hand side, that $\pi(\mathbf{x})$ is the invariant distribution for this process. If we implement CIS to sample from this Langevin diffusion, and use Brownian Motion as the proposal distribution, we get that the incremental weights are $1 + \rho(\mathbf{x}, \mathbf{y}, s)/\tilde{\lambda}(s)$ with

$$\rho(\mathbf{x}, \mathbf{y}, s) = -\frac{1}{2}\sum_{i=1}^{d}\left[\frac{(y_i - x_i)}{s}\frac{\partial \log \pi(\mathbf{x})}{\partial x_i} + \frac{\partial^2 \log \pi(\mathbf{x})}{\partial x_i^2}.\right]$$

We can unbiasedly estimate this using a sub-sample of size 1. For example, if j is drawn uniformly from $\{1, \ldots, n\}$, one unbiased estimator is

$$\hat{\rho}(\mathbf{x}, \mathbf{y}, s) = -\frac{n}{2}\sum_{i=1}^{d}\left[\frac{(y_i - x_i)}{s}\frac{\partial \log \pi_j(\mathbf{x})}{\partial x_i} + \frac{\partial^2 \log \pi_j(\mathbf{x})}{\partial x_i^2}\right].$$

Alternatively we can develop lower-variance esimators using control variates. The incremental weight is easier to estimate than the incremental weight in SCALE as it does not involve a square of the gradient of the log-posterior. However, the $(y_i - x_i)/s$ term has a variance, under the Brownian motion proposal, that is $1/s$, and care is needed to control this term for small values of the inter-event time, s.

4.5 Example: Inference for Mixture Models

To give an indication of how the SCALE algorithm of Section 4.3 works, some of the issues with its implementation, and the importance of using control-variates with the sub-sampling estimator or ρ, we will consider the simple example of Section 3.6.

Initially we implemented the SCALE algorithm without any subsampling on a data-set of size $n = 150$ and $p = 0.95$. Figure 6 show the posterior distribution for x for this data set, the log-posterior and how ρ varies as a function of x. Note that the average value of ρ at stationarity is 0, so values where ρ is greater than or less than 0 show regions where weights of particles will increase or decrease relative to the average weight. The key point in these figures is that while both the posterior and log-posterior are relatively well-behaved, ρ has multiple pronounced modes.

This multi-modality can cause issues with mixing. For example a particle that is currently at a value of -5 will have to traverse a prolonged region where ρ is negative to reach the main region of posterior mass at values of x close to 4. As it traverses this region its weight will reduce substantially, and it is likely to be lost due to

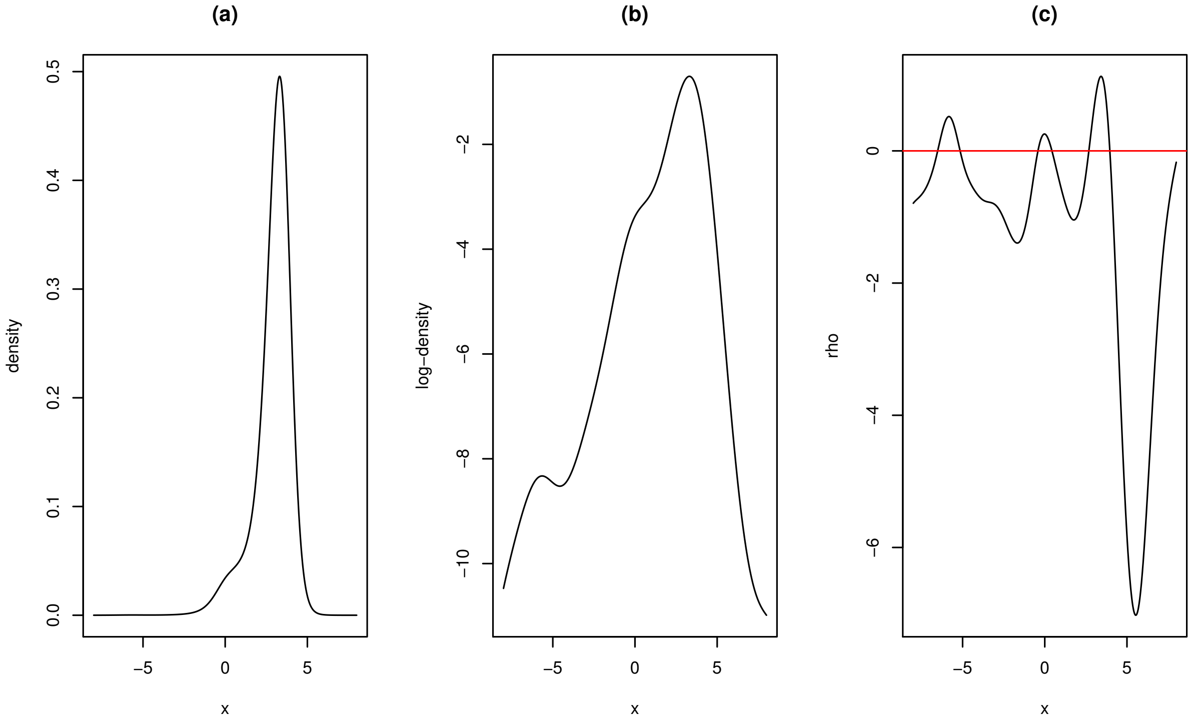

Figure 6: The posterior distribution (a); the log-posterior (b); and ρ (c) for a data set of size 150 from the mixture model. Note that for the SCALE algorithm $\rho(x, y, s)$ simplifies to a function only of x.

resampling. Thus we could have relatively rare movement of particles from one mode of ρ to another.

In practice this means that initialisation of the SCALE algorithm can be crucial. We implemented SCALE with 200 particles over a time interval of length 100. We considered resampling at every integer time-point, and resampled if the effective sample size of the weights was less than 100. We ran SCALE with a constant event rate of 12, and observed no negative weights. The performance of SCALE appeared quite sensitive to the event rate, with rates of 10 and less giving noticeably worse performance. Figure 7 shows output from two runs of SCALE, one initialised with particles drawn from the prior, and the other initialised with particles drawn uniformly on $[-10, -5]$. Figure 7 (a) and (c) show the evolution of the particles over the first 50 time-steps. They evolve according to Brownian motion, but accrue or lose weight depending whether ρ is positive or negative at the particle values at event times. Particles with low weights tend to be lost at resampling times. For the case where we initialise particles from the prior, we see particles are quickly lost in regions aroung $x = 0$ where ρ is negative. More slowly they are also lost from regions around $x = -5$ which is a local-maxima in ρ, as these particles, on average, attain a smaller weight than those particles close to the posterior mode. By about time 15 the SCALE algorithm appears to have converged, and Figure 7 (b) shows that it gives a good approximation to the true posterior distribution. By comparison, when SCALE is initialised with no particles close to the posterior mode, the particles appear to get stuck close to local mode of ρ at $x = -5$.

We now turn our attention to the possibly benefits of using subsampling to estimate ρ. Firstly to show the difference between subsampling with and without control variates, Figure 8(a) shows the variance of our the

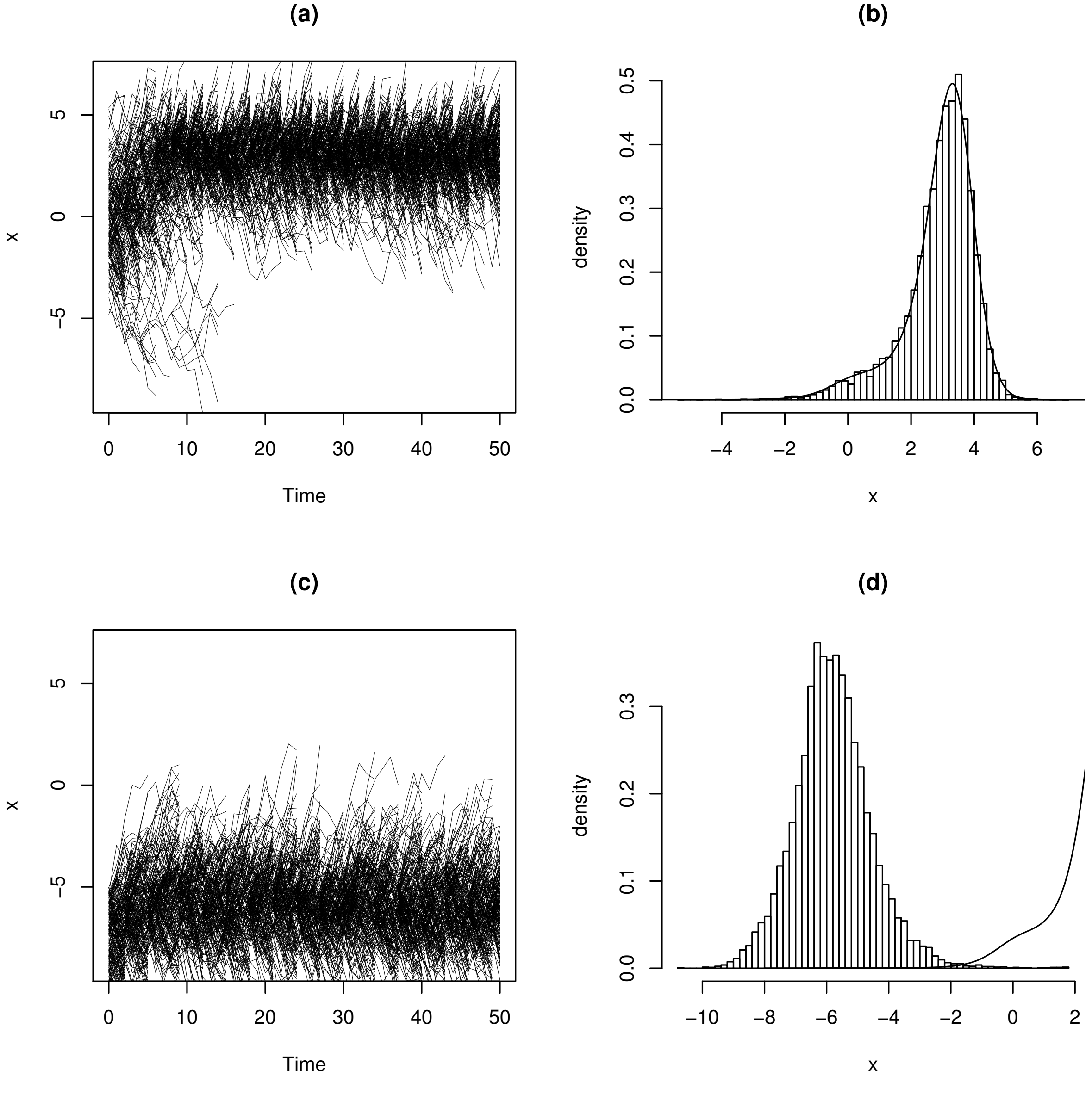

Figure 7: Evolution of particles up to time 50 (left-hand column) and estimate of posterior (right-hand column) from SCALE algorithm. Top row is for particles initiated from prior and bottom row for particles initiated uniformly on $[-10, -5]$. Estimates of posterior are show as histograms based on the weighted particles from time 25 to 100, and the true posterior is overlain.

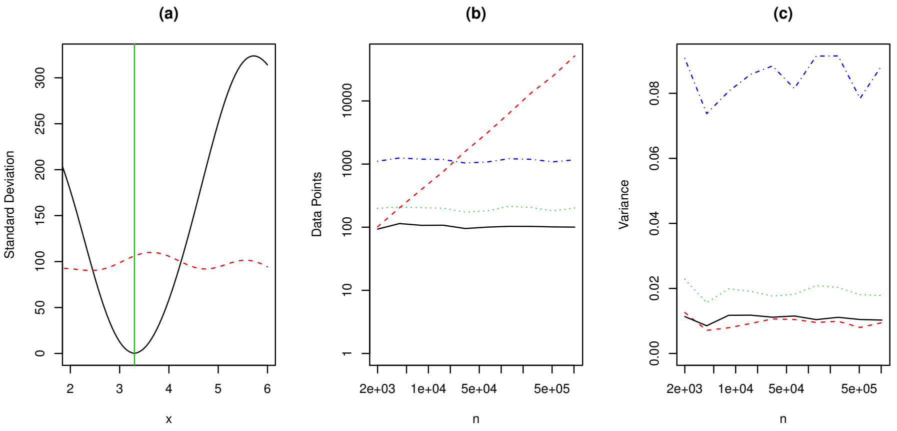

Figure 8: Variance of estimate of ρ using subsampling with (black full-line) and without (red dashed line) control variates (plot a); the vertical line shows the value of $\hat{x}$. Computational cost of estimating W_h, measured in terms of number of data point accessed, as a function of n (plot b), and variance of W_h as a function of n (plot c). For these latter two plots, results are shown for no subsampling (red dashed line) and subsampling with $\hat{x}$ as the posterior mode (black full-line) the mode plus posterior standard deviation (green dotted line) and mode plus three times the standard deviation (blue dot-dash line). Results calculated from $2,000$ estimates of W_h for each method and each value of n.

sub-sampling estimate of ρ for our data set of size 150. The control variate estimator (20) was implemented with $\hat{x}$ set to be the posterior mode. We see a substantially lower variance for x close to $\hat{x}$ when we use control variates. Though as x moves away from $\hat{x}$ the variance increases, and eventually is worse than not using control-variates. The key to why control variates works for large data is that if $\hat{x}$ is close to the posterior mode then x will be very close to $\hat{x}$ with high-probabililty. In such cases the proportionate reduction in variance will be $O(n)$ – and thus the gains of using control variates will increase for larger data sets.

To gain insight into the benefit of using sub-sampling we looked at the variance of the weight at time h, W_h, of a particle sampled from the true posterior, for different values of n both with and without subsampling. For a fixed variance of W_h we wanted to see how many data-points need to be processed by each algorithm. So without sub-sampling, each event requires access to all n data points, whereas with sub-sampling each event requires us to access only 2 data points. We found that sub-sampling without control variates performed substantially worse than the two alternatives, with increasingly poor performance as n increases. Thus we focus on comparing no subsampling and subsampling with control variates.

As we increase n the posterior standard deviation will decrease at a rate $1/\sqrt{n}$, and thus we choose $h = 1/n$ so that the distance moved by the particle will also be of order $1/\sqrt{n}$. When using no-subsampling we chose the event rate to be $n/2$. For sub-sampling with control variates we set $\hat{x}$ to be the posterior mode and let the event rate depend on the value of the particle at the most recent event, x' say, and be of the form $2n + 4n^2(x' - \hat{x})^2$. These choices gave variances of W_h that were similar for both cases and also for different n. Figure 8(b) shows how the number of data-points accessed varied with n for the two methods and Figure 8(c) shows the estimates of the variance of W_h. As n increases we have that the computational cost of using sub-sampling, as measured by the number of data-points accessed, remains constant. By comparison, without sub-sampling, to maintain a fixed variance for W_h we need to increase the computational cost linearly.

Finally we looked at how the choice of $\hat{x}$ affected the performance of subsampling with control variates. We repeated the above study but setting $\hat{x}$ to be either one posterior standard-deviation or three posterior standard-deviations from the mode. These correspond to values in the body of the posterior and in the tail of the posterior respectively. From 8 (b) and (c) we see that as $\hat{x}$ moves away from the posterior mode, the performance of the sub-sampler decreases both in terms of computational cost and variance. However, for a fixed variance of W_h both methods have a computational cost that is constant with n, as opposed to the linearly increasing cost we have when sub-sampling is not used.

5 Discussion

We have shown how piecewise deterministic processes can be used to derive continuous-time versions of sequential Monte Carlo and MCMC algorithms. These algorithms are fundamentally different from more standard discrete-time versions. Currently only a few specific algorithms, from a much wider class of possibilities, have been suggested. Whilst we have suggested a few extensions of existing methods, these just touch the surface of the range of developments that are possible. Whilst not discussed here, the continuous-time SMC methods seem particularly well-suited for implementation on a distributed computing architecture, as evolution of particles can be carried out in parallel. As well as such potential methodological developments, there are a wide-range of open theoretical questions. For example, can we get results on how well continuous-time MCMC mixes? For such results to be practically meaningful they would need to account for the computational cost of simulating the underlying PDP, as opposed to just measuring the mixing properties of the PDP itself. Can we characterise the situations where continuous-time MCMC is more efficient than its discrete-time counterpart? Or understand which versions of continuous-time MCMC are most efficient, and when? It is also of interest to explore links between continuous-time MCMC and discrete-time versions, with Sherlock and Thiery (2017) and Vanetti et al. (2017) showing links between the Bouncy Particle Sample and both a delayed rejection MCMC algorithm (Tierney and Mira, 1999) and the slice sampler (Neal, 2003).

We have also shown how subsampling ideas, where we approximate the gradient of the log-posterior using a small sample of data points, can be used with these continuous-time methods. Unlike many other subsampling approaches, the methods still remain “exact”, in the sense that they still target the true posterior. Subsampling reduces the computational cost per iteration but does lead to a increase in Monte Carlo error for a fixed number of iterations. In the examples we have considered, it is only when using control variate ideas to reduce the variance of our sub-sampling estimator of the gradient of the log posterior, that we see any overall gain in efficiency of the algorithm. Furthermore, when using suitable control variates, it appears possible to obtain algorithms whose computational cost per effective sample size increases sub-linearly with the number of data points. This adds to existing evidence of the importance of using control variates if we wish to have some form of super-efficiency for big data problems (Bardenet et al., 2017).

Acknowledgements The authors thank the Engineering and Physical Sciences Research Council for support through grants EP/K014463/1 (i-Like) and EP/D002060/1 (CRiSM).

References

Andrieu, C. and Roberts, G. O. (2009). The pseudo-marginal approach for efficient Monte Carlo computations. *Annals of Statistics*, 37:697–725.

Baker, J., Fearnhead, P., Fox, E. B., and Nemeth, C. (2017). Control Variates for Stochastic Gradient MCMC. *ArXiv e-prints 1706.05439.*

Bardenet, R., Doucet, A., and Holmes, C. C. (2017). On Markov chain Monte Carlo methods for tall data. *Journal of Machine Learning Research*, 18:47:1–47:43.

Beskos, A., Papaspiliopoulos, O., and Roberts, G. O. (2006). Retrospective exact simulation of diffusion sample paths with applications. *Bernoulli*, 12(6):1077–1098.

Beskos, A., Papaspiliopoulos, O., and Roberts, G. O. (2008). A factorisation of diffusion measure and finite sample path constructions. *Methodology and Computing in Applied Probability*, 10:85–104.

Beskos, A. and Roberts, G. O. (2005). Exact simulation of diffusions. *The Annals of Applied Probability*, 15:2422–2444.

Bierkens, J. (2015). Non-reversible Metropolis-Hastings. *Statistics and Computing*, 25:1–16.

Bierkens, J., Bouchard-Cote, A., Duncan, A., Doucet, A., Fearnhead, P., Roberts, G., and Vollmer, S. (2017a). Piecewise Deterministic Markov Processes for Scalable Monte Carlo on Restricted Domains. *ArXiv:1701.04244.*

Bierkens, J. and Duncan, A. (2017). Limit theorems for the Zig-Zag process. *Advances in Applied Probability*, 49(3):791–825.

Bierkens, J., Fearnhead, P., and Roberts, G. (2016). The Zig-Zag Process and Super-Efficient Sampling for Bayesian Analysis of Big Data. *ArXiv:1607.03188.*

Bierkens, J., Roberts, G., et al. (2017b). A piecewise deterministic scaling limit of lifted Metropolis–Hastings in the Curie–Weiss model. *The Annals of Applied Probability*, 27(2):846–882.

Bouchard-Côté, A., Vollmer, S. J., and Doucet, A. (2017). The Bouncy Particle Sampler: A Non-Reversible Rejection-Free Markov Chain Monte Carlo Method. *Journal of the American Statistical Association*, page to appear.

Burq, Z. and Jones, O. (2008). Simulation of Brownian motion at first passage times. *Mathematics and Computers in Simulation*, 77:64–71.

Carpenter, J., Clifford, P., and Fearnhead, P. (1999). An improved particle filter for non-linear problems. *IEE proceedings-Radar, Sonar and Navigation*, 146:2–7.

Cinlar, E. (2013). *Introduction to stochastic processes.* Courier Corporation.

Davis, M. H. (1984). Piecewise-deterministic Markov processes: A general class of non-diffusion stochastic models. *Journal of the Royal Statistical Society. Series B*, 46:353–388.

Davis, M. H. A. (1993). *Markov models and optimization*, volume 49 of *Monographs on Statistics and Applied Probability.* Chapman & Hall, London.

Del Moral, P. and Guionnet, A. (2001). On the stability of interacting processes with applications to filtering and genetic algorithms. *Annales de l'IHP Probabilités et statistiques*, 37(2):155–194.

Diaconis, P., Holmes, S., and Neal, R. (2000). Analysis of a nonreversible Markov chain sampler. *The Annals of Applied Probability*, 10(3):726–752.

Douc, R., Moulines, E., Olsson, J., et al. (2014). Long-term stability of sequential Monte Carlo methods under verifiable conditions. *The Annals of Applied Probability*, 24(5):1767–1802.

Doucet, A., Godsill, S. J., and Andrieu, C. (2000). On sequential Monte Carlo sampling methods for Bayesian filtering. *Statistics and Computing*, 10:197–208.

Dubey, K. A., Reddi, S. J., Williamson, S. A., Poczos, B., Smola, A. J., and Xing, E. P. (2016). Variance reduction in stochastic gradient langevin dynamics. In *Advances in Neural Information Processing Systems*, pages 1154–1162.

Ethier, S. N. and Kurtz, T. G. (2005). *Markov Processes: Characterization and Convergence (Wiley Series in Probability and Statistics).* Wiley-Interscience.

Fearnhead, P., Latuszynski, K., Roberts, G. O., and Sermaidis, G. (2016). Continuous-time Importance Sampling: Monte Carlo Methods which Avoid Time-Discretisation Error.

Fearnhead, P., Papaspiliopoulos, O., and Roberts, G. O. (2008). Particle filters for partially-observed diffusions. *Journal of the Royal Statistical Society Series B*, 70:755–777.

Foulkes, W., Mitas, L., Needs, R., and Rajagopal, G. (2001). Quantum Monte Carlo simulations of solids. *Reviews of Modern Physics*, 73(1):33.

Girolami, M. and Calderhead, B. (2011). Riemann manifold Langevin and Hamiltonian Monte Carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 73(2):123–214.

Gustafson, P. (1998). A guided walk Metropolis algorithm. *Statistics and Computing*, 8(4):357–364.

Kitagawa, G. (1996). Monte Carlo filter and smoother for non-Gaussian nonlinear state space models. *Journal of Computational and Graphical Statistics*, 5:1–25.

Lewis, P. A. W. and Shedler, G. S. (1979). Simulation of nonhomogeneous Poisson processes by thinning. *Naval Res. Logist. Quart.*, 26(3):403–413.

Li, C., Srivastava, S., and Dunson, D. B. (2017). Simple, scalable and accurate posterior interval estimation. *Biometrika*, 104(3):665–680.

Liu, J. S. and Chen, R. (1995). Blind deconvolution via sequential imputations. *Journal of the American Statistical Association*, 90:567–576.

Liu, J. S. and Chen, R. (1998). Sequential Monte Carlo methods for dynamic systems. *Journal of the American Statistical Association.*, 93:1032–1044.

Lyne, A.-M., Girolami, M., Atchad, Y., Strathmann, H., and Simpson, D. (2015). On Russian Roulette Estimates for Bayesian Inference with Doubly-Intractable Likelihoods. *Statist. Sci.*, 30(4):443–467.

Ma, Y.-A., Chen, T., and Fox, E. (2015). A complete recipe for stochastic gradient MCMC. In *Advances in Neural Information Processing Systems*, pages 2917–2925.

McGrayne, S. B. (2011). *The theory that would not die: how Bayes' rule cracked the enigma code, hunted down Russian submarines, & emerged triumphant from two centuries of controversy.* Yale University Press.

Neal, R. M. (1998). Suppressing random walks in Markov chain Monte Carlo using ordered overrelaxation. In *Learning in Graphical Models*, pages 205–228. Springer.

Neal, R. M. (2003). Slice sampling. *Annals of Statistics*, pages 705–741.

Neal, R. M. (2004). Improving Asymptotic Variance of MCMC Estimators: Non-reversible Chains are Better. Technical report, No. 0406, Department of Statistics, University of Toronto.

Neal, R. M. et al. (2011). MCMC using Hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, 2:113–162.

Neiswanger, W., Wang, C., and Xing, E. P. (2014). Asymptotically exact, embarrassingly parallel MCMC. In *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence*, pages 623–632. AUAI Press.

Oksendahl, B. (2007). *Stochastic Differential Equations.* Springer-Verlag, Berlin.

Pakman, A., Gilboa, D., Carlson, D., and Paninski, L. (2017). Stochastic bouncy particle sampler. *Proceedings of ICML.*

Peters, E. A. J. F. and de With, G. (2012). Rejection-free Monte Carlo sampling for general potentials. *Physical Review E*, 85(2):026703.

Pollock, M., Fearnhead, P., Johansen, A., and Roberts, G. O. (2016a). An Unbiased and Scalable Monte Carlo Method for Bayesian Inference for Big Data. *ArXiv:1609.03436.*

Pollock, M., Johansen, A. M., and Roberts, G. O. (2016b). On the exact and ε-strong simulation of (jump) diffusions. *Bernoulli*, 22(2):794–856.

Quiroz, M., Villani, M., and Kohn, R. (2015). Speeding up MCMC by efficient data subsampling. *Riksbank Research Paper Series*, (121).

Robert, C. and Casella, G. (2011). A Short History of Markov Chain Monte Carlo: Subjective Recollections from Incomplete Data. *Statistical Science*, 26(1):102–115.

Roberts, G. O. and Rosenthal, J. S. (1998). Optimal scaling of discrete approximations to Langevin diffusions. *Journal of the Royal Statistical Society: Series B*, 60(1):255–268.

Scott, S. L., Blocker, A. W., and Bonassi, F. V. (2016). Bayes and Big Data: The Consensus Monte Carlo Algorithm. *International Journal of Management Science and Engineering Management*, 11(2):78–88.

Sherlock, C. and Thiery, A. H. (2017). A discrete bouncy particle sampler. *arXiv:1707.05200.*

Srivastava, S., Cevher, V., Tran-Dinh, Q., and Dunson, D. B. (2015). WASP: Scalable Bayes via barycenters of subset posteriors. In *AISTATS*.

Tierney, L. and Mira, A. (1999). Some adaptive Monte Carlo methods for Bayesian inference. *Statistics in Medicine*, 18(1718):2507–2515.

Vanetti, P., Bouchard-Côté, A., Deligiannidis, G., and Doucet, A. (2017). Piecewise deterministic Markov chain Monte Carlo. *arXiv:1707.05296.*

Welling, M. and Teh, Y. W. (2011). Bayesian learning via stochastic gradient Langevin dynamics. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 681–688.

A Details for Robust Regression Example

Here we give further details of the robust regression example of Section 3.4, together with details as to how we bound the event rates so that we can simulate a PDP that samples from the posterior for this model.

To ease the exposition, we will slightly change notation so that we can write the robust regression model using standard statistical notation. We assume we have n observations, the realisation of the vector random variable $\mathbf{Y} = (Y_1, \dots, Y_n)^T$. For each observation we have a set of covariates, with covariates $(x_{i1}, \dots, x_{id})$ for observation i, and we have $x_{i1} = 1$. Let X be the $n \times d$ matrix with entries x_{ij}, then our model is

$$Y = X\boldsymbol{\beta} + \boldsymbol{\epsilon},$$

where $\boldsymbol{\beta} = (\beta_1, \dots, \beta_j)^T$ is a d-vector of parameters that we wish to infer, and ϵ is an n-vector of independent, identically distributed noise random variables. We assume the marginal distribution of one of these is an equal mixture of a standard normal random variable and a normal random variable with variance 10^2. Thus, for this section we are using X to denote the covariates and $\boldsymbol{\beta}^T$, the vector of parameters, will be the position component of our PDP.

Fix the data set, $\mathbf{y} = (y_1, \ldots, y_n)^T$, and assume independent improper uniform priors on β_j for $j = 1, \ldots, d$. We will derive upper bounds on the event-rate for the bouncy-particle sampler, with similar derivations available to bound the event-rates for the Zig-Zag sampler.

The log-posterior can be written in terms of the residuals for each observation. Assume the current position of our PDP is $\boldsymbol{\beta}^T$ and the current velocity is $\mathbf{v}$. Then, up until the next event-time, the residuals at time t in the future, $(e_1(t), \ldots, e_n(t))$ will be

$$(e_1(t), \ldots, e_n(t)) = \mathbf{y}^T - \left(\boldsymbol{\beta}^T + \mathbf{v}t\right) X^T = \mathbf{y}^T - \boldsymbol{\beta}^T X^T - t\mathbf{v}X^T.$$

Minus the log-posterior can be written as $\sum_{i=1}^n g(e_i(t))$, where

$$g(e) = -\log\left[\exp\left\{-\frac{1}{2}e^2\right\} + \frac{1}{10}\exp\left\{-\frac{1}{200}e^2\right\}\right].$$

Define $\mathbf{U}\left(\boldsymbol{\beta}^T + t\mathbf{v}\right)$ to be the gradient of the log-posterior at the corresponding time. The event-rate of the Bouncy Particle Sampler depends on

$$\mathbf{v} \cdot \mathbf{U}\left(\boldsymbol{\beta}^T + t\mathbf{v}\right) = \sum_{i=1}^n \left(\mathbf{v}X^T\right)_i g'(e_i(t)),$$

where $\left(\mathbf{v}X^T\right)_i$ denotes the ith entry of the vector $\mathbf{v}X^T$ and $g'(\cdot)$ denotes the derivate of $g(\cdot)$. We can bound the time-derivative of this quantity

$$\frac{\mathrm{d}\left\{\mathbf{v} \cdot \mathbf{U}\left(\boldsymbol{\beta}^T + t\mathbf{v}\right)\right\}}{\mathrm{d}t} = \sum_{i=1}^n \left(\mathbf{v}X^T\right)_i^2 g''(e_i(t)) < \sum_{i=1}^n \left(\mathbf{v}X^T\right)_i^2 = \mathbf{v}X^T X\mathbf{v}^T,$$

where $g''(\cdot)$ denotes the second derivate of $g(\cdot)$. The inequality comes from the fact that this second derivative is strictly less than 1 – this bound being straightforward, albeit tedious, to obtain. (It is possible to get a slightly stronger bound using the fact that maximum of $g''(\cdot)$ is $g''(0)$.)

This bound on the derivative can be used to get a piecewise linear bound on the event rate of the bouncy particle sample (see Bierkens et al., 2016) as follows:

$$\min\left\{0, \mathbf{v} \cdot \mathbf{U}\left(\boldsymbol{\beta}^T + \mathbf{v}\right)\right\} \leq \min\left\{0, \mathbf{v} \cdot \mathbf{U}\left(\boldsymbol{\beta}^T\right) + t\mathbf{v}X^T X\mathbf{v}^T, \right\}.$$

It is possible to simulate events from a Poisson process with event rate given by a piecewise linear function via inversion. Thus we can simulate events in the Bouncy Particle Sampler by proposing events with the above rate and using thinning.