Monte Carlo Simulation
The Bouncy Particle Sampler and The Zig-Zag Sampler

Jiaming Hu

Department of Computational and Applied Mathematics,
University of Chicago

Abstract

This paper dedicates to the Bouncy sampler and the Zig-Zag sampler. They are two algorithms for continuous-time Monte Carlo. We will go over the motivation and concepts of those two algorithms, and then illustrates two examples for each sampler. The testable codes are attached to the end of this paper as auxiliary code. The Bouncy Sampler was adopted from reference [3] and the Zig-Zag Sampler was adopted from reference [1]. The Cinlar's Method used to sampler from the Poisson Process was adopted from reference [5] and the Poisson Thinning Method was adopted from reference [6].

1 Introduction

The widely usage of Markov Chain Monte Carlo algorithms makes the computation and simulation about distributions particularly easy. The MC methods are widely adopted in statistical worlds, especially in Bayesian computations where people have to calculate the posterior distribution of some kind. However, all commonly used methods are majorly developed based on the Metropolis-Hastings Algorithm, which MH algorithm was developed over 60 years ago itself (Bierkens(2019)). With the development of the internet, the access so called BIG DATA has been more easier than ever, hence seriously challenging the traditional MCMC that were developed and adopted by many people. First of all, it has been shown that non-reversible chains offers greater performance and have better conceptual advantages over reversible chains, which latter was the one that most MCMC algorithms use. One of our discussed algorithms in this quarter, namely the Hamiltonian MCMC, was developed partially under this motivation to escape form reversible chains. Secondly, it has been widely accepted that traditional MCMC with MH is of inefficient when it comes to massive data-set, because it essentially has to process the whole data-set at each iteration to calculate the likelihood, and we all know that iteration number in MCMC could typically go large.
In this paper, we will introduce two different algorithms, namely the (basic) Bouncy Sampler and the (basic) Zig-Zag sampler. Those two algorithms use non-reversible chains and operates under continuous-time Markov Chain. In Section 2 we refresh the concept of Poisson Process and establish two methods on how to sample from a Poisson Process given some rate λ. In Section 3 and 4 we introduce the Bouncy sampler and the Zig-Zag sampler. Two examples of same kind will be derived and the results will be illustrated for each sampler. One obvious advantage of the Bouncy Particle Sampler and the Zig-Zag sampler is that their mixing rates are faster than traditional MCMC algorithms, because they use non-reversible Markov Chains.

2 The Poisson Process

2.1 Definition of a Poisson Process

In this section we will introduce the Poisson Process, which is one of the most important steps in establishing both the Bouncy sampler and the Zig-Zag sampler. It is also one of the differences between discrete time Markov Chain and the Continuous Time Markov Chain.

A Poisson Process is anytime when we have some events that occurs individually over time at random, but if we look at them as a group then we can analyze some average rate λ, then we have a Poisson Process. A typical example that has been used over and over again would be the number of cars at a moment at a traffic light at given moment. The Homogeneous Poisson Process ($\lambda = constant$) can be defined as the number of arrivals $N(t)$ in a finite interval of length t that follows the $Poisson(-\lambda t)$ distribution. Moreover, the number of arrivals $N(t_1, t_2)$ and $N(t_3, t_4)$ in non-overlapping intervals $(t_1 < t_2 < t_3 < t_4)$ are independent (Bernardita 2019). Our focus here will be on how to

sample from such process, given some parameter λ, or in non-Homogeneous case, when the rate $\lambda(t)$ is a function that changes its value over time t. We will introduce two algorithms to complete the task. The first one is the Cinlar's method and the second one is the Poisson Thinning method.

2.2 Sample from a Poisson Process

The first method we are going to introduce is called the Cinlar's Method. We first state the following theorem:

Theorem 1 (Cinlar, 1975): Let $\Lambda(t), t \geq 0$ be a positive-valued, continuous, non-decreasing function. Then the random variables $T_1, T_2, ...$ are event times corresponding to a non-homogeneous Poisson process with expectation function $\Lambda(t)$ if and only if $\Lambda(T_1), \Lambda(T_2), ...$are the event times corresponding to a homogeneous Poisson process with rate one.

Theorem 1 provides us with a straight forward way to generate from a non-homogeneous Poisson Process. We first generate a homogeneous Poisson Process with rate one, and the invert Λ to obtain the event time for a non-homogeneous process (Pasupathy 2011). The algorithm goes as the follows:

Algorithm 2.1 Cinlar's Method

1: Initialize s = 0
2: Generate u $\sim U(0,1)$.
3: Set $s \leftarrow s - log(u)$
4: Set $t \leftarrow$inf $\{v : \Lambda(v) \geq s\}$
5: Deliver t.
6: Go to Step (1).

The next algorithm is perhaps the most popular method for sampling from non-homogeneous Poisson Process, namely the Poisson Thinning algorithm. We first state another theorem.

Theorem 2: Consider a non-homogeneous Poisson Process with rate $\lambda_u(t)$, with $t \geq 0$. Suppose that $T_1, T_2, ..., T_n$ are random variables representing event times from the non homogeneous Poisson Process with rate function $\lambda_u(t)$, and lying in the fixed interval $(0, t_0]$. Let $\lambda(t)$ be a rate function such that $0 \leq \lambda(t) \leq \lambda_u(t), \forall t \in [0, t_0]$. If the i^{th} event T_i is independently deleted with probability $1 - \frac{\lambda(t)}{\lambda_u(t)}, fori = 1, ..., n$, then the remaining event times form a non homogeneous Poisson Process with rate function $\lambda(t)$ in $(0, t_0]$.

Theorem 2 motivates the Poisson Thinning algorithm as follows:

Algorithm 2.2 Non-Homogeneous Poisson Thinning Method

1: Choose $\bar{\lambda}$ such that $\lambda(t) \leq \bar{\lambda}$ for all $t \in [0, T]$.
2: Define the thinning probability $p(t) = \frac{\lambda(t)}{\bar{\lambda}}$.
3: Set $t = 0$.
4: Generate $U_1 \sim$ Uniform $(0, 1)$
5: Set $t = t - \frac{ln(U_1)}{\bar{\lambda}}$. If $t > T$, stop. Otherwise go to step 6.
6: Generate $U_2 \sim$ Uniform $(0, 1)$, independent of U_1.
7: If $U_2 \leq p(t)$, deliver t. Otherwise go to step 4.

Now we have two different algorithms to sample from a non-homogeneous Poisson Process. We will see how they perform under different scenarios and different dimensions in following sections. Note that each of them poss some difficulties, namely choosing values for $\Lambda(v)$ in Cinlar's Method and $\bar{\lambda}$ in Thinning Method could be problematic, but we will see how we deal with them with simple ideas in the following sections.

3 The Bouncy Particle Sampler

3.1 Bouncy Particle Sampler Description

Consider a distribution $p(x) \propto e^{-U(x)}$. The Bouncy Particle Sampler simulates a continuous piece-wise linear trajectory $\{x(t)\}_{t\geq 0}$, which can be interpreted as position. Each segment in the trajectory is specified by an initial position $x^{(i)} \in \mathbb{R}^d$, a length $\tau_{i+1} \in \mathbb{R}^+$ and a velocity $v^{(i)} \in \mathbb{R}^d$. We define $t_i = \sum_{j=1}^{i} \tau_j$ for $i \geq 1$, and set $t_0 = 0$ for convenience. It is intuitive that the position at time t is thus $x(t) = x^{(i)} + v^{(i)}(t - t_i)$ for $t \in [t_i, t_{i+1})$. Then $x_{i+1} = x_i + v^{(i)}\tau_{i+1}$. Then the length of these segments is governed by an non-homogeneous Poisson Process of a rate function $\lambda : \mathbb{R}^d \times \mathbb{R}^d \to [0, \infty)$ (Bouchard-Côté 2018):

$$\lambda(x, v) = \texttt{max}\{0, \langle \nabla U(x), v \rangle\} \tag{3.1}$$

Then the velocity after bouncing is given by (Bouchard-Côté 2018):

$$R(x)v = v - 2\frac{\langle \nabla U(x), v \rangle}{\|\nabla U(x)\|^2} \nabla U(x) \tag{3.2}$$

Where $\|\cdot\|$ denotes the Euclidean norm and $\langle \cdot, \cdot \rangle$ denotes the dot product of two vectors.

Now we are ready to define the (basic) Bouncy particle Sampler:

Algorithm 3.1 Bouncy Particle Sampler

1: Initialize $(x^{(0)}, v^{(0)})$ arbitrarily on $\mathbb{R}^d \times \mathbb{R}^d$ and let T denote the requested trajectory length.
2: Simulate the first arrive time $\tau_{bounce} \in (0, \infty)$ of a Poisson Process of intensity

$$\chi(t) = \lambda(x^{(i-1)} + v^{(i-1)}t, v^{(i-1)}). \tag{3.3}$$

3: Simulate $\tau_{ref} \sim \text{Exp}(\lambda^{ref})$
4: Set $\tau_i \leftarrow \min(\tau_{bounce}, \tau_{ref})$ and compute the next position using

$$x^{(i)} \leftarrow x^{(i-1)} + v^{(i-1)}\tau_i \tag{3.4}$$

5: If $\tau_i = \tau_{ref}$, sample the next velocity $v^{(i)} \sim \mathcal{N}(0_d, I_d)$.
6: If $\tau_i = \tau_{bounce}$, compute the next velocity $v^{(i)}$ using

$$v^{(i)} \leftarrow R(x^{(i)})v^{(i-1)} \tag{3.5}$$

7: Repeat until desire.

Now note that from step 3 to step 6 are trivial to deal with, but the difficulty lies on step 2, where we have to simulate from a non-homogeneous Poisson Process, which the rate λ is the $\lambda(x, v)$ function we defined above. We here will use Algorithm 2.2 to sample from this Poisson Process. The major obstacle about implementing Algorithm 2.2 is how to choose $\bar{\lambda}$ such that $\lambda(t) \leq \bar{\lambda}$ for all $t \in [0, T]$. In the experiments I did I figured out a very simple way to choose such $\bar{\lambda}$, which is using the optimize function in R. Since we already have x and v at each step as parameters to $\lambda(x, v)$, then we can use optimize function to find the optimal value for the function $\lambda(x, v)$ and set it to $\bar{\lambda}$. Now people could come up with better value for $\bar{\lambda}$, and the efficiency of Algorithm 2.2 does dependent on the choice of $\bar{\lambda}$, but in our case, such method should be suffice, and the algorithm is rather efficient (in my case, generating 10,000 samples for the target distribution took only about 2 seconds).

3.2 One Dimensional Example

Here we will illustrate a 1-d example to get some flavor of the Bouncy Particle Sampler. We start off by with testing on $\mathcal{N}(10, 9)$. Note that the density function for $\mathcal{N}(\mu, \sigma)$ is

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} \propto e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} \tag{3.6}$$

and we can see that $U(x) = \frac{1}{2}(\frac{x-\mu}{\sigma})^2$ in this case, so $\nabla U(x) = \frac{x-\mu}{\sigma}$. With everything plugging into Algorithm 3.1 embedded with Algorithm 2.2, 10,000 samples for $\mathcal{N}(10, 9)$ was obtained and shown below in Figure 1:

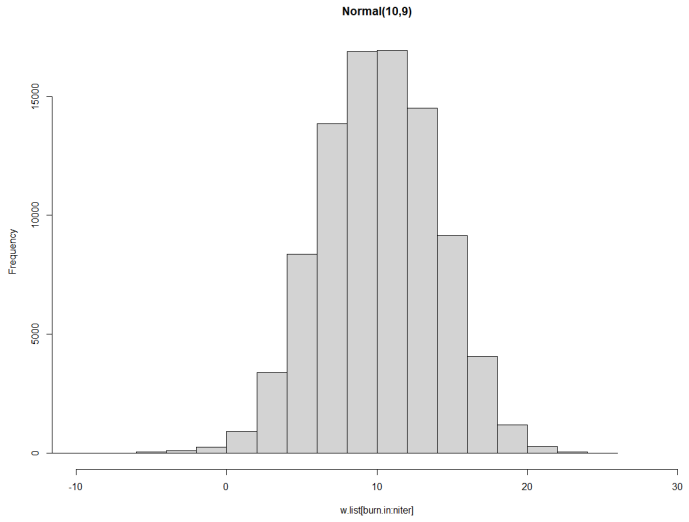

Figure 1 Results of Bouncy Particle Sampler with $\mathcal{N}(10, 9)$

We can see the result is very promising in a sense that it is close to the true distribution of $\mathcal{N}(10, 9)$. Note that Algorithm 2.2 also requires us to choose value for T for $t \in [0, T]$. In my experience there is no rule for that and the choice is rather empirical. Larger T usually resulting in longer runtime and smaller T usually resulting in losing precision in samples, so there is a trade-off. In this experiment I used $T = 100$ and the resulting samples were accurate.

3.3 Two Dimensional Example

Here we illustrate another example but in 2-d, namely, we will use BPS to sample from a Bi-variate Normal with $\mu = \begin{bmatrix} 10 \\ 5 \end{bmatrix}$ and $\Sigma = \begin{pmatrix} 3 & 0 \\ 0 & 3 \end{pmatrix}$. Now the PDF for Bi-variate Normal distribution is

$$f(x, y) \propto exp\Big(-\frac{1}{2(1-\rho^2)}\Big[\Big(\frac{x-\mu_X}{\sigma_X}^2\Big) - 2\rho\Big(\frac{x-\mu_X}{\sigma_X}\Big)\Big(\frac{y-\mu_Y}{\sigma_Y}\Big) + \Big(\frac{y-\mu_Y}{\sigma_Y}^2\Big)\Big]\Big) \tag{3.7}$$

So this gives that

$$U(x, y) = \frac{1}{2(1-\rho^2)}\left[\left(\frac{x-\mu_X}{\sigma_X}\right)^2 - 2\rho\Big(\frac{x-\mu_X}{\sigma_X}\Big)\Big(\frac{y-\mu_Y}{\sigma_Y}\Big) + \left(\frac{y-\mu_Y}{\sigma_Y}\right)^2\right] \tag{3.8}$$

The gradient $\nabla U(x, y)$ is easy to calculate, which is

$$\nabla U(x,y) = \begin{bmatrix} \left(\frac{x-\mu_X}{\sigma_X^2}\right) - \rho\left(\frac{y-\mu_Y}{\sigma_X\sigma_Y}\right) \\ \left(\frac{y-\mu_Y}{\sigma_Y^2}\right) - \rho\left(\frac{x-\mu_X}{\sigma_Y\sigma_X}\right) \end{bmatrix} \tag{3.9}$$

Again we use BPS paired with Algorithm 2.2 with everything plugged in to sample 10,000 for this Bi-variate Normal distribution, the result is showing in Figure 2 below.

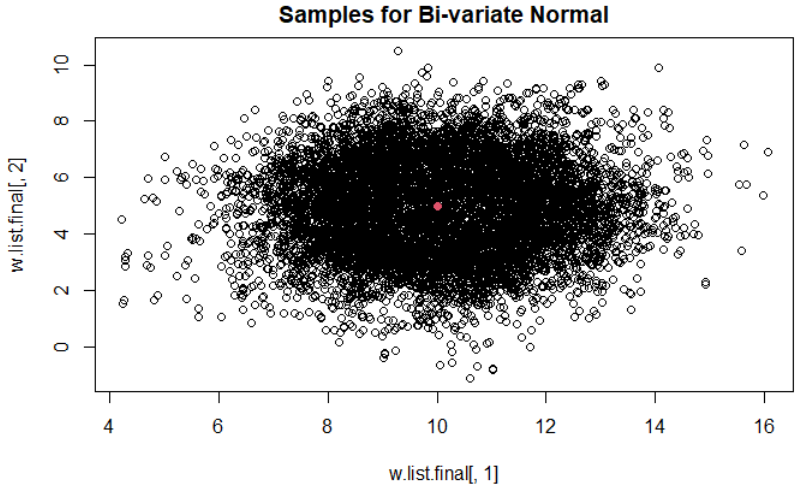

Figure 2 Results of Bouncy Particle Sampler with Bi-Variate Normal

The figure shows that the resulting samples are a good representation of the indicated Bi-variate Normal, where the red dot in the middle of the cluster is the true mean. The distribution could also be correlated, I included the this case in the auxiliary codes as comment, one could test it out as desired.

4 The Zig-Zag Sampler

4.1 Zig-Zag Sampler Description

We next introduce a sampler that is very similar to the Bouncy Particle Sampler, which is the Zig-Zag Sampler. In contrast, the major difference between the two is that the Zig-Zag Sampler has its velocity components v only takes value of either 1 or -1, and the piece-wise linear trajectories only change its direction in a single coordinate at each random breaking point (Pakman 2017). In other words, $v^{(i)}$ and $v^{(i+1)}$ only differs in one component, if one iteration ended up changing the velocity.

We first define the following function for flip of a single component of a vector x:

$$F_{i_0}(x) = \begin{cases} -x_i, & \text{if } i = i_0 \\ x_i, & \text{otherwise} \end{cases}$$

Algorithm 4.1 The Zig-Zag Sampler

1: Initialize $(\Xi^{(0)}, \Theta^{(0)})$ arbitrarily with $T^{(0)} = 0$ for convenience.
2: For $k = 1, 2, ...,$ define $m_i(t) = \lambda_i(\Xi^{(k-1)} + \Theta^{(k-1)}t, \Theta^{(k-1)})$ for $t \geq 0$ and $i = 1, ..., d$ where d is the dimension of the data.
3: For $i = 1, 2, ..., d$, let (M_i) denote the computational bounds for (m_i).
4: Draw $\tau_1, \tau_2, ..., \tau_d$ such that

$$\Pr(\tau_i \geq t) = exp\Big(-\int_0^t M_i(s)ds\Big) \tag{4.1}$$

5: Define $i_0 = \text{argmin}_{i=1,2,...,d}\{\tau_i\}$ and $\tau = \tau_{i_0}$.
6: Set

$$(T^{(k)}, \Xi^{(k)}) = (T^{(k-1)} + \tau, \Xi^{(k-1)} + \Theta^{(k-1)}\tau) \tag{4.2}$$

7: With probability $\frac{m_{i_0}(\tau)}{M_{i_0}(\tau)}$, $\Theta^{(k)} = F_{i_0}\Big[\Theta^{(k-1)}\Big]$, otherwise, $\Theta^{(k)} = \Theta^{(k-1)}$.

Note that comparing with BPS, the Zig-Zag Sampler is very similar in a way that it uses non-homogeneous Poisson Process (at step 4) as well as the same $\lambda(x, v)$ function (at step 2). But as noted before, the component of velocity only changes one component at a time (if it changes). The major difficulty besides sampling from the Poisson Process is the choice of the upper bound (M_i) at step 3, and it influence the efficiency of of the sampler. In our experiment, choosing $(M_i) = (m_i)$ should suffice, but in other scenarios, it might poss a difficulty. This time we will use the Cinlar's Method (Algorithm 2.1) to deal with sampling from the Poisson Process.

4.2 One Dimensional Example

Here we illustrate using the Zig-Zag Sampler on $\mathcal{N}(10, 9)$. As same as in the Bouncy Particle Sampler, the major difficulty for the Zig-Zag Sampler lies at step 4 where we need to sample from a non-homogeneous Poisson Process. We here will derive the details for the Cinlar's Method.

The function $U(x)$ and $\nabla U(x)$ is the same as in Section 3.2, then

$$m(t) = \max\left\{0, \frac{\Theta^{k-1}\Xi^{k-1}}{\sigma^2} + \frac{t}{\sigma^2}\right\} \tag{4.3}$$

So let $c_1 = \frac{|\Theta^{k-1}\Xi^{k-1}|}{\sigma^2}$ and $c_2 = \frac{1}{\sigma^2}$ and then

$$M(t) = c_1 + c_2 t \tag{4.4}$$

should meet the requirement. Then solving step 4 using Cinlar's Method becomes

$$t = \inf\{v : \Lambda(v) \geq s\} = \inf\{v : c_1 v + \frac{c_2 v^2}{2}\} = \frac{\sqrt{c_1^2 + 2c_2 s} - c_1}{c_2} \tag{4.5}$$

which is a close form solution and very trivial to deal with. So we could replace step 4 in Algorithm 4.1 with equation 4.5 and the specific algorithm becomes the following.

Algorithm 4.2 The Zig-Zag Sampler for $\mathcal{N}(10, 9)$

1: Initialize $(\Xi^{(0)}, \Theta^{(0)})$ arbitrarily with $T^{(0)} = 0$ for convenience.
2: Define $m(t) = \max\Big\{0, \frac{\Theta^{k-1}\Xi^{k-1}}{\sigma^2} + \frac{t}{\sigma^2}\Big\}$.
3: Define $M(t) = c_1 + c_2 t$ with $c_1 = \frac{\left|\Theta^{k-1}\Xi^{k-1}\right|}{\sigma^2}$ and $c_2 = \frac{1}{\sigma^2}$.
4: Simulate $u \sim U(0, 1)$ and set $s = -log(u)$
5: Define $\tau = \frac{\sqrt{c_1^2 + 2c_2 s} - c_1}{c_2}$
6: Set

$$(T^{(k)}, \Xi^{(k)}) = (T^{(k-1)} + \tau, \Xi^{(k-1)} + \Theta^{(k-1)}\tau) \tag{4.6}$$

7: With probability $\frac{m(\tau)}{M(\tau)}$: $\Theta^k = -\Theta^{k-1}$, otherwise, $\Theta^k = \Theta^{k-1}$.
8: Repeat for $k = 1, 2, ..., K$

Note that in this way, the Zig-Zag Sampler can generate 10,000 samples from $\mathcal{N}(10, 9)$ within no time, and Figure 3 shows the resulting samples.

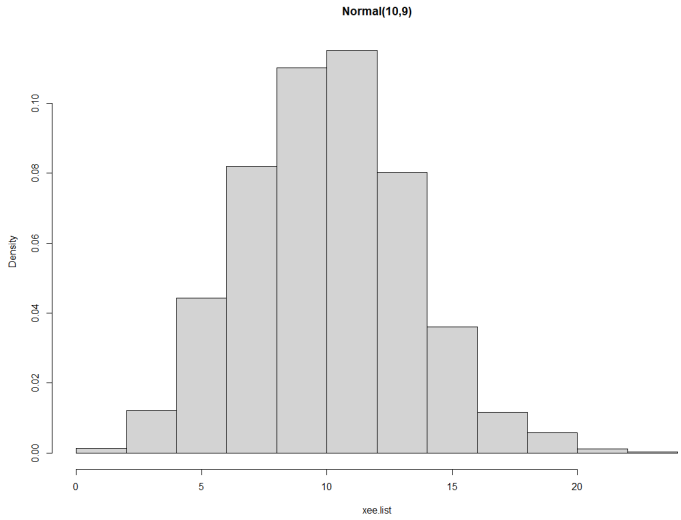

Figure 3 Results of Zig-Zag Sampler on $\mathcal{N}(10, 9)$

We can see the resulting sample gives a promising estimated PDF of $\mathcal{N}(10, 9)$, which is similar to Figure 1, so we know that both algorithms works well on this example.

4.3 Two Dimensional Example

We here illustrate the derivation and performance for the Zig-Zag Sampler on 2-d distribution, namely, the Bi-variate Normal distribution with the same parameters as in Section 3.3, so $U(x, y)$ and $\nabla U(x, y)$ would be the same. But note here since we need to choose m_i and M_i for different components of the trajectory, it would be easier if we

adopt the entry-wise notation instead of vector and matrix multiplication.

Let ξ denote the vector of interests (in our case $\xi = \begin{bmatrix} x \\ y \end{bmatrix}$) and let ξ_i denote each component of the vector, then one can show that

$$\frac{\partial U(\xi_i)}{\partial \xi_i} = \sum_{j=1}^{d} (\xi_j - \mu_j)\Sigma_{ij} \tag{4.7}$$

Then, we have that

$$m_i(t) = \lambda_i(\Xi^{k-1} + \Theta^{k-1}t, \Theta^{k-1}) = \max\left\{0, \Theta^{k-1}\sum_{j=1}^{d}\Sigma_{ij}\left(\Xi_j^{k-1} + \Theta_j^{k-1}t - \mu_j\right)\right\} \tag{4.8}$$

Then with a similar fashion as in Section 3.3, we can define

$$c_{1,i} = \sum_{j=1}^{d}|\Sigma_{ij}|\left|\Xi_j^{k-1} - \mu_j\right|, c_{2,i} = \sum_{j=1}^{d}|\Sigma_{ij}| \tag{4.9}$$

Then define

$$M_i(t) = c_{1,i} + c_{2,i}t \tag{4.10}$$

Then, with the same log in Section 3.3, one can show that from the Cinlar's Method we have

$$\tau_i = \frac{\sqrt{c_{1,i}^2 + 2c_{2,i}s_i} - c_{1,i}}{c_{2,i}} \tag{4.11}$$

where $s_i = -log(u_i)$ and $u_i \sim U(0,1)$ independently.

Now with everything in place, we can rewrite the Zig-Zag Sampler as the follows.

Algorithm 4.3 The Zig-Zag Sampler for Bi-variate Normal with indicated μ and Σ

1: Initialize $(\Xi^{(0)}, \Theta^{(0)})$ arbitrarily with $T^{(0)} = 0$ for convenience.
2: Set $m_i(t) = \max\left\{0, \Theta^{k-1}\sum_{j=1}^{d}\Sigma_{ij}\left(\Xi_j^{k-1} + \Theta_j^{k-1}t - \mu_j\right)\right\}$
3: Set $c_{1,i} = \sum_{j=1}^{d}|\Sigma_{ij}|\left|\Xi_j^{k-1} - \mu_j\right|, c_{2,i} = \sum_{j=1}^{d}|\Sigma_{ij}|$
4: Set $M_i(t) = c_{1,i} + c_{2,i}t$
5: Simulate $u_i \sim U(0,1)$ independently and set $s_i = -log(u_i)$
6: Define $\tau_i = \frac{\sqrt{c_{1,i}^2 + 2c_{2,i}s_i} - c_{1,i}}{c_{2,i}}$
7: Set $\tau = \min\tau_i, i_0 = \text{argmin}\tau_i$

$$(T^{(k)}, \Xi^{(k)}) = (T^{(k-1)} + \tau, \Xi^{(k-1)} + \Theta^{(k-1)}\tau) \tag{4.12}$$

8: With probability $\frac{m_{i_0}(\tau)}{M_{i_0}(\tau)}$: $\Theta^k = -\Theta^{k-1}$, otherwise, $\Theta^k = \Theta^{k-1}$.
9: Repeat for $k = 1, 2, ..., K$

In my experience, since we defined everything in simple calculations, the algorithm can generate 10,000 samples from the indicated Bi-variate Normal distribution within 1 second, and the resulting from the samples is shown in Figure 4 below.

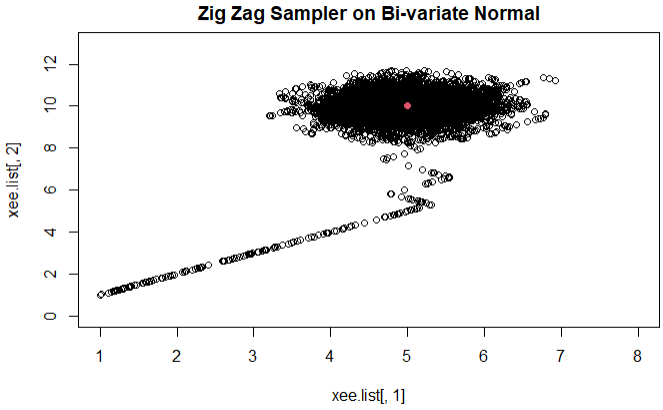

Figure 4 Results of Zig-Zag Sampler on indicated Bi-variate Normal distribution

One interesting thing to note comparing with the $\mathcal{N}(10, 9)$ example is that in Bi-variate case we can clearly see the zig-zag pattern from the path, and that is where the name of the algorithm comes from. Also we can see the result is accurate as both Figure 2 and 4 are showing the same thing. So the Zig-Zag Sampler works well in 2-d.

5 Discussion

The two algorithms are very similar in the way they use Poisson Process and they use the same $\lambda(x, v)$ function, and their generated results were also similar. But note that we used two different methods to sample from the Poisson Process, and that causes the efficiency for the two algorithms differ. If we use the same sampling method for both BPS and ZZ, I think their efficiency should be the same. Now I personally prefer the Cinlar's method because I was able to derive some closed form solution for each τ, but I doubt that would be the case if we increase the dimension to say, 10, then a closed form would be impossible. The Poisson Thinning Method has been very popular and widely used by people, as I have shown that a simple optimize function in R would meet the requirement for the choice of $\bar{\lambda}$, and people could be using much smarter ideas to choose such $\bar{\lambda}$. I think the two methods are similar to each other, it is rather how people sample from the Poisson Process makes the difference in efficiency.

The two continuous-time MC methods we discussed above in my experience is much more efficient than using the traditional MCMC algorithms we discuss in our lectures. The reason is that non-reversible Markov Chain could resulting in a much faster mixing rate. As I mentioned above, the run-time for the ZZ method in two examples was under 1 second, and BPS could also finish generating 10,000 samples within 2 seconds. So I think we should be using those two methods rather than traditional MCMC whenever the situation allows.

Reference

[1] Bierkens, J., Fearnhead, P., & Roberts, G. (2019). The Zig-Zag process and super-efficient sampling for Bayesian analysis of big data. The Annals of Statistics, 47(3). https://doi.org/10.1214/18-aos1715

[2] Bernardita. (2019). bernarditaried/ZigZag-Sampling-Tutorial. GitHub. https://github.com/bernarditaried Sampling-Tutorial/blob/master/ZigZag-Sampling-Tutorial.ipynb

[3] Bouchard-Côté, A., Vollmer, S. J., & Doucet, A. (2018). The Bouncy Particle Sampler: A Nonreversible Rejection-Free Markov Chain Monte Carlo Method. Journal of the American Statistical Association, 113(522), 855–867. https://doi.org/10.1080/01621459.2017.1294075

[4] Pakman, Ari, et al. "Stochastic Bouncy Particle Sampler." ArXiv:1609.00770 [Stat], June 2017. arXiv.org, http://arxiv.org/abs/1609.00770.

[5] Pasupathy, R. (2011). Generating Nonhomogeneous Poisson Processes. Https://Web.Ics.Purdue.Edu/. https://web.ics.purdue.edu/ pasupath/PAPERS/2011pasB.pdf

[6] Nonhomogeneous poisson process simulation. (2018, September 29). Cross Validated. https://stats.stackexchange.com/questions/369288/nonhomogeneous-poisson-process-simulation