Using the Bouncy Particle Sampler To Remove Reversibility in MCMC

Rebekah Fearnhead

1 Introduction

Markov Chain Monte Carlo (MCMC) is a class of algorithms which can be used to draw samples from a probability distribution, often ones which are high dimensional or complex meaning that analytic techniques alone can not be used to study them. This is achieved by constructing a Markov Chain with elements which have a distribution that approximates the distribution of interest. This allows the chain's equilibrium distribution to match the target distribution, and the more steps performed in the chain, the more closely the sample produced matches the desired distribution.

One of the first algorithms produced was the Metropolis-Hastings algorithm [13][11]. This method involves using a proposal density for new steps to create a Markov chain and then adds a method for rejecting some of the proposed moves. This was then used as a general framework for many later algorithms including Gibbs Sampling [9] and Metropolis-adjusted Langevin Algorithm (MALA) [1]. All these methods have some similar properties, with one of these being that they are all reversible.

2 Disadvantages of Reversibility

One disadvantage of the methods mentioned above is that, as they satisfy detailed balance they are therefore reversible. This means that the dynamics of the process are the same both forwards and backwards in time [6]. The Markov Chain exhibits random walk behaviour and therefore it is possible for it to return to regions of the statespace recently visited. This may lead to it exploring the statespace less efficiently than a chain would if it was non-reversible due to the worse mixing. Better mixing properties in non-reversible chains occur because the random walk behaviour is suppressed, meaning it will tend to move in the same direction in subsequent steps [10], compared to a random walk, where each step is equally as likely to move in any direction.

One method for creating non-reversible MCMC algorithms is to use 'lifting' [8]. This involves defining a Markov Chain on a higher dimensional state space than what is being sampled on. If we wish to sample from $\pi(\boldsymbol{\theta})$, which is a target distribution, a Markov Chain with state $(\boldsymbol{\theta}, \mathbf{p})$ can be used. The variable $\mathbf{p}$ can be interpreted as the momentum or velocity which describes the direction and speed the Markov Chain is moving in, and is of the same dimension, d, as $\boldsymbol{\theta}$. This achieves non-reversibility by trying to encourage successive Markov chain moves to be in roughly the same direction.

2.1 Continuous-Time MCMC

One type of MCMC algorithms which exhibit non-reversible behaviour are piecewise deterministic Markov Processes (PDMP). These are based off simulating a continuous-time Markov Chain and then taking equally spaced samples, and are Markov processes which evolve deterministically between a set of random events.

To be able to use a PDMP as part of a sampling algorithm, we need to be able to simulate from the PDMP. One way to store a continuous-time path from a PDMP is to store the initial state, and the time and event after each event. Given these, we can then fill in the continuous-time path using a transition function for the deterministic dynamics.

The main challenge of this is to simulate the event times. However, it can be shown that the time until the next event can be modelled as the time until the first event in a time inhomogeneous Poisson process [8][12].

2.2 Limiting Process

If we want to sample from a distribution $\pi(\theta)$, for a scalar θ, we can introduce a momentum p, and define the state $\mathbf{z} = (\theta, p)$. We can define the limiting process of a univariate PDMP with $p \in \{-1, 1\}$ with [5]:

1. Deterministic dynamics of a constant velocity model, $\frac{\mathrm{d}\theta_t}{\mathrm{d}t} = p_t$ and $\frac{\mathrm{d}p_t}{\mathrm{d}t} = 0$.
2. Rate of events $\lambda(\theta, p) = \max\left\{0, -p\frac{\mathrm{d}\log\pi(\theta)}{\mathrm{d}\theta}\right\}$
3. At an event, the velocity component of the state flips.

In this simple example, the invariant distribution is $\tilde{\pi}(\mathbf{z}) = \pi(\theta)\pi_p(p)$. with π_p the probability mass function on $\{-1, 1\}$. Under the invariant distribution, p is independent of θ and has a uniform distribution. Unless there is a region where $\pi(\theta) = 0$ therefore separating two regions with positive probability, causing reducibility, then for this simple example, the invariant distribution will be equal to the stationary distribution.

Whilst this works for $d = 1$, in practice, we want to use MCMC to sample a target density in higher dimensions. For us to do this, there are many families of PDMPs, which differ for $d > 1$, however all reduce to the univariate PDMP described above for $d = 1$. Each of these families differ in the possible values for the velocity, the event rate, and transition kernel, however they are always chosen so $\mathbf{p}$ is independent of position $\boldsymbol{\theta}$, and the invariant distribution has a $\boldsymbol{\theta}$-marginal of $\pi(\boldsymbol{\theta})$.

3 The Bouncy Particle Sampler

One example of these samplers is the Bouncy Particle Sampler [14]. This was first used to simulate particle systems, but can also be used as a general sampling algorithm [4]. It is the continuous time limit of the Discrete Bouncy Particle Sampler [15], and at events, the transitions are reflections of the velocity in contours of $\log\pi(\boldsymbol{\theta})$.

3.1 Reflections

To define the event transition of a reflection, we can let $\hat{\mathbf{g}} = \mathbf{g}/(\mathbf{g}\cdot\mathbf{g})^{1/2}$ be a unit vector in the direction $\mathbf{g}$, for a d-dimensional vector. We can then define the reflection of $\mathbf{p}$ in the hyperplane perpendicular to $\mathbf{g}$ as $\mathcal{R}_{\mathbf{g}}(\mathbf{p}) = \mathbf{p} - 2(\mathbf{p}\cdot\hat{\mathbf{g}})\hat{\mathbf{g}}$.

This definition means that a reflection has two important properties. The first is that it preserves the size of the vector being reflected. The second is that it is a involution, meaning $\mathcal{R}_{\mathbf{g}}(\mathcal{R}_{\mathbf{g}}(\mathbf{p})) = \mathbf{p}$.

3.2 Form of the PDMP

For a refresh rate $\lambda_r \geq 0$, the Bouncy Particle Filter is a PDMP which has constant velocity dynamics and an event rate of $\lambda_{\mathrm{BPS}}(\boldsymbol{\theta}, \mathbf{p}) = \max\{0, -\mathbf{p}\cdot\nabla_{\boldsymbol{\theta}}\log\pi(\boldsymbol{\theta})\} + \lambda_r$ [4]. At an event, the velocity is reflected in the hyperplane perpendicular to $\nabla_{\boldsymbol{\theta}}\log\pi(\boldsymbol{\theta})$, with a probability of $1 - \lambda_r/\lambda_{\mathrm{BPS}}(\boldsymbol{\theta}, \mathbf{p})$. Otherwise, a new velocity is chosen by sampling from a standard normal distribution and this is referred to as a refresh event.

The value of λ_r needs to be tuned carefully. If $\lambda_r \approx 0$, the sampler could mix poorly, however if this value is too large, the random walk behaviour we wish to avoid will be introduced, also leading to poor mixing.

4 Using a Bouncy Particle Sampler to Sample from a Gaussian Target

When using a Bouncy Particle Sampler, we need to tune the hyperparameter λ_r which controls the rate of the refresh event occurring. If we set this either too low or too high, then this method does not perform well. We can look at the affect of tuning this by using this method on a standard two dimensional Gaussian distribution.

4.1 Deriving Times Until Events

Before we can run the sampler on this distribution, we first need to derive the equations for the time that each of the two events (refresh and bounce) will next occur.

4.1.1 Refresh Events

To calculate the time until the next refresh event, we first need to generate an independent realisation of a standard exponential random variable, w_1. Our time until the next refresh rate τ_1 is then $\tau_1 = w_1/\lambda_r$, with λ_r needing to be tuned. If a refresh event occurs, we then simulate our new velocity $\mathbf{p}'$ from the invariant distribution (in this example it is a standard Gaussian).

4.1.2 Bounce Events

We can define the rate at which the next bounce event occurs in terms of future time t, as $\tilde{\lambda}_{\mathbf{z}}(t) = \max\{0, \mathbf{p}^\top \mathbf{Q}\boldsymbol{\theta} + t\mathbf{p}^\top \mathbf{Q}\mathbf{p}\}$. Viewing $\tilde{\lambda}_{\mathbf{z}}(t)$ as a function of time until event t, then $\tilde{\lambda}$ is the maximum of a linear function of t, and zero [8]. Setting $a = \mathbf{p}^\top \mathbf{Q}\boldsymbol{\theta}$ and $b = \mathbf{p}^\top \mathbf{Q}\mathbf{p}$, and letting w_2 be an independent realisation of a standard exponential random variable, we have $\tau_2 = \sqrt{2w_2/b} + |a|/b$ for $a < 0$ and $\tau_2 = -a/b + \sqrt{a^2 + 2w_2 b}/b$ otherwise. If a bounce event occurs, the velocity becomes $\mathbf{p}' = \mathbf{p} - 2(\mathbf{p}^\top \mathbf{Q}\boldsymbol{\theta}')\frac{\mathbf{Q}\boldsymbol{\theta}'}{\boldsymbol{\theta}'^\top \mathbf{Q}^\top \mathbf{Q}\boldsymbol{\theta}'}$.

To calculate which of the two types of events occur first, we can then set the next event time as $t = \min\{\tau_1, \tau_2\}$.

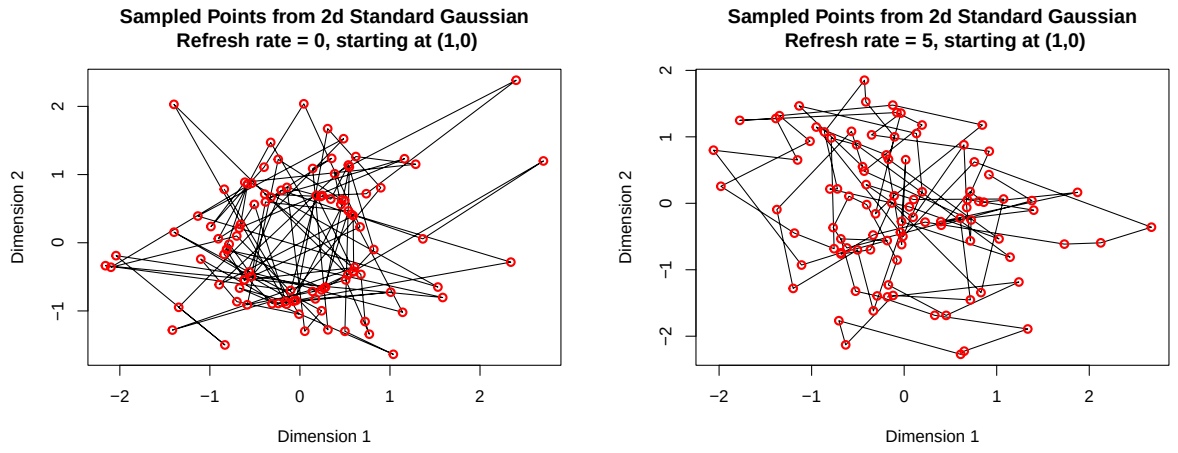
4.2 Tuning the Refresh Rate

4.2.1 Unwanted Behaviour of an Untuned Refresh Rate

For the 2-dimensional standard uncorrelated Gaussian we can first look at what happens if we set our refresh rate λ_r either too low or too high. If $\lambda_r = 0$ then a refresh event can never occur, and if $\lambda_r = 5$ then this event will happen most of the time meaning that the bounce event which tries to reduce the random walk behaviour will occur less [7]. We can see these behaviours by running the sampler for 1000 events, starting at $\boldsymbol{\theta} = (1, 0)$ and $\mathbf{p} = (1, 1)$, and then taking 100 samples from equally spaced time points to get our sample.

Figure 1 shows the unwanted behaviour that these two refresh rates produce. For $\lambda_r = 0$, whilst the means for the two variables are 0.01593 and 0.0437 and the effective sample sizes are 99 and 66 respectively, when looking at the positions of the sampled points we can see that the behaviour is not what we would expect. This can be seen by none of the sample points having both variables being close to 0 at the same time. However points in this location would be expected when sampling from a normal distribution with mean 0.

On the other hand, if we let $\lambda_r = 5$, we can see that the location of the sampled points are distributed more like we would expect them to be, with means of -0.0315 and -0.0955 for each variable respectively. However, the effective sample sizes for each of the two dimensions are 21.3 and 8.06, which are much lower than the number of points we have sampled. This shows that the mixing becomes worse as the refresh rate increases, as the behaviour becomes more like a random walk, similar to what occurs in the reversible MCMC techniques. The Autocorrelation Function (ACF) is also a lot worse when $\lambda_r = 5$, as



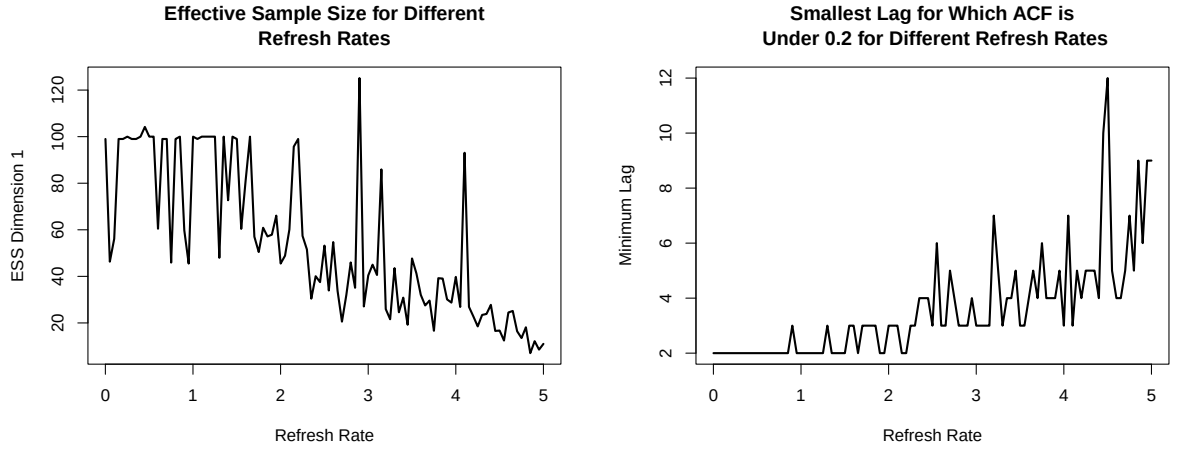
(a) (b)

Figure 1: Sampled Points From a Bouncy Particle Sampler, with $\boldsymbol{\theta} = (1, 0)$, for (a) $\lambda_r = 0$ and (b) $\lambda_r = 5$.

it takes 16 sampled points to get to get an autocorrelation value which is insignificant (below 0.2). This problem does not occur for $\lambda_r = 0$ as this ACF is below 0.2 for two neighbouring points.

4.2.2 Tuning the Refresh Rate

Given the examples of what can happen if we do not tune λ_r correctly, we need to be able to find what values of the refresh rate perform well. Two important values that can help us decide what refresh rate performs well are the effective sample size and the ACF. Running experiments with the same parameters as in section 4.2.1, for multiple values of λ_r between 0 and 5 can show how these two values vary for different refresh rates.



(a) (b)

Figure 2: (a) Effective Sample Size and (b) Minimum Lag such that ACF < 0.2 for $0 < \lambda_r \leq 5$.

Figure 2 shows how these values change for each of these measures as the refresh rate increases. For the effective sample size, we want this to be close to 100 which is the number of sampled points we have. This suggests that we would want $\lambda_r < 0.75$ as after this the effective sample size decreases in some instances. The graph also shows that if the refresh rate is set too close to 0, the effective sample size also decreases, suggesting that we do not want it to be too small, however this could also just be noise

which could be removed if the sampler is run multiple times and then the effective sample size values are averaged.

On the other hand, we want the smallest lag for which the autocorrelation is insignificant to be as low as possible as this means there is not much dependency between consecutive samples, and this is one of the problems that MCMC algorithms with reversibility have. Figure 2 suggests that a refresh rate value of under 0.9 would be suitable for this.

4.2.3 Optimal Refresh Rate Value

It has been shown that the optimal value of λ_r is one that makes the ratio between refresh events and bounce events approximately 0.78 [15]. This ratio can be calculated for each of the experiments by counting the number of times each type of event occurs. We expect this ratio to be small for smaller values of λ_r as this means that refresh events are likely to be infrequent, and then the ratio will increase as λ_r increases as the probability of an event being a refresh rate increases.

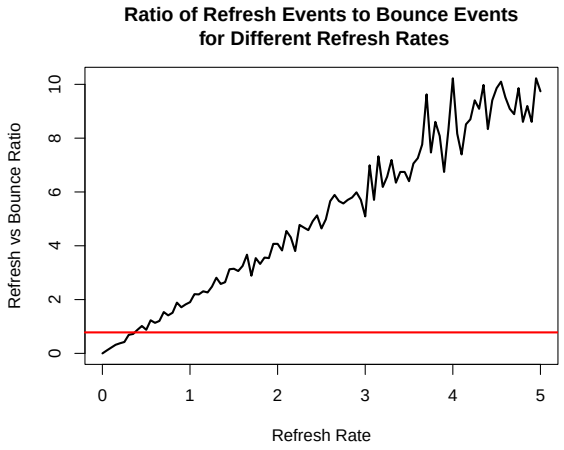


Figure 3: Ratio Between Refresh and Bounce Events with the red line representing the theoretical optimal ratio.

Figure 3 shows how this ratio increases and the red line shows that to achieve a ratio of 0.78, then a refresh rate of $\lambda_r = 0.5$ is wanted. This value is consistent with the range of λ_r values that seemed reasonable using the information in Figure 2.

5 Conclusions and Further Work

If we make sure that the hyperparameter λ_r is tuned correctly, then using a non-reversible MCMC algorithm such as the Bouncy Particle Sampler performs better than more traditional algorithms such as Metropolis-Hastings or Gibbs sampler. If however, the value is too low, there is not enough mixing, and there are areas that are not able to be sampled from, even if they are highly likely to occur in the distribution we are wishing to model. On the other hand, setting λ_r too high leads to poor mixing and a low effective sample size due to the behaviour being similar to that of a reversible sample which exhibits random walk characteristics, which we wish to avoid. This means that whilst the Bouncy Particle Sampler can perform well for MCMC, the choice of hyperparameter is important which is an added complexity which is not needed in samplers such as a Gibbs Sampler.

There are many different non-reversible samplers that could also be used, for example the Boomerang Sampler [2] and the Zig-Zag Sampler [3]. Each of these have the same behaviour in one dimension, however vary when the number of dimensions are increased. The performance of these different methods could be

compared to see which works the best depending on the distribution and the number of dimensions of the distribution we wish to sample from, especially if some of the methods are less sensitive to having a correctly tuned hyperparameter.

References

[1] J Besag. In Discussion of 'Representations of Knowledge in Complex Systems' by U. Grenander and M. Miller. *Miller. JR Statist. Soc. B*, 56:591–2, 1994.

[2] Joris Bierkens, Sebastiano Grazzi, Kengo Kamatani, and Gareth Roberts. The Boomerang Sampler. In *International conference on machine learning*, pages 908–918. PMLR, 2020.

[3] Joris Bierkens and Gareth Roberts. A Piecewise Deterministic Scaling Limit of Lifted Metropolis–Hastings in the Curie–Weiss Model. 2017.

[4] Alexandre Bouchard-Côté, Sebastian J Vollmer, and Arnaud Doucet. The Bouncy Particle Sampler: A Nonreversible Rejection-Free Markov Chain Monte Carlo Method. *Journal of the American Statistical Association*, 113(522):855–867, 2018.

[5] Mark HA Davis. Piecewise-Deterministic Markov Processes: A General Class of Non-Diffusion Stochastic Models. *Journal of the Royal Statistical Society: Series B (Methodological)*, 46(3):353–376, 1984.

[6] Persi Diaconis, Susan Holmes, and Radford M Neal. Analysis of a Nonreversible Markov Chain Sampler. *Annals of Applied Probability*, pages 726–752, 2000.

[7] Paul Fearnhead, Joris Bierkens, Murray Pollock, and Gareth O Roberts. Piecewise Deterministic Markov Processes for Continuous-Time Monte Carlo. *Statistical Science*, 33(3):386–412, 2018.

[8] Paul Fearnhead, Christopher Nemeth, Chris J Oates, and Chris Sherlock. Scalable Monte Carlo for Bayesian Learning. *arXiv preprint arXiv:2407.12751*, 2024.

[9] Stuart Geman and Donald Geman. Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (6):721–741, 1984.

[10] Paul Gustafson. A Guided Walk Metropolis Algorithm. *Statistics and Computing*, 8:357–364, 1998.

[11] W Keith Hastings. Monte carlo Sampling Methods Using markov Chains and Their Applications. 1970.

[12] Peter AW Lewis and Gerald S Shedler. Simulation of Nonhomogeneous Poisson Processes with Degree-Two Exponential Polynomial Rate Function. *Operations Research*, 27(5):1026–1040, 1979.

[13] Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[14] Elias AJF Peters and G de With. Rejection-Free Monte Carlo Sampling for General Potentials. *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics*, 85(2):026703, 2012.

[15] Chris Sherlock and Alexandre H Thiery. A Discrete Bouncy Particle Sampler. *Biometrika*, 109(2):335–349, 2022.