On Event-Chain Monte Carlo Methods.

Nicholas Galbraith

Mansfield College
University of Oxford
September 2016

A dissertation submitted in partial fulfilment of the requirements for the degree of

Master of Science in Applied Statistics

Abstract

In this dissertation we consider a pair of continuous-time, non-reversible, rejection-free, and piecewise deterministic MCMC methods, referred to as Event-Chain Monte Carlo methods; respectively reflect-ECMC and flip-ECMC. We compare the two methods in a handful of settings, and find that in all cases the performance of the reflect-ECMC algorithm is superior. We consider extensions of the algorithms proposed in the context of large-scale Bayesian analysis, and combine various improvements proposed in the literature yielding a method which we demonstrate to outperform all previously considered methods for Bayesian logistic regression. Still in a Bayesian context, we show how the reflection algorithm scales in the limit as the number of observations $n \to \infty$, and find that - as was previously demonstrated for the flip algorithm - that it is possible, using a combination of sub-sampling and control variate ideas, to obtain a reflect-ECMC method for which the cost of obtaining an independent point is $O(1)$ in n. Furthermore, we present the first detailed discussion concerning the tuning of the parameters of these two methods, and we demonstrate empirically the considerable efficiency gains which are made possible by the use of a non-diagonal 'mass matrix' for the reflect algorithm; this we do using a real-data logistic regression example, and an example in which the target distribution is that of a latent field in a Poisson-Gaussian Markov random field model.

I dedicate this work

to Grandpa and Grand-papa,
who did not live to see me fully grown
- how I wish I could see you now.

to Grandma,
who would have been proud to send a grandson off to Oxford,
as she did her son many years ago;
whose unwavering cheer was constantly uplifting, until the very end,
- how dearly I miss you.

and, to Mormor,
whom I will call as soon as I submit,
- if my writing is not up to Desbarats standard,
I will try harder next time.

And finally, to my father,
who might (I hope) at last bury his fears that I should end up working at Canadian Tire.

Acknowledgements.

Special thanks to my supervisor Professor Arnaud Doucet, without whose direction I would not have discovered this beautiful topic, without whose invaluable guidance and insightful suggestions this work would not have been possible, and without whose conversation I might have forgotten how to speak French. Thanks also to my father; I am indebted to both he and Professor Doucet for having read an early draft and discovering many errors which had escaped my notice. Naturally, I bear full responsibility for any and all that remain.

Contents

"I am thinking of something much more important than bombs. I am thinking about computers."

- John von Neumann

1 Introduction.

Despite the (comparatively) recent explosion of interest in Markov chain Monte Carlo methods, heralded by the seminal papers Geman and Geman [16] and Gelfand and Smith [15] - the annals of history will testify to the fact that the first Markov-chain Monte Carlo (MCMC) algorithm was developed by physicists. Motivated by the need to simulate configurations of particle systems, in 1953 a group of researchers at the Los Alamos laboratories (including Nicholas Metropolis, the algorithm's namesake) employed a simple random-walk sampler to explore the distribution of the states [26]. An extension of this method proposed in Hastings [18] - known as the Metropolis-Hastings algorithm - has enjoyed widespread popularity and success, although it is not without its limitations. Hampered in practice by the slow exploration of the state space which results from random-walks, the MH method has been shown to be dramatically inferior in many applications to more sophisticated MCMC algorithms which employ some device to avoid random-walk behaviour. A well-known example being the Hamiltonian Monte Carlo (HMC) algorithm, another MCMC method introduced by physicists, originally proposed in Duane et al. [14], where it was successfully used for lattice field theory simulations of quantum chromodynamics; it was not until Neal [29, 30] that the method was brought to the attention of the statistical community - for an excellent review, see Neal [31]. HMC is an MCMC method which operates on a state space augmented to include velocity variables, the joint density of these and the variables of interest is expressed as a function of the Hamiltonian which encodes the total energy of the system. Leveraging knowledge of the gradient of the Hamiltonian, proposal moves are designed by approximating the Newtonian dynamics of the system and using a Metropolis-Hastings correction [31]; using such transition kernels markedly reduces the number of iterations needed to reach an independent point, and thus effectively suppresses random-walks [31].

In this dissertation, we will consider a novel type of MCMC algorithm - once again proposed by physicists. The algorithm was originally introduced in Peters and de With [33] and used to simulate molecular dynamics under general forms of pairwise potential energies, where its effectiveness was demonstrated in simulating a system governed by Lennard-Jones interactions. The method has since been successfully implemented in a range of other settings, such as hard-sphere systems, ferromagnetic Heisenberg models, continuous-spin systems, and many more; in each case showing marked efficiency gains over local random-walk Metropolis algorithms and often outperforming other state-of-the-art methods as well, see e.g. [22, 27, 28, 32, 33]. Despite the manifest utility and versatility of these event-chain Monte Carlo (ECMC) algorithms, no notice of them was taken by statisticians until Bouchard-Côté et al. [8] very recently expounded and generalized the

algorithm of Peters and de With [33]; even more recently, Bierkens et al. [7] propose a very similar method and elaborate on its remarkable properties.

Both the classical Metropolis-Hastings (MH) algorithm and HMC adhere to a common framework for constructing MCMC algorithms in which candidate moves are generated according to some proposal distribution and then accepted or rejected with probability given by the MH ratio, creating a discrete-time reversible Markov chain on the state space which converges to its invariant distribution which is by construction the target distribution of interest. By contrast, algorithms which we will consider break free from this restrictive paradigm. These ECMC methods exploit continuous-time Markov processes to generate a 'continuum of samples' from the distribution of interest; furthermore, they are non-reversible and rejection-free. The condition of detailed balance that is habitually invoked to demonstrate that particular MCMC samplers have the correct invariant distribution is broken, and proofs of correctness rely on showing that the weaker global balance condition is satisfied [33]. Simulation of the process is carried out by a simulation of a succession of events, in between which the process is deterministic; whence the name event-chain Monte Carlo (ECMC).

In this dissertation, we will present, compare, analyse, and where possible improve two ECMC algorithms: namely those proposed in Bouchard-Côté et al. [8] and Bierkens et al. [7], respectively. As mentioned above, these are non-reversible and rejection-free MCMC methods. Theoretical vindication of the use of non-reversible MCMC methods is well established, having been shown to yield significantly faster mixing Markov chains in some simple examples, see e.g. Diaconis et al. [13] or Hwang et al. [20]. Empirical results are in many cases equally encouraging - a small selection of examples include [22, 27, 28, 32, 33, 8, 7]. A humorous and yet perspicacious analogy which we take the liberty of quoting is drawn in Turitsyn et al. [38] between the use of non-reversible sampling and a real-life scenario with which many of us will no doubt be all too familiar: the mixing of a cup of coffee. They percipiently state it thus: "Consider mixing sugar into a cup of coffee, which is similar to sampling, as long as the sugar particles have to explore the entire interior of the cup. [Standard MCMC] dynamics corresponds to diffusion taking an enormous mixing time. This is certainly not the best way to mix; moreover, our everyday experience suggests a better solution - enhance mixing with a spoon. Spoon stirring... significantly accelerates mixing, while achieving the same result: uniform distribution of sugar concentration over the cup." The methods we consider employ the expedient of a 'lifted' state space - first introduced and analysed in Diaconis et al. [13] and generalized in Turitsyn et al. [38] - in which introducing variables which guide the dynamics of the non-reversible processes and curb the diffusive behaviour which is so detrimental to rapid mixing. In the continuous state-space settings which will be our focus, this lifting variables correspond to velocities which determine the speed and direction of motion through the support of the target. While similar in nature to the velocity variables of HMC, these are purely synthetic and no physical interpretation is forthcoming [38].

The structure of this paper is as follows: in Section 2, we introduce two ECMC meth-

ods which we refer to respectively as the reflection method (of Bouchard-Côté et al. [8]) and the flip method (of Bierkens et al. [7]). In Section 3, we discuss certain practical considerations involved in the use of these methods; of chief concern will be efficient methods of simulating the event times. In Section 4, we compare the empirical performance of these two methods in various simple scenarios in which the target follows a Gaussian distribution. In Section 5, we discuss the use of these methods for large-scale Bayesian analysis, and demonstrate using logistic regression as an example that large gains over the vanilla algorithms are possible using sub-sampling ideas, as first shown in both [8, 7]; furthermore, we show that the various improvements suggested in [8, 7] can be combined, yielding a strategy which outperforms all previous implementations. Additionally, we show that the arguments of [7] concerning the scaling for large numbers of data points are applicable in the context of the reflection algorithm, and we leverage this analysis to glean an understanding of the potential efficiency gains made possible by the 'informed' sub-sampling method introduced in [8]. In Section 6, we discuss the issue of tuning the various parameters of the two algorithms, and in particular give an indication by means of examples on synthetic and real data of the potential for large improvement - particularly in the context of the reflection algorithm. Finally, in Section 7, we present our conclusions, discuss the scope and limitations of the algorithms, and suggest directions for further research.

"Come Watson, come! The game is afoot."

- Sherlock Holmes, *The Adventure of the Abbey Grange*

2 Two Event-Chain Monte Carlo Methods.

Consider the general problem of drawing samples from a probability measure μ on $(\mathbb{R}, \mathcal{B}(\mathbb{R}^d))$ - that is, d-dimensional Euclidean space with the Borel σ-algebra - in order to evaluate expectations $\mathbb{E}_\mu[\phi] = \int \phi(x)\, d\mu$ of arbitrary functions ϕ: $\mathbb{R}^d \to \mathbb{R}$. In the ECMC framework, this is accomplished via the construction of a continuous-time Markov 'switching' process (see, e.g. [3]) on an extended state space which, as we shall see shortly, possesses the usual desired properties of invariance and ergodicity. For our purposes we may assume that μ admits a density with respect to the Lesbesgue measure which we will denote by π; thus:

$$\mu(dx) = \pi(x)\, dx;$$

furthermore, we assume that $\pi : \mathbb{R}^d \to \mathbb{R}$ is continuously differentiable. We let $U(x) = -\log \pi(x)$, which we refer to as the associated energy. In the following subsections, we describe two ECMC methods which marginally produce samples from π, first defining them through their generators, then describing informally how they evolve with time and finally giving an algorithmic description of how to simulate them. We follow the work of Bouchard-Côté et al. [8] and Bierkens et al. [7] respectively.

2.1 Reflection ECMC.

Consider the space $E_R = \mathbb{R}^d \times \mathbb{R}^d$, and let $C^1(E_R)$ denote the space of continuously differentiable real-valued functions on E_R. Let ψ be a density for a probability measure on $\mathbb{R}^d$, and let $\rho(x, v) = \pi(x)\psi(v)$ for $x \in \mathbb{R}^d$ and $v \in \mathbb{R}^d$ be a density on E_R. Now, for $h \in C^1(E_R)$ and $\lambda_0 \geq 0$, consider the stochastic process $\{\Xi(t)\}_{t\geq 0} = \{(X(t), V(t))\}_{t\geq 0}$ with infinitesimal generator given by

$$\mathcal{L}h(x, v) = \langle \nabla_x h, v\rangle + \lambda(x, v)\,(h(x, R[x]v) - h(x, v)) + \lambda_0 \int (h(x, s) - h(x, v))\psi(s)\, ds, \quad (1)$$

where $\langle \cdot, \cdot \rangle$ denotes the Euclidean inner product and $\nabla_x = \left(\frac{\partial}{\partial_{x_1}}, \ldots, \frac{\partial}{\partial_{x_d}}\right)$ denotes the gradient operator with respect to the x variables. Finally, $R[x]$ denotes the following reflection operator at x:

$$R[x]v = \left(I_d - 2\frac{\nabla U(x)\nabla U(x)^t}{\langle \nabla U(x), \nabla U(x)\rangle}\right) v = v - 2\frac{\langle \nabla U(x), v\rangle}{||U(x)||^2}\nabla U(x), \quad (2)$$

where I_d is the $d \times d$ identity matrix and $||\cdot||$ the Euclidean norm. This operator models an elastic collision of a particle of velocity v against an energy barrier orthogonal to the gradient vector $\nabla U(x)$, and is what drives the dynamics of the process, along with

the intensity function $\lambda(\cdot,\cdot)$ which determines the rate at which reflections occur; this is defined to be

$$\lambda(x,v) = (\langle \nabla U(x), v\rangle)^+ \tag{3}$$

where $(a)^+$ denotes, for $a \in \mathbb{R}$, the positive part of a, that is, $(a)^+ := \max(0,a)$. It can be shown that the operator in (1) is the generator of a piecewise-deterministic Markov process, which evolves linearly in between random 'switching' events, and satisfies the strong Markov property; see Davis [11].

Given an initial state $\Xi^{(0)} = \Xi(0) = (X(0), V(0)) = (x^{(0)}, v^{(0)}) \in E_R$, the process may be described as follows: for $t \in [0,\tau)$, the velocity remains constant while the position variables move in a straight line determined by v_0, thus $\Xi(t) = (x^{(0)} + tv^{(0)}, v^{(0)})$. The first event time $\tau^{(0)}$ is defined to be the minimum of τ_1, τ_2 - the first arrival times of two Poisson processes, respectively the first arrival of a homogeneous Poisson process with rate λ_0 and the first arrival of an inhomogeneous process with rate

$$\lambda(x(t), v(t)) = \lambda(x^{(0)} + tv^{(0)}, v^{(0)}) = \Big(\langle \nabla U(x^{(0)} + tv^{(0)}), v^{(0)}\rangle\Big)^+.$$

If $\tau^{(0)} = \tau_1$, then $x^{(1)} = x^{(0)} + \tau^{(0)} v^{(0)}$ and $v^{(1)} \sim \psi$ is drawn from its marginal distribution which will usually be an isotropic Gaussian or the uniform distribution on the $(d-1)$-sphere, so that $\Xi(\tau^{(0)}) = \Xi^{(1)} = (x^{(1)}, v^{(1)})$; in this case we say that $\tau^{(0)}$ is a 'refreshment' event. If $\tau^{(0)} = \tau_2$, then again $x^{(1)} = x^{(0)} + \tau^{(0)} v^{(0)}$, but now $v^{(1)} = R[x^{(1)}]v^{(0)}$ so that $\Xi(\tau^{(0)}) = \Xi^{(1)} = (x^{(1)}, v^{(1)})$, and we say that $\tau^{(0)}$ is a reflection event. The process now begins anew with initial state $\Xi^{(1)}$, yielding a sequence $\{\Xi^{(n)}, \tau^{(n)}\}_{n\geq 0}$ consisting of the event times and the corresponding values of the position and velocity; clearly it suffices to store only the (x,v) coordinates at the times when events occur, as the state at any intermediary time can easily be interpolated from them. Pseudocode for the algorithm is given in Algorithm 1 below.

The following result (Theorem 1 from [8]) allows us to use the reflection algorithm in practice.

Theorem 2.1 *For any $\lambda_0 \geq 0$, the Markov kernel associated to the generator in (1) is non-reversible with invariant distribution ρ, where $\rho(x,v) = \pi(x)\psi(v)$. Furthermore, if $\lambda_0 > 0$, then ρ is the unique invariant measure of the transition kernel specified by (1), and the corresponding process satisfies the following strong law of large numbers: for ρ-almost every $\Xi(0)$ and $h \in \mathrm{L}^1(\rho)$, we have that*

$$\lim_{T\to\infty} \frac{1}{T} \int_0^T h(\Xi(t))\, dt = \int_{E_R} h(\xi)\rho(\xi)\, d\xi \quad a.s.$$

Note that the condition $\lambda_0 > 0$ cannot be dropped - see [8] for an example where the reflection algorithm fails to produce an ergodic chain when $\lambda_0 = 0$. Figure 1 shows a sample trajectory from a reflection ECMC run.

Algorithm 1 Basic reflection algorithm

Arbitrarily initialize $(x^{(0)}, v^{(0)}) \in \mathbb{R}^d \times \mathbb{R}^d$.
Let $T = 0$.
for $i = 1, 2 \dots$ **do**
    Simulate $\tau_{reflect}$ as the first arrival time of a Poisson process of rate $\left(\langle \nabla U(x^{(i-1)} + tv^{(i-1)}), v^{(i-1)} \rangle\right)^+$.
    Simulate $\tau_{refresh} \sim \text{Exp}(\lambda_0)$.
    Set $\tau^{(i)} \leftarrow \min(\tau_{refresh}, \tau_{reflect})$.
    Set $x^{(i)} \leftarrow x^{(i-1)} + \tau^{(i)} v^{(i-1)}$.
    if $\tau^{(i)} = \tau_{refresh}$ **then**
        Set $v^{(i)} \sim \psi$.
    end if
    if $\tau^{(i)} = \tau_{reflect}$ **then**
        Set $v^{(i)} \leftarrow R[x^{(i)}]v^{(i-1)}$.
    end if
    Set $T \leftarrow T + \tau^{(i)}$.
    Return $(x^{(i)}, v^{(i)}, T)$.
end for

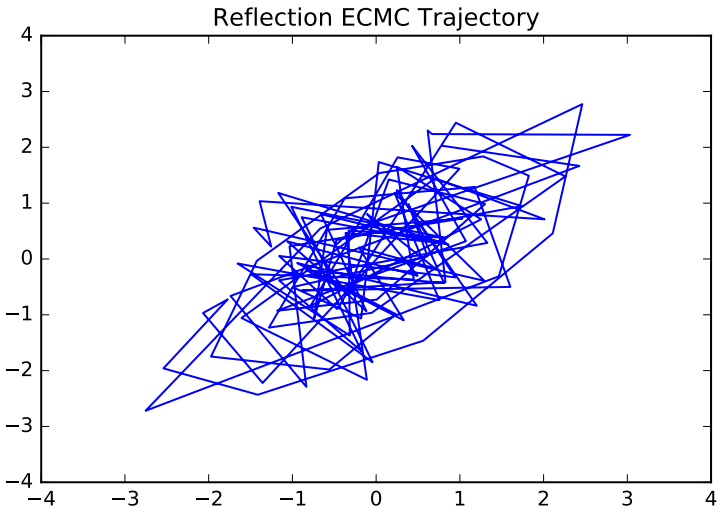

Figure 1: Trajectory constructed from 100 events from a reflection-ECMC algorithm with a bivariate-normal invariant distribution with mean $\mu = (0, 0)^T$; the marginal variances are both set to 1 and the two components have correlation equal to 0.6. The refreshment parameter was set to $\lambda_0 = 0.2$.

2.2 Flip ECMC.

Rather than using continuous velocities, the flip algorithm allows only a finite number of velocity vectors. Consider the space $E_F = \mathbb{R}^d \times \{-1, 1\}^d$, and let $C^1(E_F)$ denote the set of real-valued functions on E_F which are continuously differentiable in their first d arguments, i.e. $f \in C^1(E_F)$ if $f(\cdot, v)$ is continuously differentiable for each $v \in \{-1, 1\}^d$. Let ψ denote the density of the uniform distribution on $\{-1, 1\}^d$, so that $\rho_0(x, v) \coloneqq \psi(v)\pi(x) \propto \pi(x)$. Now, for $h \in C^1(E_F)$, consider the stochastic process $\{\Xi(t)\}_{t\geq 0} = \{(X(t), V(t))\}_{t\geq 0}$ with infinitesimal generator given by

$$\mathcal{L}h(x, v) = \langle \nabla_x h, v\rangle + \sum_{i=1}^{d} \lambda_i(x, v)\left(h(x, F_i[v]) - h(x, v)\right) \tag{4}$$

where $F_i[x]$ denotes the i-th flip operator at x:

$$\left(F_i[v]\right)_j \coloneqq \begin{cases} v_j & \text{if } i \neq j \\ -v_j & \text{if } i = j. \end{cases} \tag{5}$$

for $j = 1, \ldots, d$, and $\lambda_i(x, v)$ denotes the i-th flip rate, which is defined to be

$$\lambda_i(x, v) = (v_i \partial_i U(x))^+ + \gamma_i(x, v) \tag{6}$$

where $\gamma_i(x, v)$ is an arbitrary non-negative bounded function which satisfies $\gamma_i(x, v) = \gamma_i(x, F_i[v])$ and ∂_i is the partial derivative with respect to the i-th component. Just as was the case for the reflection algorithm, it can be shown (again, see Davis [11]) that the generator in (4) determines a piecewise-deterministic Markov process which is linear in between switching events and satisfies the strong Markov property. The trajectories of the process can be described in much the same way as those for the reflection algorithm; in between flipping events, the velocity is constant while the position is linear in t with $\frac{d}{dt}X(t) = V(t)$. In this case however, only one component of the velocity is altered when an event occurs, and it is simply reversed. Each dimension has an individual flipping rate, and the first arrival among the d point processes determines which component flips. Pseudocode for the algorithm is given in algorithm 2 below.

The following results allow us to use the flipping algorithm in practice (Theorems 2.2 and 2.11 from [7]).

Theorem 2.2 *The Markov kernel associated to the generator in* (4) *is non-reversible with invariant distribution ρ, where $\rho(x, v) \propto \pi(x)$. Furthermore, if the functions γ_i in* (6) *are positive and bounded everywhere, then ρ is the unique invariant measure of the transition kernel specified by* (4)*, and the corresponding process satisfies the following strong law of large numbers: for ρ-almost every $\Xi(0)$ and $h \in \mathrm{L}^1(\rho)$, we have that*

$$\lim_{T\to\infty} \frac{1}{T}\int_0^T h(\Xi(t))\, dt = \int_{E_F} h(\xi)\rho(\xi)\, d\xi \quad a.s.$$

Algorithm 2 Basic flipping algorithm

Arbitrarily initialize $(x^{(0)}, v^{(0)}) \in \mathbb{R}^d \times \mathbb{R}^d$.
Let $T = 0$.
for $i = 1, 2 \dots$ **do**
    for $j = 1, 2 \dots, d$ **do**
        Simulate τ_j as the first arrival time of a Poisson process of rate $\left(v_j^{(i-1)} \partial_j U(x^{(i-1)} + t v^{(i-1)})\right)^+$.
    end for
    Set $\tau^{(i)} \leftarrow \min_{j=1,2\dots d}(\tau_j)$.
    Set $x^{(i)} \leftarrow x^{(i-1)} + \tau^{(i)} v^{(i-1)}$.
    Set $v^{(i)} \leftarrow F_j(v^{(i-1)})$.
    Set $T \leftarrow T + \tau^{(i)}$.
    Return $(x^{(i)}, v^{(i)}, T)$.
end for

See Figure 2 for a sample trajectory from a flip ECMC run.

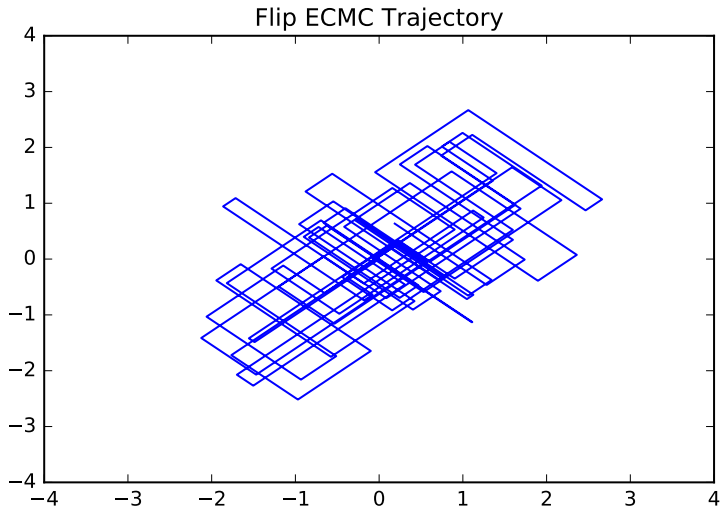

Figure 2: Trajectory constructed from 100 events from a flip-ECMC algorithm with a bivariate-normal invariant distribution with mean $\mu = (0, 0)^T$. The marginal variances are both set to 1 and the components have correlation equal to 0.6.

"When a coin is tossed, it does not necessarily fall heads or tails; it can roll away or stand on its edge."

- William Feller

3 Simulation in Practice.

The need to simulate the first arrival times of the inhomogeneous Poisson processes in Algorithms 1.4 and 2.5 is the only practical impediment to the implementation of the two algorithms outlined in the previous section. To simplify our notation in this section, we will suppress the dependence on the x, v variables and simply express the rates as functions of time, i.e. $\lambda(x(t), v(t)) = \lambda(t)$. Letting $\Lambda(t) = \int_0^t \lambda(t)\, dt$ denote the integrated rate function and τ the first arrival time, we have that

$$\mathbb{P}(\tau > t) = \exp\{-\Lambda(t)\} \tag{7}$$

and so we may simulate τ by letting

$$\tau = \Lambda^{-1}(-\log U) \tag{8}$$

where U is uniformly distributed on $(0, 1)$ and $\Lambda^{-1}(p) = \inf\{t : p \leq \Lambda(t)\}$ is the generalized inverse function. This inverse will usually not be analytically tractable, however there exist a number of methods which allow one to circumvent this problem. Perhaps the most useful is the thinning method due to Lewis and Shedler [23]:

Proposition 3.1 *Let $\lambda : \mathbb{R}^+ \to \mathbb{R}^+$ and $M : \mathbb{R}^+ \to \mathbb{R}^+$ be continuous functions such that $\lambda(t) \leq M(t)$ for $0 \leq t$. Suppose that $\tau_1, \tau_2, \ldots$ are a (finite or infinite) sequence of arrival times of a Poisson process with rate function $M(t)$. If for each $i = 1, 2 \ldots$ the point τ_i is deleted from the sequence with probability $\lambda(\tau_i)/M(\tau_i)$, then the remaining points correspond to the arrival times of a Poisson process with rate function $\lambda(t)$.*

This will be especially useful when we can find affine (or piecewise affine) bounds for the rate function, i.e. $\lambda(t) \leq a + bt = M(t)$ for some $a, b \in \mathbb{R}^+$, as in this case the inversion (8) will be available analytically for $M(t)$.

One scenario in which we need not rely on recourse to the above method is when the distribution of interest has a strictly log-concave density function. Observe that the inversion in (8) amounts to finding τ such that

$$\begin{aligned} \int_0^\tau \lambda(t)\, dt &= -\log U \\ \int_0^\tau \langle \nabla U(x + tv), v \rangle^+ &= -\log U \\ \int_0^\tau \left(\frac{dU(x + tv)}{dt} \right)^+ &= -\log U. \end{aligned}$$

Now, if f is strictly log-concave, then $-\log f$ is strictly convex, and so there exists a unique τ^* such that $\tau^* = \arg\min_{t\geq 0} U(x+tv)$. On $[0,\tau^*)$ (possibly empty) we have $dU/dt < 0$ and $dU/dt \geq 0$ on $[\tau^*,\infty)$, and so we have

$$\int_{\tau^*}^{\tau} \frac{dU(x+tv)}{dt}\,dt = U(x+\tau v) - U(x+\tau^* v) = -\log U. \tag{9}$$

In many cases this equation will be easily solvable; if not we may solve using line search with arbitrary precision. Frequently, we will use this method in conjunction with the above thinning method.

Another useful method for simulation is the superposition method ([8]). Supposing that the energy function can be expressed as a sum $U(x) = \sum_{i=1}^n U_i(x)$, then we have

$$\lambda(t) = \langle v(t), U(x(t))\rangle^+ \leq \sum_{i=1}^n \langle v(t), U_i(x(t))\rangle^+ = \sum_{i=1}^n m_i(t) = m(t). \tag{10}$$

If we can simulate $\tau_1, \tau_2, \ldots, \tau_n$ with intensities $m_i(t)$, then we simulate τ with intensity $m(t)$ by letting $\tau = \min_i \tau_i$, and then we using thinning to generate the first arrival time from the process with intensity $\lambda(t)$. This will be useful, for example, for Bayesian applications in which the energy is the sum of the likelihood and a prior which can be handled analytically, e.g. a multivariate Gaussian (see below).

Additionally, if the distribution of interest is from an exponential family, then (8) may typically be solved analytically; see Bouchard-Côté et al. [8] for details.

3.1 Example: Gaussian Distributions

As we will frequently make use of the ECMC algorithms to sample from Gaussian distributions in our experiments, we demonstrate here how the arrival times (8) may be computed in this setting; we will only illustrate the case of the arrival times in the reflection algorithm, as those from the flip algorithm may be computed in the same way.

Suppose our target distribution is a d-dimensional multivariate Gaussian with variance-covariance matrix Σ. For simplicity - and without loss of generality - we let the mean be equal to zero. The density function is thus:

$$\pi(x) = (2\pi)^{-d/2}\,|\Sigma|^{-1/2} \exp\left(-\frac{1}{2}x^T\Sigma^{-1}x\right)$$

and so $U(x) = -\log\pi(x) = \frac{1}{2}\log(2\pi)^d|\Sigma| + \frac{1}{2}x^T\Sigma^{-1}x$. We find that $\nabla U(x) = \Sigma^{-1}x$, and so

$$\lambda(x,v) = \langle v, \nabla U(x)\rangle^+ = \left(v^T\Sigma^{-1}x\right)^+.$$

We now look to solve for τ^* such that $\tau^* = \arg\min_{t\geq 0} U(x+tv)$. We have

$$\begin{aligned}\tau^* &= \arg\min_{t\geq 0}\ U(x+tv)\\ &= \arg\min_{t\geq 0}\ \frac{1}{2}(x+tv)^T\Sigma^{-1}(x+tv)\end{aligned}$$

$$= \arg\min_{t\geq 0} x^T\Sigma^{-1}x + 2t\, x^T\Sigma^{-1}v + t^2\, v^T\Sigma^{-1}v.$$

The third term in the final line is increasing in t by the positive-definiteness of Σ^{-1}, and so we see that if $x^T\Sigma^{-1}v \geq 0$ then $\tau^* = 0$, otherwise, one easily finds that $\tau^* = -x^T\Sigma^{-1}v/v^T\Sigma^{-1}v$ so that finally

$$\tau^* = \left(-\frac{x^T\Sigma^{-1}v}{v^T\Sigma^{-1}v}\right)^+.$$

We may now solve for τ using (9). Suppose first that $\tau^* = 0$. Then the expression $U(x+\tau v) - U(x) = -\log U$ is a quadratic in τ, taking the positive root yields

$$\tau = \left(v^T\Sigma^{-1}v\right)^{-1}\left(-x^T\Sigma^{-1}v + \sqrt{(x^T\Sigma^{-1}v)^2 - 2\, v^T\Sigma^{-1}v\log U}\right). \tag{11}$$

Suppose now that $\tau^* = -x^T\Sigma^{-1}v/v^T\Sigma^{-1}v$. Once again, $U(x+\tau v) - U(x+\tau^* v) = -\log U$ is a quadratic in τ, and after some convenient cancellations of terms one finds the positive root

$$\tau = \left(v^T\Sigma^{-1}v\right)^{-1}\left(-x^T\Sigma^{-1}v + \sqrt{-2\, v^T\Sigma^{-1}v\log U}\right). \tag{12}$$

Equations (11) and (12) may be compactly expressed as

$$\tau = \left(v^T\Sigma^{-1}v\right)^{-1}\left(-x^T\Sigma^{-1}v + \sqrt{((x^T\Sigma^{-1}v)^+)^2 - 2\, v^T\Sigma^{-1}v\log U}\right). \tag{13}$$

This expression allows us to simulate exactly the event times for the reflection algorithm - analogous expressions exist for the flip algorithm, which we omit.

"I think that it is a relatively good approximation to truth — which is much too complicated to allow anything but approximations — that mathematical ideas originate in empirics."

- John von Neumann

4 Numerical Comparisons for Gaussian Targets.

In this section, we will compare the performance of the two basic ECMC methods from Section 2 in a handful of simple settings. In these numerical experiments, we will restrict ourselves to sampling from multivariate Gaussian distributions. Although these will of course be simpler than the distributions one would usually wish to sample from in practice, there are a number of advantages that make them appealing to use as toy problems. Firstly, we will be able to avail ourselves of the results from the previous section to simulate the trajectories cheaply and exactly. Although results like these will not typically be available in practice, it is nonetheless useful to see how the algorithms fare in these 'best-case' settings. Secondly, they offer a straightforward way of ascertaining how the performance is linked to the covariance structure of the target distribution. Thirdly, as it is common for Gaussians to be used to demonstrate MCMC samplers, one may compare results with the performance of a wide variety of methods. In the following three subsections, we consider sampling from, respectively, a two-dimensional Gaussian with variable correlation, a 100-dimensional Gaussian with a diagonal variance-covariance matrix with different marginal variances, and a 100-dimensional Gaussian with a randomly generated variance-covariance matrix.

4.1 A Two-Dimensional Example.

We first consider a two dimensional Gaussian distribution with mean zero and both marginal variances set to one. We expect that, as is usually the case for MCMC algorithms, the performance of both the flip method and the reflection method will suffer in the presence of strong correlation in the target distribution; however, it is not a priori clear to what extent the performances will deteriorate as correlation increases, nor which method will suffer the most heavily.

Consider Figure 3 below, which shows the estimated autocorrelation functions for both flip-ECMC and reflect-ECMC across four different correlation settings - $\rho = 0$, 0.75, 0.9 and 0.99. The target distribution is in this case symmetric in its components, and so we restrict our attention to the first. As expected, we see from Figure 3 that each algorithm suffers from increased correlation - the ACFs take longer to reach zero, and the integrated autocorrelation increases commensurately (not shown). The Figure also indicates that (note that the dark shades are from the reflection algorithm) the flip algorithm suffers more heavily from increased correlation - indeed, the ratio of the integrated autocorrelation time (IACT) between the flip and reflect methods grows as correlation increases (or, put another way, the ratio of effective sample sizes (ESS) decreases), and performs

considerably worse overall in this setting that the reflection algorithm (and drastically worse when correlation is very high). A possible exception occurs when the components are independent (red/salmon ACFs); in this case, we see that the autocorrelation for flip-ECMC is negative for a few lags, which will lower the IACT.

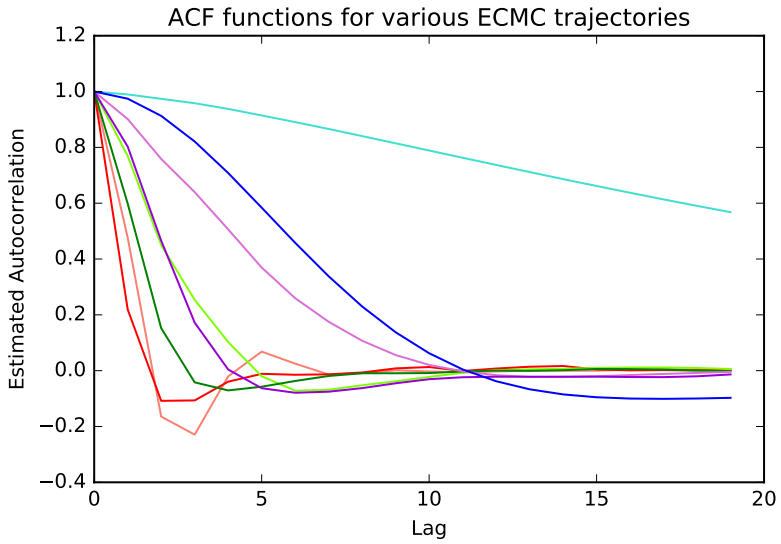

Figure 3: Estimated autocorrelation functions for both flip-ECMC and reflect-ECMC at four different correlation levels: 0 - red / salmon, 0.75 - green / light green, 0.9 - violet / light violet and 0.99 - blue / turquoise. The darker shades are from the reflection method and the lighter shades from the flip method. In each case, the ECMC samplers were run for 100000 events. Refreshment rates for the reflection method were set at $0.02, 0.04, 0.06$ and 0.1 respectively.

4.2 A first 100-Dimensional Example.

We now consider a one-hundred dimensional Gaussian distribution with mean zero and a diagonal covariance matrix, the marginal standard deviations being given by $0.01, 0.02, \ldots, 1$. This example was used in Neal [31] to compare the performance of the Hamiltonian Monte Carlo to that of classical random-walk Metropolis algorithms, and again in Bouchard-Côté et al. [8] to compare the performance of the reflection-ECMC algorithm to HMC.

Figure 4 shows the results of applying the flip and reflection algorithm to this distribution for 50000 events each. From the two left-hand panels we see that the reflection algorithm has estimated the means much more accurately than the flip algorithm; the right-hand panels suggest that neither method particularly outshines the other at estimating the marginal variances - the reflect method does a bit better, but it is barely perceptible. However, this does not quite accurately reflect the practical potential of these algorithms, as the running time was significantly longer for flip-ECMC (specifically, 70.63 seconds to 4.72 seconds for reflect-ECMC). To account for this, below in Figure 5 we display the same plots from trajectories of 10000 flipping events and 200000 reflection events, which took 23.80 and 18.58 seconds respectively.

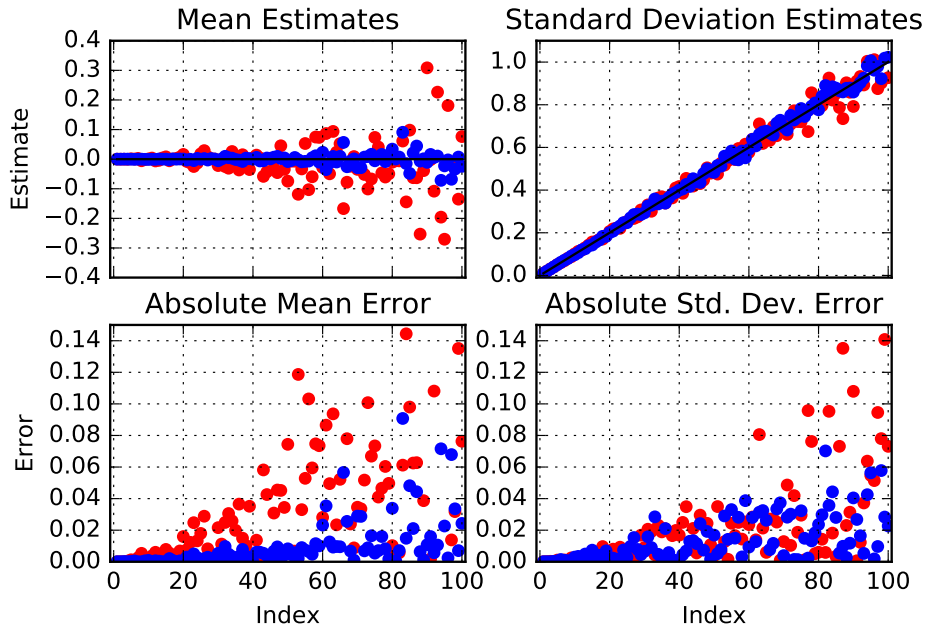

Figure 4: Clockwise from top left: estimates of the mean, estimates of the standard deviation, absolute error of standard deviation estimates, and absolute error of mean estimates for each component of a one-hundred dimensional Gaussian target distribution from trajectories of 50000 events for the flip method (red dots) and the reflect method (blue dots). The black lines in the top figures show the true means/standard deviations respectively. The running times were 73.80 seconds for the flip method and 4.23 seconds for the reflection method. The refreshment rate was $\lambda_0 = 0.65$.

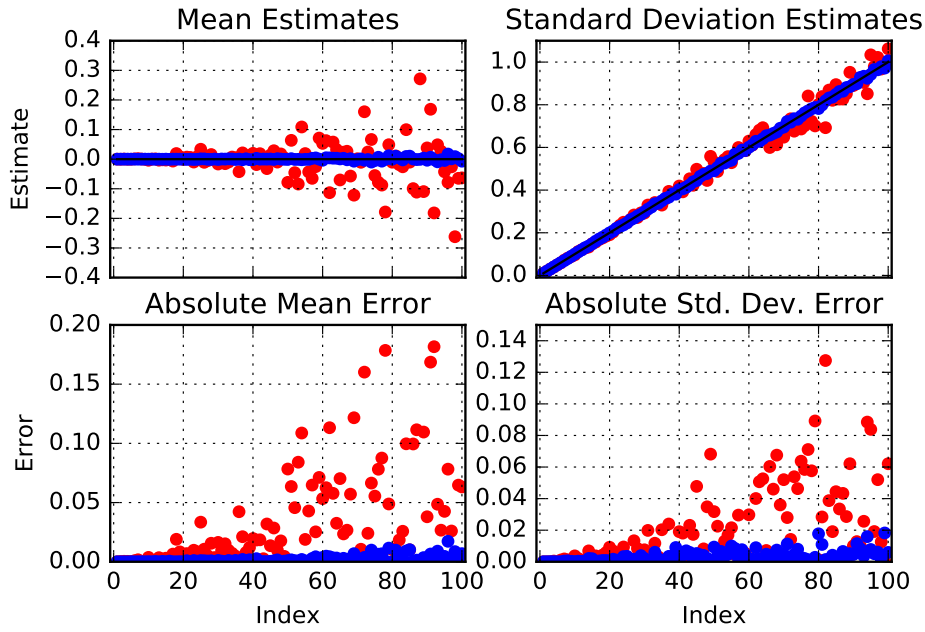

Figure 5: Clockwise from top left: estimates of the mean, estimates of the standard deviation, absolute error of standard deviation estimates, and absolute error of mean estimates for each component of a one-hundred dimensional Gaussian target distribution from trajectories of 50000 events for the flip algorithm (red) and 750000 events for the reflection algorithm (blue). The former ran for 72.99 seconds and the latter for 67.11 seconds. Refreshment rate was $\lambda_0 = .65$.

As one might have expected, Figure 5 shows that when the two algorithms are allowed to run for similar lengths of time, the performance is no longer comparable - the flip algorithm simply takes too much time sweeping through the d dimensions generating a candidate flip time for each component in turn. Below, we compare the performance of the reflection algorithm with HMC, using the implementation described in Neal [31].

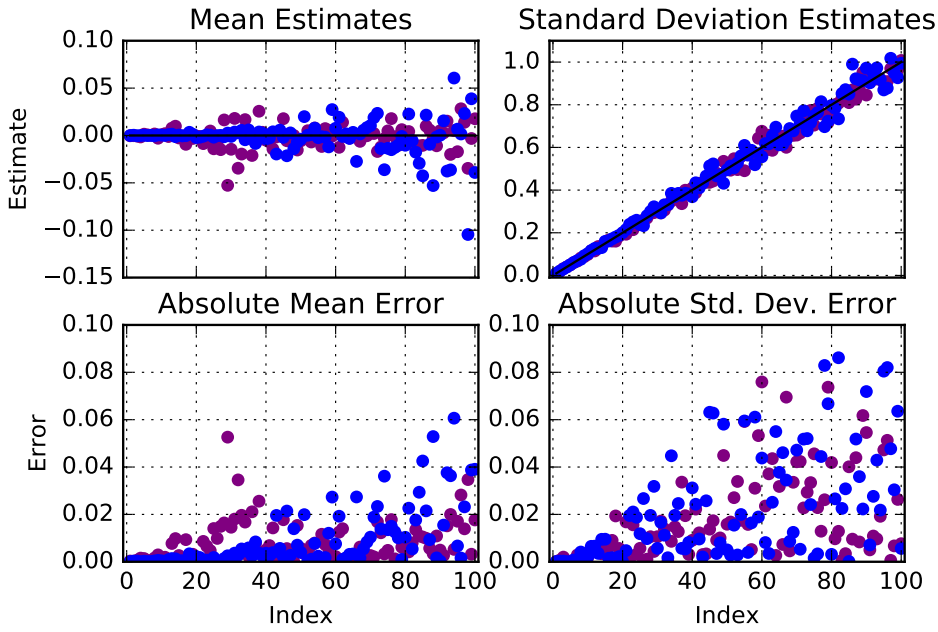

Figure 6: Clockwise from top left: estimates of the mean, estimates of the standard deviation, absolute error of standard deviation estimates, and absolute error of mean estimates for each component of a one-hundred dimensional Gaussian target distribution from trajectories of 30000 events for the reflection algorithm (blue) and 1500 iterations of HMC (purple). The former took 2.63 seconds and the latter 2.56s. Black lines indicate the true means and standard deviations. Refreshment rate for the reflection algorithm was $\lambda_0 = .65$, while HMC used $L = 150$ steps per iteration with stepsizes ϵ chosen uniformly on $(0.0104, 0.0156)$.

As we see from Figure 6 above, the performance of the reflection algorithm compares reasonably well with that of HMC for this problem when both methods are allowed to run for comparable amounts of time. The mean estimates are worse for dimensions 50-100, although it can be seen that HMC suffers at components around index 30; this is due to an issue with periodicity in the Hamiltonian trajectories [31], and would be significantly worse were the stepsize not randomly selected at each iteration - naturally this is not a problem from which the ECMC algorithms suffer as, unlike HMC, the entire trajectory may be used to compute Monte Carlo averages, rather than simply the points at which events occur [8]. Neither method dominates when it comes to estimating the standard deviations, though HMC is perhaps marginally more effective. We emphasize however that in this example, *HMC is run with near optimal settings* for the tuning parameters, which are in many cases extremely difficult to find. On the other hand, the λ_0 parameter was chosen based on a cursory examination of a few preliminary runs and is therefore almost certainly not optimal - and therefore the relative performance to HMC seen above could almost certainly be improved. The extreme sensitivity to the tuning parameters is one of the primary impediments to the widespread use of HMC in practice [31, 40] - see Hoffman and Gelman [19], Girolami and Calderhead [17] and Wang et al. [40] for some useful strategies developed to facilitate this task. For further details concerning

the tuning of the ECMC algorithms, see Section 6. Furthermore, the energy function in this (and any Gaussian) example is particularly simple: $U(x) = x^T\Sigma^{-1}x/2$ up to a constant, and may be computed very quickly, while in other settings, such as sampling from a posterior distribution over a large number of datapoints, HMC will suffer from the large amount of computation required to calculate the MH acceptance probability, whereas the exact sub-sampling methods for ECMC (see Section 5) will not, and will therefore be likely to iterate much more quickly than HMC. In Bouchard-Côté et al. [8], the authors exhibit a number of scenarios in which the reflection algorithm outperforms even state-of-the-art HMC methods. Finally, we note that for this simple example, the trajectories of the Hamiltonian flow could be computed exactly, precluding the need for a Metropolis-Hastings correction, however we have used the Stormer-Verlet (leapfrog) integrator (see [31, 17]) to ensure a fair comparison indicative of the relative performance of the methods in other settings.

In passing, we observe that - like HMC ([31]) - the reflection algorithm is invariant to rotation, which means that the above example can be seen as a demonstration of how it would perform on any Gaussian distribution in which the square roots of the eigenvalues of the covariance matrix were equal to $0.01, 0.02, \ldots, 1$; on the other hand, the flipping method is not, and so its performance will vary under different rotation of the variables. To see this, suppose that Q is a rotation matrix, and consider a rotation $x' = Qx$ of the original variables x. Then $\pi'(x') = \pi(Q^{-1}x)/|\det Q| = \pi(Q^{-1}x)$ and so $\nabla U'(x') = Q^{-1}\nabla U(Q^{-1}x)$. The dynamics of the original process at (x, v) will be governed by $\langle v, \nabla U(x)\rangle = v^T\nabla U(x)$; these will be identical to the dynamics of the rotated variables starting with initial velocity $w = Qv$, because $w^T\nabla U'(x') = v^TQ^tQ^{-1}\nabla U(Q^{-1}x') = v^T\nabla U(x)$, and so the invariance follows because $\psi(v) = \psi(Qv)$, i.e. because ψ is itself rotationally invariant. The flip algorithm will only be invariant under rotations Q which, for all $v \in E_F = \{-1, 1\}^d$, satisfy $Qv \in E_F$ (for example, if $d = 2$, the only non-trivial rotations under which the process remains invariant are those of $\pi/2$, π, and $3\pi/2$ about the origin). Of course, both methods are invariant under translations of the x variables.

4.3 A second 100-Dimensional Example.

We now consider another one-hundred dimensional Gaussian target distribution - with mean zero and using the covariance matrix Σ used in Roberts and Rosenthal [35] to assess the performance of an adaptive Metropolis-Hastings algorithm; we simulate such a matrix by letting M be such that for each $i, j = 1, \ldots 100$ we have $M_{ij} \sim$ i.i.d. $\mathrm{N}(0, 1)$, and taking $\Sigma = MM^T$, the idea being to generate a covariance matrix sufficiently "erratic, so that sampling from $\pi(\cdot)$ represents a significant challenge if the dimension is at all high" [35].

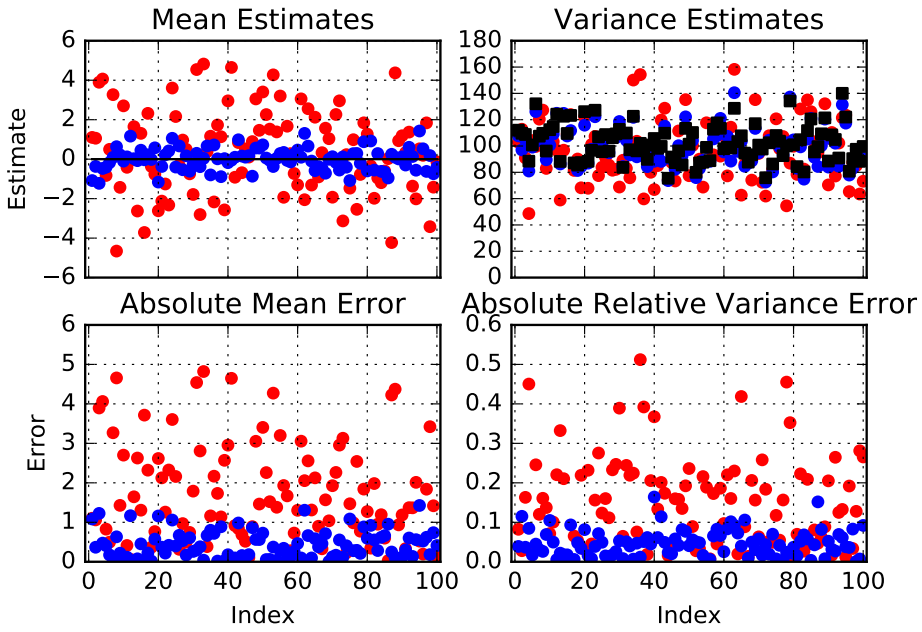

Figure 7: Clockwise from top left: estimates of the mean, estimates of the variance, absolute relative error of the variance estimates, and absolute error of mean estimates for each component of a one-hundred dimensional Gaussian target distribution with covariance matrix Σ from trajectories of 500000 events for the flip method (red dots) and the reflect method (blue dots). The black lines/dots in the top figures show the true means/-variances respectively. The running times were 1061.71 seconds for the flip method and 46.03 seconds for the reflection method. The refreshment rate was $\lambda_0 = 0.65$.

As we see above in Figure 7, even after 500000 events, the estimates from the flip algorithm are still well off target; those from the reflection algorithm are considerably better. Once again, we draw attention to the running times - again, of course, had the reflection algorithm been allowed to run as long as the flip-ECMC method, then the gulf in performance would be immense. We do not labour this point however, as in other scenarios there will usually be a need to use methods - e.g. superposition - that will reduce the discrepancy in computation time.

"It is quite a three-pipe problem. Pray do not speak to me for fifty minutes."

- Sherlock Holmes, *The Adventure of the Red-Headed League*

5 Improvements for Handling Large-Scale Inference.

Owing to the high demands imposed upon statistical methodology by ever increasing volumes of available data, it has in recent times become imperative that improvements be made so as to increase the computational efficiency of algorithms used for statistical inference. The computations required in Bayesian statistics are especially intense, and Markov chain Monte Carlo methods - the most commonly used tools to perform them - are known to suffer immensely as the dimension and number of observations in datasets increase. Accordingly, there has been a considerable amount of work done to address this, so that Bayesian methods depending on MCMC will be able to keep pace with the ever expanding frontiers of data science.

While, naturally, the performance of traditional MCMC methods degrade as the dimension of the target distribution d increases, they are for practical purposes almost unusable in situations where the number of observations in a dataset n is large, due to the need to compute at each iteration of the chain an acceptance probability which depends on a likelihood ratio involving each of the individual observations. A large proportion of the developments in scalable MCMC algorithms have directly addressed this - see Bardenet et al. [4] for a review of some of the methods that have been proposed. As these authors indicate, these can be broadly categorized as "divide and conquer" and "sub-sampling" methods - in the former, the dataset is divided in to batches and then MCMC is run on each batch in turn and then the results are combined to get an approximation of the posterior distribution, while in the latter the emphasis is on methods which reduce the number of data points required in likelihood calculations at each iteration. Unfortunately, divide and conquer approaches rely on inchoate methods for combining posterior approximations lacking firm theoretical justification, scale poorly with the number of batches, and often rely on results which are asymptotic in batch size [4]. Meanwhile, with a few notable exceptions (e.g. 'Firefly Monte Carlo' - see MacLaurin and Adams [25] and 'pseudo-marginal MCMC' - see Andrieu and Roberts [1]) such methods are inexact, that is to say that even in the limit as $n \to \infty$ they sample from an approximation to the posterior distribution. In what follows, we present an 'exact approximate scheme' which was employed in Bouchard-Côté et al. [8] and then again in Bierkens et al. [7] which functions by replacing a full evaluation of the gradient of the log-likelihood of all n observations with an unbiased estimator while nonetheless sampling from the exact posterior distribution. Furthermore, we will discuss two powerful ways in which the efficiency of this exact sub-sampling method can be improved, namely: a technique invoking an alias sampling idea (see Devroye [12]) which was used in this context independently by Bouchard-Côté et al. [8] and Kapfer and Krauth [22], and the use of control variates to reduce the variance of the unbiased estimator of the gradient of the

log-likelihood, which was used to great effect in Bierkens et al. [7] and has appeared in similar contexts as well, see e.g. Bardenet et al. [4]. Using Bayesian logistic regression as a running example, we present a number of numerical comparisons between the various methods.

5.1 Sub-Sampling and the Alias Method.

It is often the case that MCMC methods can be modified or extended so as to capitalize on certain structural properties of the target distribution of interest. The most well known example is perhaps the Gibbs sampler which exploits conditional independences between variables, although there are many other instances of structural exploitation in MCMC - see for example Shariff et al. [37] where symmetries in the target are used to design efficient MCMC proposals. Bouchard-Côté et al. [8] propose a 'local' extension of the reflection algorithm which requires the target density to admit a representation of the form

$$\pi(x) = \prod_{f \in F} \pi_f(x_f) \tag{14}$$

where x_f is the subset of the variables x given by $N_f \subset \{1, 2, \ldots, d\}$ and F is an index set called the set of factors. In this setting, the energy associated to the density π can be expressed as

$$U(x) = \sum_{f \in F} U_f(x), \tag{15}$$

and we have that $\partial U_f(x)/\partial x_k = 0$ for $k \in \{1, 2, \ldots, d\} \backslash N_f$. This framework "can be formalized using factor graphs, ...and generalizes undirected graphical models" [8]. Observe that in the setting in which the target is a Bayesian posterior distribution arising from a prior and the likelihood of R data points which are conditionally independent given variables x, the energy can be written as

$$U(x) = U_0(x) + \sum_{r=1}^{R} U_r(x), \tag{16}$$

and thus is incorporated into the framework given by (15) with one factor being the prior likelihood and R subsequent factors which are the individual likelihoods of the data points, and $N_f = \{1, 2, \ldots, d\}$ for all $f \in F$. In this setting, the algorithm reduces to the sub-sampling approach outlined in Bierkens et al. [7]; we refer the reader to [8] for details of the local algorithm in full generality, and in what follows we present only the special case which corresponds to what is found in [7]; furthermore, we present details only for the reflection algorithm as the details for the flipping algorithm are entirely analogous.

The method proceeds by defining, for each of R factors, a reflection operator akin to (2) and an intensity akin to (3); that is, for $j = 1, 2, \ldots R$, let

$$\lambda_j(x, v) = \langle v, \nabla U_j(x) \rangle^+ \tag{17}$$

and let

$$R_j[x]v = \left(I_d - 2\frac{\nabla U_j(x)\nabla U_j(x)^t}{\langle \nabla U_j(x), \nabla U_j(x)\rangle}\right) v = v - 2\frac{\langle \nabla U_j(x), v\rangle}{||U_j(x)||^2}\nabla U_j(x). \tag{18}$$

Supposing then that we have access to bounds M_j for the intensities, i.e. $\lambda_j(x(t), v(t)) = \lambda_j(t) \leq M_j(t)$ for all $j = 1, 2, \ldots, R$, we let τ be the first arrival time of a nonhomogeneous Poisson process of intensity $M(t) = \sum_{j=1}^{R} M_j(t)$, and then rather than using the full energy to determine whether to reflect at time τ, instead we choose factor r by letting

$$\mathbb{P}(r = s) = \frac{M_s(\tau)}{M(\tau)}, \tag{19}$$

and then a reflection occurs if

$$u < \frac{\lambda_r(\tau)}{M_r(\tau)}, \tag{20}$$

where $u \sim \mathrm{U}(0, 1)$, in which case we set $v' = R_j[x(\tau)]v$. Pseudocode for the sub-sampling reflection algorithm is given in Algorithm (3) below.

Algorithm 3 Reflection ECMC with sub-sampling.

Arbitrarily initialize $(x^{(0)}, v^{(0)}) \in \mathbb{R}^d \times \mathbb{R}^d$.
Let $T = 0$.
for $i = 1, 2 \ldots$ **do**
    Simulate $\tau_{reflect}$ as the first arrival time of a Poisson process of rate $M(t) = \sum_j M_j(t)$, where $M_j(t) \geq \lambda_j(x(t), v(t))$ for each j.
    Simulate $\tau_{refresh} \sim \mathrm{Exp}(\lambda_0)$.
    Set $\tau^{(i)} \leftarrow \min(\tau_{refresh}, \tau_{reflect})$.
    Set $x^{(i)} \leftarrow x^{(i-1)} + \tau^{(i)} v^{(i-1)}$.
    if $\tau^{(i)} = \tau_{refresh}$ **then**
        Set $v^{(i)} \sim \psi$.
    end if
    if $\tau^{(i)} = \tau_{reflect}$ **then**
        Choose factor r with probability $M_j(\tau^{(i)})/M(\tau^{(i)})$.
        if $u < \lambda_j(\tau^{(i)})/M_j(\tau^{(i)})$ where $u \sim \mathrm{U}(0, 1)$, **then**
            Set $v^{(i)} \leftarrow R[x^{(i)}]v^{(i-1)}$.
        else
            Set $v^{(i)} \leftarrow v^{(i-1)}$.
        end if
    end if
    Set $T \leftarrow T + \tau^{(i)}$.
    Return $(x^{(i)}, v^{(i)}, T)$.
end for

Proofs of correctness (i.e. correct invariant distribution and ergodicity of resulting Markov chain) for the sub-sampling algorithm for flip-ECMC and reflect-ECMC are given as Theorem 4.1 in Bierkens et al. [7], and as an extension to Proposition 1 in Appendix 3 of Bouchard-Côté et al. [8]. When $M_j(t) = \overline{M}(t)$ for all $j = 1, 2, \ldots, R$ so that (19) reduces to sampling uniformly from $\{1, 2, \ldots, R\}$; we shall refer to this procedure as *naive* sub-sampling.

In general, the need to evaluate only one of the intensities (17) at each iteration, coupled with the fact that the sum $M(t) = \sum_j M_j(t) = \sum_j \overline{M}(t) = R\overline{M}(t)$ can be computed in $O(1)$ time will mean that the algorithmic complexity of an iteration will be reduced by a factor of $O(n)$ [7]; however, the requirement of using the 'worst case' bound $\overline{M}$ means that the efficiency of the naive algorithm may be dramatically reduced, as the ratio in (20) will be typically be extremely small, and so most iterations will fail to produce a reflection. However, in scenarios in which - usually by recourse to pre-computed data structures - one can loop over the factors implicitly to compute the sum $\sum_j M_j(t)$ and perform the sampling step (19) in constant time, then it will be possible to enjoy the computational parsimony of the naive method without suffering from the loss of efficiency due to the loose bounds. This will be made possible by the alias sampling method, given as Theorem 4.1 in Chapter 3 of Devroye [12]:

Proposition 5.1 *Every probability vector $p_1, p_2, \ldots, p_k$ (i.e. $p_i \geq 0$ and $\sum_i p_i = 1$) can be expressed as an equiprobable mixture of k two-point distributions.*

Proposition 5.1 will make it possible to compute the sampling in (19) in constant time by first sampling uniformly from $\{1, 2, \ldots, R\}$ and then sampling from the corresponding two-point distribution; note that the alias method requires a set-up which can be performed in $O(k)$.

At this point, it is convenient to introduce an example that will be used to illustrate the sub-sampling method and the alias method.

5.1.1 Example: Bayesian Logistic Regression.

Consider a dataset consisting of binary outcomes $y^r \in \{0, 1\}$ associated to d-dimensional covariates $\xi^r \in \mathbb{R}^d$ and parameter $x \in \mathbb{R}^d$, where the outcomes are assumed to been generated from the logistic regression model

$$\mathbb{P}(y = 1|\xi, x) = \frac{1}{1 + \exp(-\sum_{i=1}^d x_i \xi_i)}. \tag{21}$$

With a flat prior for x, which we assume for simplicity, the likelihood function is given by

$$\pi(x) = \prod_{r=1}^R \frac{\exp(y^r \sum_{i=1}^d x_i \xi_i)}{1 + \exp(\sum_{i=1}^d x_i \xi_i)} \tag{22}$$

and so the energy function (plus a constant) is

$$U(x) = \sum_{r=1}^R \left\{ \log\left(1 + \exp\left(\sum_{i=1}^d x_i \xi_i^r\right)\right) - y^j \sum_{i=1}^d x_i \xi_i^r \right\}, \tag{23}$$

and so the i-th component of the gradient is easily seen to be

$$\partial_i U(x) = \sum_{r=1}^R \left(\frac{\xi_i^r \exp\left(\sum_{j=1}^d x_j \xi_j^r\right)}{1 + \exp\left(\sum_{j=1}^d x_j \xi_j^r\right)} - y^r \xi_i^r \right). \tag{24}$$

Now we seek to bound the intensities (17) uniformly in r. We have

$$\begin{aligned}
\lambda_r(x(t), v(t)) &= \langle v, \nabla U_r(x + tv)\rangle^+ \\
&= \left(\sum_{i=1}^d v_i \left(\frac{\xi_i^r \exp\left(\sum_{j=1}^d (x_j + tv_j)\xi_j^r\right)}{1 + \exp\left(\sum_{j=1}^d (x_j + tv_j)\xi_j^r\right)} - y^r \xi_i^r \right) \right)^+ \\
&\leq \sum_{i=1}^d \left(v_i \left(\frac{\xi_i^r \exp\left(\sum_{j=1}^d (x_j + tv_j)\xi_j^r\right)}{1 + \exp\left(\sum_{j=1}^d (x_j + tv_j)\xi_j^r\right)} - y^r \xi_i^r \right) \right)^+ \\
&\leq \sum_{i=1}^d \left\| v_i \left(\frac{\xi_i^r \exp\left(\sum_{j=1}^d (x_j + tv_j)\xi_j^r\right)}{1 + \exp\left(\sum_{j=1}^d (x_j + tv_j)\xi_j^r\right)} - y^r \xi_i^r \right) \right\| \\
&\leq \sum_{i=1}^d |v_i||\xi_i^r| \\
&\leq \sum_{i=1}^d |v_i| \max_r |\xi_i^r|,
\end{aligned}$$

where the third inequality follows from $0 < \exp(a)/(1 + \exp(a) < 1$ for $a \in \mathbb{R}$. Hence, if $\overline{\xi}_i = \max_r |\xi_i^r|$, we may implement the naive sub-sampling method with $\overline{M}(t) = \sum_i |v_i||\overline{\xi}_i$.

Now we revisit the above calculations in order to construct bounds in such a way that the sampling in (19) is amenable to the alias method. We follow calculations from Section 4.6 and Appendix B of Bouchard-Côté et al. [8], and extend their presentation to allow for the possibility of negative covariates. This will require that various computations be performed before the sampling can begin. First, for each $i = 1, 2, \ldots, d$ we let

$$(\xi_i)^{+,1} + (\xi_i)^{-,0} = \sum_{r=1}^r \left\{ (\xi_i^r)^+ [y^r = 1] + (\xi_i^r)^- [y^r = 0] \right\} \tag{25}$$

and

$$(\xi_i)^{+,0} + (\xi_i)^{-,1} = \sum_{r=1}^r \left\{ (\xi_i^r)^+ [y^r = 0] + (\xi_i^r)^- [y^r = 1] \right\}. \tag{26}$$

Then we create, for each $i = 1, 2, \ldots, d$ the following two probability vectors of length R, with r-th entries given by

$$\frac{(\xi_i^r)^{+,1} + (\xi_i^r)^{-,0}}{(\xi_i)^{+,1} + (\xi_i)^{-,0}} \tag{27}$$

and

$$\frac{(\xi_i^r)^{+,0} + (\xi_i^r)^{-,1}}{(\xi_i)^{+,0} + (\xi_i)^{-,1}}, \tag{28}$$

where the denominators above are given by (25) and (26) respectively, and then conclude the pre-computation by constructing alias sampling tables according to the scheme outlined in Section 3.4 of Devroye [12]. Now, recall that for $a \in \mathbb{R}$, $(a)^+ = \max(0, a)$ denotes the positive part of a, and let $(a)^- = -\min(0, a)$ denote the negative part of a so that $a = (a)^+ - (a)^-$. Let $s : \mathbb{R} \to \mathbb{R}^+$ denote the logistic function $a \mapsto \exp(a)/(1 + \exp(a)$,

with $0 < s(a) < 1$ for all $a \in \mathbb{R}$. Let $[\cdot]$ be a shorthand for the indicator function, i.e. $[A](x) = 1$ if $x \in A$ and $[A](x) = 0$ if $x \notin A$; we will abuse the notation and write $[A]$ for $[A](x)$ when the context is clear. By (24), for $r \in \{1, 2, \ldots, R\}$ with $y^r = 0$ we have

$$\begin{aligned}
\lambda_r(t) &= \left(\sum_{i=1}^{d} v_i \xi_i^r \, s\left(\langle \xi^r, x(t) \rangle \right) \right)^+ \\
&\leq \left(\sum_{i=1}^{d} v_i \xi_i^r \right)^+ \\
&\leq \sum_{i=1}^{d} (v_i \xi_i^r)^+ \\
&= \sum_{i=1}^{d} |v_i| \left([v_i \geq 0] (\xi_i^r)^+ + [v_i < 0] (\xi_i^r)^- \right).
\end{aligned}$$

Likewise, for $r \in \{1, 2, \ldots, R\}$ with $y^r = 1$ we have

$$\begin{aligned}
\lambda_r(t) &= \left(\sum_{i=1}^{d} v_i \xi_i^r \, \left(s\left(\langle \xi^r, x(t) \rangle \right) - 1 \right) \right)^+ \\
&= \left(\sum_{i=1}^{d} -v_i \xi_i^r \, \left(1 - s\left(\langle \xi^r, x(t) \rangle \right) \right) \right)^+ \\
&\leq \left(\sum_{i=1}^{d} -v_i \xi_i^r \right)^+ \\
&\leq \sum_{i=1}^{d} (-v_i \xi_i^r)^+ \\
&= \sum_{i=1}^{d} |v_i| \left([v_i < 0] (\xi_i^r)^+ + [v_i \geq 0] (\xi_i^r)^- \right).
\end{aligned}$$

Combining these expressions yields

$$\lambda_r(t) \leq \sum_{i=1}^{d} |v_i| \left(\left[v_i (-1)^{y^r} \geq 0 \right] (\xi_i^r)^+ + \left[v_i (-1)^{y^r} < 0 \right] (\xi_i^r)^- \right) = M_r(t) = M_r. \tag{29}$$

Summing over the data points gives

$$\begin{aligned}
M(t) = \sum_{r=1}^{R} M_r(t) &= \sum_{r=1}^{R} \sum_{i=1}^{d} |v_i| \left(\left[v_i (-1)^{y^r} \geq 0 \right] (\xi_i^r)^+ + \left[v_i (-1)^{y^r} < 0 \right] (\xi_i^r)^- \right) \\
&= \sum_{i=1}^{d} |v_i| \left\{ \sum_{r=1}^{R} \left(\left[v_i (-1)^{y^r} \geq 0 \right] (\xi_i^r)^+ + \left[v_i (-1)^{y^r} < 0 \right] (\xi_i^r)^- \right) \right\}
\end{aligned}$$

where, depending on the signs of the v_i's, the inner sums can be computed in constant time by recourse to either (25) or (26). Now, to implement the sampling in (19) efficiently, we consider (see [8]) a contrived distribution over the data points and dimension indices

with mass function given by

$$\mathbb{P}(r,i) = \frac{1}{M}\left\{|v_i|\left(\left[v_i(-1)^{y^r} \geq 0\right](\xi_i^r)^+ + \left[v_i(-1)^{y^r} < 0\right](\xi_i^r)^-\right)\right\} \tag{30}$$

where $M = \sum_r M_r$. The marginal distribution of i is given by

$$\mathbb{P}(i) = \frac{1}{M}\sum_{r=1}^{R}|v_i|\left(\left[v_i(-1)^{y^r} \geq 0\right](\xi_i^r)^+ + \left[v_i(-1)^{y^r} < 0\right](\xi_i^r)^-\right) \tag{31}$$

$$= \frac{1}{M}|v_i|\left((\xi_i)^{+,[v_i<0]} + (\xi_i)^{-,[v_i\geq 0]}\right), \tag{32}$$

where the term in brackets is either (25) or (26), again depending on the sign of v_i. By construction, the marginal distribution of r is given by (19), and so to sample from $\mathbb{P}(r)$ we may sample first from $\mathbb{P}(i)$ and then from the conditional $\mathbb{P}(r|i)$ which is given by

$$\mathbb{P}(r|i) = \frac{\mathbb{P}(r,i)}{\mathbb{P}(i)} = \frac{(\xi_i^r)^{+,[v_i<0]} + (\xi_i^r)^{-,[v_i\geq 0]}}{(\xi_i)^{+,[v_i<0]} + (\xi_i)^{-,[v_i\geq 0]}}; \tag{33}$$

this we achieve in $O(1)$ using the alias tables.

5.2 Control Variates.

The most promising improvement to the basic flipping algorithm suggested in Bierkens et al. [7] according to their simulations seems to be the method of control variates. After deriving the method for the flip algorithm, they demonstrate its effectiveness using Gaussians and posterior distributions arising from Bayesian logistic regression models. In what follows we show that the technique can be used for the reflection algorithm as well, and in the next section we undertake some numerical comparisons between the reflect algorithm with control variates and the flip method with control variates presented in [7]. We also compare its effectiveness against the alias method presented above.

The control variate method relies on the assumption that the components of the gradient of the energy function are globally and uniformly Lipschitz ([7]), that is, that there exist constants C_i for $i = 1, 2, \ldots, d$ such that for some $p \in [1, \infty]$, we have for all $x_1, x_2 \in \mathbb{R}^d$ and for each $i = 1, 2, \ldots, d$, $j = 1, 2, \ldots, R$ we have

$$|\partial_i U^j(x_1) - \partial_i U^j(x_2)| \leq C_i||x_1 - x_2||_p, \tag{34}$$

where $||\cdot||_p$ is the L^p norm. Proceeding under this assumption, we select a reference point $x^* \in \mathbb{R}^d$, and we observe that, when (16) holds, for all $x \in \mathbb{R}^d$ we have

$$\partial_i U(x) = \partial_i U(x^*) + \sum_{j=1}^{R}\left(\partial_i U^j(x) - \partial_i U^j(x^*)\right). \tag{35}$$

We define

$$\widetilde{U}^j(x) = \frac{1}{R}U(x^*) + U^j(x) - U^j(x^*), \tag{36}$$

and we consider a process identical to the subsampling method, except rather than using (17) and (18) for the intensity function and reflection operators respectively, we replace $U^j(x)$ with $\widetilde{U}^j(x)$ and instead use the intensity

$$\widetilde{\lambda}_j(x,v) = \langle v, \nabla \widetilde{U}^j(x)\rangle^+ \tag{37}$$

and the reflection operator

$$\widetilde{R}_j[x]v = \left(I_d - 2\frac{\nabla\widetilde{U}^j(x)\nabla\widetilde{U}^j(x)^t}{\langle\nabla\widetilde{U}^j(x),\nabla\widetilde{U}^j(x)\rangle}\right)v = v - 2\frac{\langle\nabla\widetilde{U}^j(x),v\rangle}{||\widetilde{U}^j(x)||^2}\nabla\widetilde{U}^j(x). \tag{38}$$

Now, for this intensity, we have

$$\begin{aligned}
\widetilde{\lambda}_j(x,v) &= \langle v, \nabla\widetilde{U}^j(x)\rangle^+ \\
&= \frac{1}{R}\langle v, \nabla U(x^*) + \big(U^j(x) - U^j(x^*)\big)\rangle^+ \\
&\leq \frac{1}{R}\langle v, \nabla U(x^*)\rangle^+ + \langle v, \big(U^j(x) - U^j(x^*)\big)\rangle^+ \\
&\leq \frac{1}{R}\langle v, \nabla U(x^*)\rangle^+ + \sum_{i=1}^d \big(v_i\partial_i U^j(x) - \partial_i U^j(x^*)\big)^+ \\
&\leq \frac{1}{R}\langle v, \nabla U(x^*)\rangle^+ + \sum_{i=1}^d |v_i||\partial_i U^j(x) - \partial_i U^j(x^*)|,
\end{aligned}$$

where we have used that for $a, b \in \mathbb{R}$, $(a+b)^+ \leq (a)^+ + (b)^+$. To bound the intensity as a function of t, we note that

$$\begin{aligned}
\widetilde{\lambda}_r(t) = \widetilde{\lambda}_j(x+tv,v) &\leq \frac{1}{R}\langle v, U(x^*)\rangle^+ + \sum_{i=1}^d |v_i||\partial_i U^j(x+tv) - \partial_i U^j(x^*)| \\
&= \frac{1}{R}\langle v, \nabla U(x^*)\rangle^+ + \sum_{i=1}^d |v_i||\partial_i U^j(x+tv) - \partial_i U^j(x) + \partial_i U^j(x) - \partial_i U^j(x^*)| \\
&\leq \frac{1}{R}\langle v, \nabla U(x^*)\rangle^+ + \sum_{i=1}^d |v_i|\left(|\partial_i U^j(x+tv) - \partial_i U^j(x)| + |\partial_i U^j(x) - \partial_i U^j(x^*)|\right) \\
&\leq \frac{1}{R}\langle v, \nabla U(x^*)\rangle^+ + \sum_{i=1}^d |v_i| C_i\left(t\,||v||_p + ||x - x^*||_p\right) \\
&= a + b\,t = M(t),
\end{aligned}$$

which follows from (34) and the triangle inequality; this is an affine bound, and thus the process with intensity $RM(t)$ may be simulated exactly. In order to guarantee that these bounds work well in practice, we will usually choose x^* to be a point around which much of the probability mass of the posterior is concentrated, such as the posterior mode or the maximum likelihood estimate. Finding such a reference point will require a computational overhead before the algorithm may begin, although the time spent on this phase will usually be negligible.

The validity of this method follows from a straightforward modification of the theorem in Appendix A.1 of Bouchard-Côté et al. [8].

5.2.1 Lipschitz Bounds for Logistic Regression.

In the logistic regression example from above, we have from (24) that the i-th component of the gradient for the r-th observation is given by

$$\partial_i U^r(x) = \frac{\xi_i^r \exp\left(\sum_{j=1}^d x_j \xi_j^r\right)}{1 + \exp\left(\sum_{j=1}^d x_j \xi_j^r\right)} - y^r \xi_i^r \tag{39}$$

and so for $k = 1, \ldots, d$ the k, i-th entry of the Hessian matrix is given by

$$\partial_k \partial_i U^r(x) = \frac{\xi_k^r \xi_i^r \exp\left(\sum_{j=1}^d x_j \xi_j^r\right)}{\left(1 + \exp\left(\sum_{j=1}^d x_j \xi_j^r\right)\right)^2}. \tag{40}$$

Using the bounds $0 < \exp(a)/(1+\exp(a)) < 1$ and $0 < \exp(a)/(1+\exp(a))^2 \leq 1/4$ yields

$$|\partial_i U^r(x)| \leq |\xi_i^r| \tag{41}$$

and

$$|\partial_k \partial_i U^r(x)| \leq \frac{1}{4} |\xi_k^r \xi_i^r|, \tag{42}$$

and thus we have that (34) holds for $p = 2$ with

$$C_i = \max_{r=1,\ldots,R} \frac{1}{4} |\xi_i^r| \, \|\xi^r\|_2 \,, \tag{43}$$

which follows from the mean value theorem along the line from x_1 and x_2 [7]. These expressions will be used to implement the control variate method for both the flip and the reflection algorithms.

5.3 Numerical Experiments.

In this section we perform a sequence of experiments comparing the performance of the two ECMC methods and their variants described above for Bayesian logistic regression; for simplicity we use flat priors for the parameters. Figures 8 and 9 below show, respectively, boxplots of the time-normalized effective sample sizes (ESS per second) and raw effective sample sizes (ESS) for 10 runs of the various methods repeated on each of four different types of datasets, one for each combination of low/high dimension ($d = 5$, $d = 20$), and small/large number of observations ($R = 500$, $R = 10000$). We remark that none of the ESS/s figures include the pre-computation times for the informed subsampling or control variate methods - in long runs such as these they are negligible. Note that as dimension and observation count increased, we ran the chains for larger number of iterations to ensure that the approximations involved in estimating the ESS remained

reasonable (see Appendix 8.1), and so raw ESS totals across the settings are inflated for the longer runs, while the values of ESS/s across settings must be interpreted with care. Thus when we discuss changes in performance across the four settings, we will largely be referring to performance *relative to the other methods* .

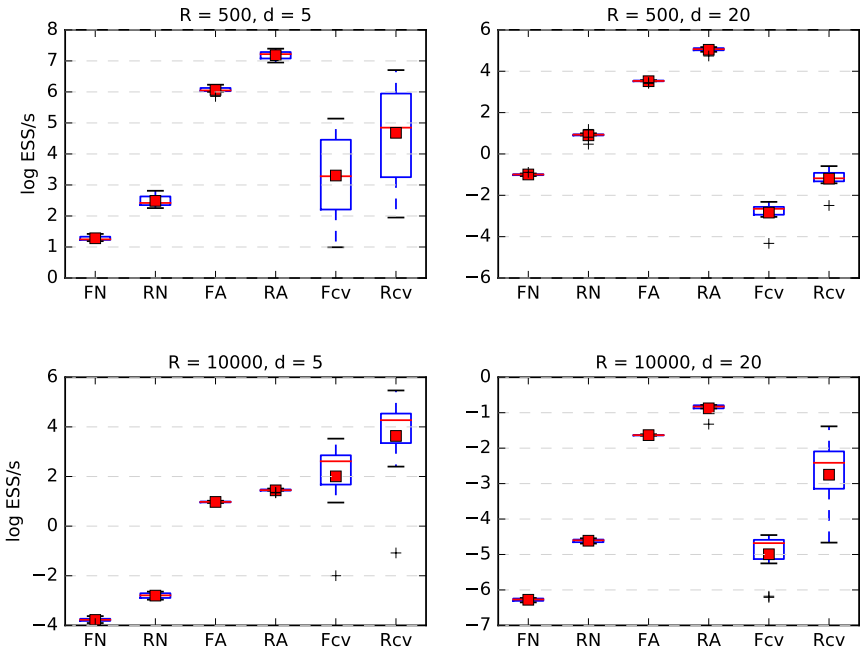

Figure 8: Boxplots showing the Effective Sample Size per CPU second for 10 experiments of flip-ECMC and reflect-ECMC in four different settings: $R = 500, d = 5$, $R = 500, d = 20$, $R = 10000, d = 5$ and $R = 10000, d = 20$. The red dashes indicate the median, and the red boxes show the mean. The horizontal axis indexes the method that was used: FN and RN for the naive subsampling variants of the flip/reflect algorithms respectively, FA/RA for the alias sampling variants, and Fcv/Rcv for the control variate variants. Each experiment consisted of, respectively, 10^6, 2×10^6, 3×10^6 and 10^7 events, and was carried out on a synthetic binary dataset in which the true parameters were randomly generated from a d-dimensional standard normal distribution, and covariates were randomly generated as the absolute values of d-dimensional standard normals for each observation. For each experiment, various methods were carried out on the same dataset. The refreshment parameter for the reflection algorithm was set (without any preliminary tuning) to $\lambda_0 = 1, 2, 3, 6$ for the four scenarios listed above, respectively. All experiments are initialized at the MLE, with randomly drawn velocities.

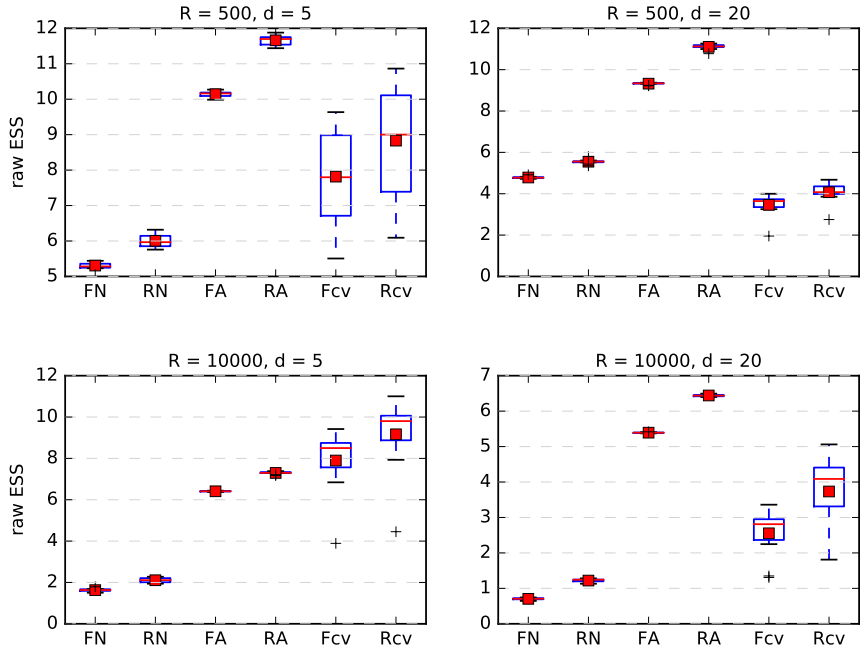

Figure 9: Boxplots showing the corresponding raw Effective Sample Sizes for the 10 experiments of flip-ECMC and reflect-ECMC shown above; that is, for four different settings: $R = 500, d = 5$, $R = 500, d = 20$, $R = 10000, d = 5$ and $R = 10000, d = 20$. Each experiment consisted of, respectively, 10^6, 2×10^6, 3×10^6 and 10^7 events, and was carried out on a synthetic binary dataset in which the true parameters were randomly generated from a d-dimensional standard normal distribution, and covariates were randomly generated as the absolute values of d-dimensional standard normals for each observation. For each experiment, various methods were carried out on the same dataset.

Several features revealed in Figures 8 and 9 are immediately striking. As expected, we note the poor performance of the naive sub-sampling methods relative to the alias sampling methods, which use the same technique to bound the intensity for a given observation but do not require a uniform bound for all $j \in \{1, \ldots, R\}$. In the low-dimension setting, the naive methods are the least effective, although at $d = 20$ they are seen to outperform their control variate counterparts (that is, Fn outperforms Fcv and Rn outperforms Rcv). This rectifies itself as the number of observations increases however, and at $d = 20$, $R = 10000$, we see that they are again inferior to the control variate methods. The sharp decline in the performance of the control variate methods as the dimension increases will be largely due to the presence in the Lipschitz bound (34) of the distance term $\|x - x^*\|_p$. This will cause the intensity to be quite high (and the bound quite loose) whenever the chain moves away from the reference point (which is usually a region of high probability) regardless of the direction of the velocity, and so large numbers of candidate event times will be drawn (and rejected) even if the chain is moving to regions of lower energy; it is likely that this behaviour is also partly responsible for

the much larger variability in the performance of the CV methods relative to the others. This problem may be alleviated somewhat if a Lipschitz bound may be found with a higher value of p, (as for $p, q \in [1, \infty]$ such that $p < q$, we have $\|\cdot\|_p \geq \|\cdot\|_q$) although there is a trade-off between p and the constants C_i in (34) that must be considered (see Bierkens et al. [7] for a brief discussion of the trade-off; in particular they find that $p = \infty$ is optimal when the target is Gaussian and recommend $p = 2$ as a sensible choice when no knowledge of the optimal value is available). Another possible solution would be to periodically reset the reference point x^* after an event to be the current position x of the chain, and to continually reset it after some fixed number of iterations, setting it back to its original value should the chain pass within some tolerable distance; this is similar to the 'drop proxies along the way' idea proposed in Bardenet et al. [4]. The bounds used for the simulation via thinning affect only the algorithmic efficiency - not the invariance or ergodicity properties of the chain, and so this is easily seen to be valid. We do not pursue this possibility however, as it is not possible to implement simultaneously with another improvement which we propose in the next subsection.

Another salient point that we see from the figures concerns the difference between the flip method and the reflection method; we have already seen evidence for the superiority of the reflection method in Section (4). These experiments strengthen that evidence, as *in each case, across each setting,* the reflection method is superior (both in terms of raw ESS and ESS/s) to the flip method, which indicates that it both mixes more quickly (raw ESS) and iterates more quickly (as, at least for the control variate method, the gulf in ESS/s greater on the log scale than the gulf in ESS). This is least pronounced for the alias method, because the additional step in the reflection algorithm required to sample from the marginal distribution of the dimension indices (see (31)) will increase the time required for an event-time to be computed; this step reduces the speed advantage that the reflection algorithm has over the flip algorithm in situations where the event-times can be simulated in a more straightforward fashion (e.g. for Gaussians - c.f. Figures 4,7). Of course, regardless of whether this step is implemented or not, both methods will require $O(d)$ steps to compute an event-time, although this will noticeably increase the constant factor for the reflection method.

5.4 Informed Sub-Sampling with Control Variates.

In the previous section, we saw the vast improvements in the performance of both the flip method and the reflection method that came as a result of using the informed sub-sampling method of Bouchard-Côté et al. [8], which achieves the factor selection step (19) in the same $O(1)$ time as the naive uniform sub-sampling of [7] without suffering the inefficiencies which result from having to use the same bound for each factor, which may dramatically reduce the number of events that lead to a flip or a reflection. Naturally, the magnitude of this gulf in performance depends on the nature of the data in question; the bounds will be worse in cases where the covariates tend to vary largely (in relative scale) from the means of their absolute values, e.g. when the covariates are drawn from heavy-tailed distributions or have large outliers. Additionally, the naive bounds will of

course be extremely sensitive to outliers. We also saw in the previous section, especially in low dimensions, the benefits to be gained by using the control variate technique to reduce the variance of the gradient estimators used by the sub-sampling methods. These improvements motivate the consideration of a method which combines both improvements, i.e. uses the control variate bounds and implements the informed sub-sampling via the alias method; this we introduce below.

Consider again the case of the reflection algorithm for logistic regression, and recall the Lipschitz bounds (43). For an individual observation $j \in \{1, \ldots, R\}$, we have then that

$$|\partial_i U^j(x_1) - \partial_i U^j(x_2)| \leq C_i^j \, \|x_1 - x_2\| \tag{44}$$

holds with

$$C_i^j = \frac{1}{4} |\xi_i^j| \left\|\xi^j\right\|_2, \tag{45}$$

and so, following the same steps as in Section 5.2 we have the bound

$$\widetilde{\lambda}_j(x(t), v(t)) \leq \frac{1}{R} \langle v, \nabla U(x^*) \rangle^+ + \sum_{i=1}^{d} C_i^j |v_i| \, (\, t \, ||v||_2 + ||x - x^*||_2) = M_r(t), \tag{46}$$

where, as before, $x^* \in \mathbb{R}^d$ is an arbitrary reference point. Making the further assumption that $x^* = \widehat{x}$ is the maximum likelihood estimate yields

$$M_r(t) = \sum_{i=1}^{d} C_i^j |v_i| \, (\, t \, ||v||_2 + ||x - x^*||_2) \, . \tag{47}$$

Once again, we consider a contrived distribution over the observation indices $j \in \{1, \ldots, R\}$ and the variable indices $i \in \{1, \ldots, d\}$. Let $\mathbb{P}(i, j)$ be given by

$$\mathbb{P}(i, j) \propto C_i^j |v_i|, \tag{48}$$

so that the marginal distribution of i is given by

$$\mathbb{P}(i) = \sum_j \mathbb{P}(i, j) \propto \mathbf{C}_i |v_i| \tag{49}$$

where $\mathbf{C}_i = \sum_j C_i^j$. By design, we see that the informed sub-sampling (19) that we wish to carry out may be achieved by sampling the marginal distribution of j, which is given by

$$\mathbb{P}(j) = \frac{M_r(t)}{\sum_r M_r(t)}. \tag{50}$$

This we may sample from by letting $i \sim \mathbb{P}_i(\cdot)$ and then taking $j \sim \mathbb{P}_{j|i}(\cdot|i)$ from the conditional which by (48) and (49) is given by

$$\mathbb{P}_{j|i}(j|i) = \frac{C_i^j}{\mathbf{C}_i}. \tag{51}$$

After constructing the alias table dictated by (51), we may thus carry out informed sub-sampling using control-variate bounds in $O(1)$ time in R. We do not derive the procedure in the case of the flipping algorithm; the procedure is identical save for the fact that there is no need to recourse to the synthetic distribution as the events are determined on a component-by-component basis, and the required alias tables are the same.

5.5 Further Experiments.

In this section we reconsider the experiments discussed above, and compare the results that we observed with the performance of the control variate method with informed sub-sampling.

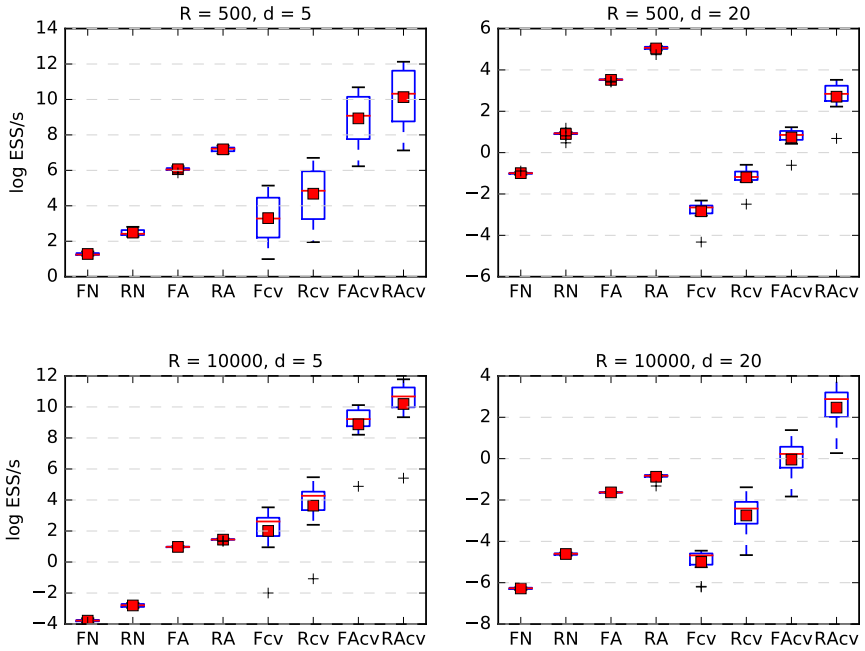

Figure 10: Same as Figure 8 above, except two boxplots for each scenario have been added to display the results of the Flip/Reflect algorithms using informed-subsampling with control variates (respectively FAcv and RAcv).

Figure 10 displays the results. As expected, we see that the use of informed sub-sampling leads to substantial gains in efficiency. Of course, as we saw earlier, this new method still suffers a severe drop in performance as the dimension increases due to the control variate bound, though with informed sub-sampling we see that they outperform all the other methods, except in the $R = 500$, $d = 20$ case, where R is not yet high enough relative to d for the benefits of the use of control variates to be decisive. Note that once again the reflection algorithm has outperformed its flip counterpart in each setting.

5.6 On Scaling, and the Advantages of Informed Sub-Sampling.

In this section, we will consider how the reflection algorithms presented above scale for big data, i.e. as the number n of data points becomes large, and we quantify the difference between naive sub-sampling and informed sub-sampling using the alias method in the control variate setting, which we illustrate with an example. Throughout, we will closely follow the analysis shown in Bierkens et al. [7], where analogous arguments are laid out for the flip algorithm.

5.6.1 Scaling of the Reflection Algorithm.

Let $n \in \mathbb{N}$ and suppose that the energy function may be expressed as

$$\begin{aligned} U(\theta) &= -\sum_{j=1}^{n} \log f(y^j|\theta) \\ &= -\sum_{j=1}^{n} U^j(\theta), \end{aligned}$$

where the observations y^j are drawn independently from the data generating distribution $f(y^j|\theta_0)$. Letting $\widehat{\theta}$ denote the maximum likelihood estimator of θ based on observations $y^1, \ldots, y^n$, and let $\phi(\theta) = \sqrt{n}(\theta - \hat{\theta})$, so that $\theta(\phi) = n^{-1/2}\phi + \hat{\theta}$. Now, in the limit as $n \to \infty$, the posterior distribution with respect to the variable ϕ will converge to a zero mean multivariate normal distribution with covariance given by $\mathcal{I}(\theta_0)^{-1}$, the inverse of the expected Fisher information [7, 21]. To analyse the limit of the event rate, we expand the gradient of the energy function around $\hat{\theta}$, yielding

$$\begin{aligned} \nabla_i U(\theta) &= \nabla_i U(\hat{\theta}) + \sum_{j=1}^{n}\sum_{k=1}^{d} \partial_i\partial_k U^j(\hat{\theta})(\theta_k - \hat{\theta}_k) + O(|\theta - \hat{\theta}|^2) \\ &= \sum_{j=1}^{n}\sum_{k=1}^{d} \partial_i\partial_k U^j(\hat{\theta})(\theta_k - \hat{\theta}_k) + O(|\theta - \hat{\theta}|^2) \end{aligned}$$

where $\partial_i U(\theta) = \partial/\partial\theta_i U(\theta)$, which follows from the multivariate analogue of Taylor's theorem and the fact that $\hat{\theta}$ is the maximum likelihood estimate. The intensity of the non-homogeneous Poisson process which determines the event times can thus be expressed, in terms of ϕ, as

$$\langle v, \nabla U(\theta)\rangle^+ = n^{-1/2}\left(\sum_{i=1}^{d} v_i \left(\sum_{j=1}^{n}\sum_{k=1}^{d} \partial_i\partial_k U^j(\hat{\theta})\phi_k\right)\right)^+ + O\left(\frac{\|\phi\|^2}{n}\right); \qquad (52)$$

note that the first term on the left-hand side is $O(n^{1/2})$ by the law of large numbers (note that ϕ is $O(1)$ - e.g. [36]). Arguing as in [7], we observe that in terms of ϕ, the process has velocity given by $n^{1/2}v$, and so after a time-scale transformation by $n^{1/2}$, we recover

a velocity of v and the intensity becomes

$$\left(\frac{1}{n}\sum_{i=1}^{d} v_i \left(\sum_{j=1}^{n}\sum_{k=1}^{d} \partial_i\partial_k U^j(\hat{\theta})\phi_k\right)\right)^+ + O(n^{-1/2}), \tag{53}$$

as $\|\phi\|$ is $O(1)$. By the strong law of large numbers, the above converges to

$$\widetilde{\lambda}(\phi, v) = \langle v, \mathcal{I}(\theta_0)\phi\rangle^+ \tag{54}$$

with probability 1, which is precisely the intensity arising from a Gaussian distribution with zero mean and covariance matrix $\mathcal{I}(\theta_0)^{-1}$. Now, as this expression is now free from dependence on n, we see, assuming we are starting from the stationary distribution, that an approximately independent point will be reached within a time interval of $O(1)$; in the original time scale, this corresponds to a time interval of $O(n^{-1/2})$. Provided that the bound on the intensity is of order no greater than $O(n^{1/2})$, this interval will be realized after $O(1)$ candidate event-times are proposed. If the algorithm is implemented without sub-sampling, then the cost of accepting or rejecting an event-time is $O(n)$, as the energy gradient must be calculated with respect to all of the data points; thus, the computational complexity of obtaining an independent point using the basic reflection algorithm is $O(n)$ as long as the bound on the intensity is $O(n^{1/2})$. The same is true for the flipping algorithm [7].

5.6.2 Scaling of the Reflection Algorithm with Control Variates.

Consider now the case in which Lipschitz bounds are used to bound the intensity given by (36) and (37). Suppose, for now, that there exist Lipschitz bounds such that the constants C_i (as in (34)) are uniformly $O(1)$ in $j = 1, \ldots, n$ (more on this later), with $p = 2$ for definiteness. Suppose further that the reference points θ^* in (35) are such that $\left\|\theta^* - \hat{\theta}\right\|$ is $O(n^{-1/2})$. Recall the expression for the estimate of the energy:

$$\widetilde{U}^j(\theta) = \frac{1}{n}U(\theta^*) + U^j(\theta) - U^j(\theta^*). \tag{55}$$

Taking the gradient and examining the i-th component yields

$$\begin{aligned}
\left\|\nabla_i \widetilde{U}^j(\theta)\right\| &= \left\|\frac{1}{n}\partial_i U(\theta^*) + \partial_i U^j(\theta) - \partial_i U^j(\theta^*)\right\| \\
&= \left\|\frac{1}{n}\partial_i U(\theta^*) - \frac{1}{n}\partial_i U(\hat{\theta}) + \partial_i U^j(\theta) - \partial_i U^j(\theta^*)\right\| \\
&\leq C_i \left\|\theta^* - \hat{\theta}\right\|_2 + C_i \left\|\theta - \hat{\theta}\right\|_2 \\
&= O(1) \times O(n^{-1/2}) + O(1) \times O(n^{-1/2}) \\
&= O(n^{-1/2}),
\end{aligned}$$

where the second equality follows because $\hat{\theta}$ is the MLE, the first inequality follows from the Lipschitz assumption, and the third follows from the reference point assumption and standard MLE asymptotics (see e.g. Shao [36]). Assume now, for simplicity of presentation (and because the informed sub-sampling requires it), that $\theta^* = \hat{\theta}$, i.e. that the reference point is the MLE. As above, we want an expression for the limiting intensity as $n \to \infty$. Observe that in this case, after rescaling by $n^{1/2}$, we have that

$$\begin{aligned} n^{1/2}\partial_i \widetilde{U}^j(\theta) &= n^{1/2}\left(\partial_i U^j(\theta) - \partial_i U^j(\hat{\theta})\right) \\ &= n^{1/2}\sum_{k=1}^{d} \partial_i\partial_k U^j(\hat{\theta})(\theta_k - \hat{\theta}_k) + O(n^{1/2}|\theta - \hat{\theta}|^2) \\ &= \sum_{k=1}^{d} \partial_i\partial_k U^j(\hat{\theta})\phi_k + O(n^{-1/2}). \end{aligned}$$

where we have used a multivariate Taylor expansion, ϕ as above, and the fact that $(\theta - \hat{\theta})$ is $O(n^{-1/2})$. Before we proceed, we will require the following result, which follows from the proof of the validity of the sub-sampling reflection algorithm (see appendix of Bouchard-Côté et al. [8]), although we present a self-contained version:

Lemma 5.2 *Let $\lambda^j(t)$ denote the true intensity from the j-th observation, and let $\lambda^j(t) \leq m^j(t)$ be an upper bound used for thinning; let $M(t) = \sum_j m^j(t)$. Then the event-times of the sub-sampling algorithm are generated according to the effective rate function*

$$\lambda(t) = \sum_{j=1}^{n} \lambda^j(t). \tag{56}$$

Proof Let τ denote a candidate event time resulting from the sub-sampling algorithm. Then conditional on τ, the probability of a reflection event occuring at that point is easily seen to be

$$\mathbb{E}_J\left[\frac{\lambda^j(\tau)}{m^j(\tau)}\right] = \sum_{j=1}^{n} \frac{\lambda^j(\tau)}{m^j(\tau)}\frac{m^j(\tau)}{M(\tau)} = \frac{\sum_{j=1}^{n}\lambda^j(\tau)}{M(\tau)}. \tag{57}$$

Since τ was generated as the first arrival time of the process with intensity $M(t)$, the result follows by Proposition 3.1.

Using this result, the effective time re-scaled intensity function is given, in terms of ϕ, by

$$\begin{aligned} \widetilde{\lambda}(\phi, v) = n^{-1/2}\lambda(\phi, v) &= \frac{1}{n}\sum_{j=1}^{n} n^{1/2}\langle v, \widetilde{U}^j(\theta(\phi))\rangle^+ \\ &= \frac{1}{n}\sum_{j=1}^{n}\left(\sum_{i=1}^{d} v_i \sum_{k=1}^{d} \partial_i\partial_k U^j(\hat{\theta})\phi_k\right)^+ + O(n^{-1/2}) \\ &\to \mathbb{E}_Y\left[-\left(\sum_{i=1}^{d} v_i \sum_{k=1}^{d} \partial_i\partial_k \log f(Y|\theta_0)\right)^+\right] = O(1), \end{aligned}$$

where the third equality follows from the expression for $n^{1/2}\widetilde{U}^j(\theta)$ derived above, and the convergence follows from the bound on $\left\|\widetilde{U}^j(\theta)\right\|$. Once again, all dependence on n has vanished in the limiting intensity, and so following the arguments given above we see an approximately independent point will be reached by the process after $O(1)$ proposed events, in this case however, the sub-sampling ensures that the cost of an iteration is $O(1)$, and so provided the Lipschitz constants satisfy $C_i^j = O(1)$ then we see that the computational complexity per independent sample of the reflection algorithm with control variates is $O(1)$ - an order-n increase in efficiency compared to the basic algorithm; the same holds for the flip algorithm [7]. This allows us to conclude, remarkably, quoting Bierkens et al. [7], that we have "*an unbiased algorithm for which the computational cost of obtaining an independent sample does not depend on the size of the data*" .

Using the above, it is not hard to perceive the advantages offered by the alias sub-sampling method. Consider for example the Lipschitz constants C_i for the logistic regression example (43); in this case, the need to take a maximum over the observations means that depending on the distribution from which the covariates are drawn, the bound may not be $O(1)$ - indeed, while trivially it will be $O(1)$ if the covariates are taken from a bounded set, if they are drawn, for example, from a (sub) Gaussian distribution, then we will have $C_i = O(\log n)$ [7]; distributions with heavier tails will result in even worse scaling. However, using the alias method will always preclude the need to take a maximum over n, ensuring that no matter the distribution of the covariates the $O(1)$ bound will hold and the above analysis will be valid. Thus we see that when $C_i = O(1)$, the increase in efficiency due to the use of informed sub-sampling method will be a constant factor, although when C_i are of higher order, the relative efficiency will increase with n. We illustrate this with a series of experiments below. Figure 11 shows the results of the control variate method both with and without informed sub-sampling on four settings on datasets of increasing dimension. In the first case, the covariates are drawn from a Gaussian, in the second they are drawn from a Student-t distribution with 3 degrees of freedom, and in the third and fourth they are drawn from a uniform distribution on $(0, 1)$ - the fourth setting has a single outlier drawn from $U(0, 10)$.

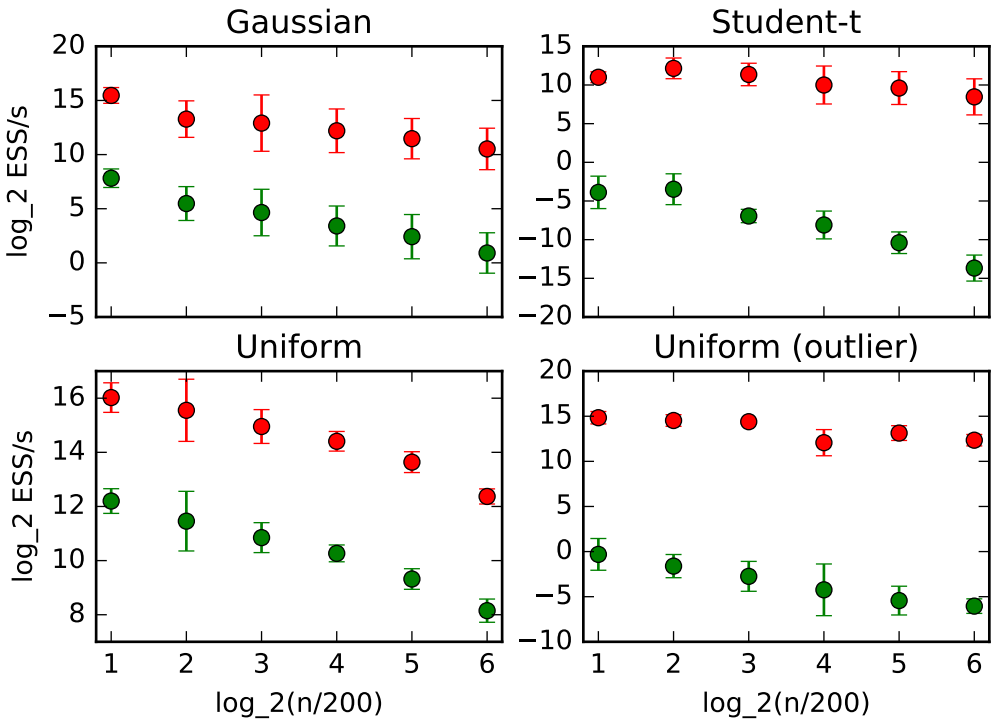

Figure 11: Mean ESS/s with error bars over 6 runs of the reflection algorithm using control variates (green) and control variates with alias sub-sampling (red). Covariates are generated as the absolute values of (clockwise from top left): standard Gaussian, Student-t with df $= 3$, Uniform $(0, 1)$ with an outlier that is $\mathrm{U}(0, 10)$, and Uniform $(0, 1)$. In each case the method was run for 10^6 iterations.

The above figure clearly illustrates the advantages of the alias sub-sampling method. The top two plots demonstrate that the efficiency gain from the alias method grows with n when the bounds C_i are of order 1 in n. As expected, the gain is greater when the tails are heavier, being hardly perceptible when the covariates are Gaussian, and marked when they are Student-t distributed. We are reminded however of the limitations of the above analysis, as the decreasing ESS/s with increasing n indicates that the mixing time does decrease substantially as the size of the dataset grows - recall that the arguments presented above hold under the condition that the processes have reached the stationary distribution. The bottom row displays another serious pitfall of naive sub-sampling, namely, the susceptibility to outliers. The need to take a bound uniform in n (see (43)) means that even a single outlier can dramatically worsen the performance of the method, while the alias method of course does not suffer.

5.7 Limitations.

While the performance seen above is encouraging, it is important to carefully consider the scope and limitations of these methods. As mentioned above, a key advantage of the ECMC algorithms that we have considered here is their amenability to exact sub-sampling for Bayesian applications - unlike MH algorithms for which sub-sampling methods are

usually inexact [4]. A notable exception is the FlyMC algorithm of MacLaurin and Adams [25] mentioned above; however, this was shown in Bouchard-Côté et al. [8] to be less efficient in terms of ESS/s than the reflection algorithm using the alias sampling method by roughly an order of magnitude for logistic regression, and as we have shown, the alias sampling method can be improved dramatically by using control variates. However, we note that the alias method suffers from several drawbacks. Firstly, in the context of the reflection algorithm, the need to sample from a distribution over the d components of the density (see (31), (49)) will markedly reduce the speed in high dimensions, although it will still be superior to naive sub-sampling. Secondly and more importantly, the alias set-up is problem dependent, and in many instances it will not be possible to implement. However, provided the maximum likelihood estimator exists and can be computed, *it will always be possible to implement informed-sub-sampling via the alias method while using the control variate bounds.* This can be seen by inspection of (47), as only the C_i^j terms are problem-dependent; these are constants, so the set-up for informed sub-sampling will be identical (the choice of L^p norm will also be problem dependent, although again, this difference will not affect the derivation of the set-up). This brings us to the next limitation: namely, the assumption that the control variate estimators are good. This will usually only be the case when the posterior distribution is approximately normal - i.e., when the posterior resembles its Bernstein-von Mises approximation [39, 4]. When the posterior is highly complex and/or multi-modal, this approximation will be poor, and the control variate method will fail. This is a problem shared by many MCMC methods that have been proposed to handle tall (large n) data sets; see Bardenet et al. [4] for further discussion. For multi-modal distributions, it may be possible to implement a procedure such as described above in which new reference points are computed at certain intervals, although they would have to be local posterior maxima for the alias method to work, and in any case the performance would likely be poor nonetheless.

"With four parameters I can fit an elephant. With five I can make him wiggle his trunk."

- John von Neumann

6 On Tuning Parameters and Exploiting Problem Geometry.

We have thus far remained aloof from any discussion regarding the tuning of the parameters of the algorithms which we have presented; indeed, the literature of event-chain Monte Carlo is unforthcoming on the subject. There is no mention at all of tuning in much of the physics literature - e.g. [22, 27, 28], while Peters and de With [33] briefly mention the inclusion of a mass matrix (see below) in the expression for the collision operator but then use the identity matrix for their experiments. Of the two papers concerning ECMC in the statistics literature, Bierkens et al. [7] state the possibility of using velocities of different scales for each component but do not elaborate, and though their proof of ergodicity relies on the presence of non-negative γ_i terms that we saw in (6) in the flipping rates, they do not make any further mention of them, and there is no indication as to what values they used for their experiments. Meanwhile, the coverage in Bouchard-Côté et al. [8] is more satisfying - they display a handful of figures indicating that the performance of the reflection algorithm is robust at low values (roughly between 0 and 1) of the refreshment parameter λ_0, and that performance degrades sharply at values of higher orders of magnitude; apart from one terse comment in their final example, they do not mention the mass matrix (again, see below) at all. In this section, we discuss the problem of tuning the parameters of the flip and reflection algorithms, and through numerical experiments give an indication of the gains that are achievable through thoughtful tuning, especially for the reflection algorithm.

6.1 Tuning of Flip-ECMC.

For a d-dimensional target distribution, the vanilla flip method uses velocities defined on $\{-1, 1\}^d$ to guide the variables of interest through the state-space. However, the algorithm remains valid if the unit velocities are scaled by factors α_i for $i = 1, 2, \ldots, d$, which means d tuning parameters. Furthermore, the functions $\gamma_i(x, v)$ alluded to above make for an additional d tuning *functions,* although the twin conditions $\gamma_i(x, v) = \gamma_i(x, F_i[v])$ and $\gamma_i > 0$, along with considerations involving convenience of simulation suggest that it will usually be best to select constant functions $\gamma_i(\cdot, \cdot) \equiv \gamma_i > 0$; thus we will say that the flip method has $2d$ tuning parameters in all.

6.1.1 The Speed Parameters.

As can be seen by in Figure 4, the flip method will struggle when the variables of the target distribution differ greatly in scale. With unit speeds in each direction, the components

with small variance will flip much more often than those with large variance, leading to poor mixing for the latter. Below in Figure (refthisfigure) we show an extreme example of this. The (top row) trace plots clearly demonstrate the contrast in mixing speeds between the smallest and the largest component. Naturally, when the components of the target are independent, the obvious way to mitigate this problem is simply to let the parameters α_i vary in proportion to the marginal standard deviations of the variables they are associated to, so that the length of time that it takes to cross the distribution is roughly the same for each coordinate. This will ensure that the flipping events are evenly distributed across the d components, and will ensure that mixing times are comparable; of course, this will mean that the components of smaller variance will mix more slowly relative to the case when unit speeds are used. Figure 12 below illustrates this phenomenon.

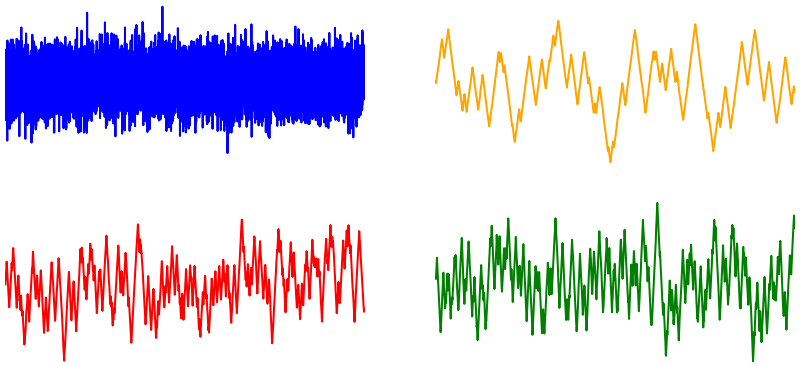

Figure 12: Trace plots for the 1st and 100th components of a 100-d Gaussian distribution with standard deviations $0.01, 0.02, \ldots, 1.00$ for two runs of the flip algorithm, each run consisting of 20000 events. Top row shows results with unit speed in every direction (1st component - blue, 100th component - orange) and bottom row with speed proportional to standard deviation (1st - red, 100th - green).

When the variables are highly correlated as well as being of different scales, the solution is by no means so obvious, and setting speeds in proportion to the standard deviations may not be the optimal thing to do (see Neal [31]), although it will likely still be an improvement over using the same speed for each component.

6.1.2 The Gamma Parameters.

The optimal values for the parameters γ_i are less clear. For reasons that we shall now discuss, in our simulations we used small values, so that $\gamma_i = \gamma \approx 0$ for all $i = 1, \ldots, d$; we suspect that Bierkens et al. [7] either did the same or simply set $\gamma_i = 0$ - their theorem on the ergodicity of the flipping algorithm requires $\gamma_i > 0$, although they conjecture that this condition is not necessary in many cases. Furthermore, unlike the case of the reflection algorithm in which the refreshment parameter λ_0 is essential to avoid spending too much time going in 'bad directions' - for an extreme example see Figure 3 of Bouchard-Côté

et al. [8] - in the flip algorithm the directions of motion are fixed in a discrete set and the relative magnitudes of the speeds are fixed. Therefore it seems likely that higher values of γ_i will simply result in a larger degree of random-walk behaviour, which it is of course desirable to avoid; without the incentive to refresh more often that is a factor in the reflect method, we see no reason to choose anything other than very small values of γ. Indeed, in a recent preprint Bierkens and Duncan [6], in the one-dimensional case the authors show that for large γ the process does resemble a random-walk, and in the limit as $\gamma \to \infty$ the (time-rescaled) process converges to an over-damped Langevin diffusion.

6.2 Tuning of Reflect-ECMC.

We have already briefly mentioned the λ_0 parameter, which determines the rate at which the velocity variables are re-sampled, and thus the ratio of re-sampling events to reflection events. Above we allude to another set of parameters: the mass matrix M - the presentation of the reflection algorithm given in Section 2 and all of our experiments conducted thus far have used the special case $M = I_d$, however, the algorithm remains valid if we select M to be a symmetric positive-definite matrix, and let the marginal distribution of the velocity variables given by $v \sim \mathrm{N}(0, M)$ and use the following modification of the reflection operator:

$$R[x]v = \left(I_d - 2\frac{M\nabla U(x)\nabla U(x)^t}{\nabla U(x)^T M \nabla U(x)}\right) v. \tag{58}$$

The properties of the reflection operator which are required to ensure the correctness of the reflection algorithm are that $R[x]^T \nabla U(x) = -\nabla U(x)$ and that $\psi(v) = \psi(R[x]v)$. These are straightforward to verify - indeed, we have

$$\begin{aligned} R[x]\nabla U(x) &= \left(I_d - 2\frac{M\nabla U(x)\nabla U(x)^t}{\nabla U(x)^T M \nabla U(x)}\right)^T \nabla U(x) \\ &= \nabla U(x) - 2\frac{\nabla U(x)\nabla U(x)^t M^T \nabla U(x)}{\nabla U(x)^T M \nabla U(x)} \\ &= \nabla U(x) - 2\nabla U(x) \\ &= \nabla U(x). \end{aligned}$$

where we have used the symmetry of M. To see that ψ is preserved under $R[\cdot]$, we let $v' = R[x]v$ and observe that

$$\begin{aligned} v'^T M^{-1} v' &= \left(v^T - 2\frac{\langle v, \nabla U(x)\rangle \nabla U(x)^t M}{\nabla U(x)^T M \nabla U(x)}\right) M^{-1} \left(v - 2\frac{M\nabla U(x)\langle v, \nabla U(x)\rangle}{\nabla U(x)^T M \nabla U(x)}\right) \\ &= v^T M^{-1} v - 2\frac{\langle v, \nabla U(x)\rangle^2}{\nabla U(x)^T M \nabla U(x)} \\ &\quad - 2\frac{\langle v, \nabla U(x)\rangle^2}{\nabla U(x)^T M \nabla U(x)} + 4\frac{\langle v, \nabla U(x)\rangle^2 \nabla U(x)^T M \nabla U(x)}{\left(\nabla U(x)^T M \nabla U(x)\right)^2} \\ &= v^T M^{-1} v, \end{aligned}$$

and since $\psi(v) = f(v^T M^{-1} v)$, the result follows.

Since M is symmetric, this makes for $d + (d^2 - d)/2 = (d^2 + d)/2$ parameters to tune.

6.2.1 The Refreshment Parameter.

To gain insight into how the value of λ_0 affects the dynamics of the reflection algorithm, it helps to understand how it interacts with another quantity: $\|v\|$, the magnitude of the velocity. It is easy to see that what matters is not the size of either of these quantities, but rather their ratio. To see this, let $v = \|v\| u$, where u is a unit vector. In general, the intensity of the non-homogeneous Poisson process that determines the time until the next event can be expressed as $\lambda(t) = \lambda_0 + \langle v, \nabla U(x(t))\rangle^+ = \lambda_0 + \|v\| \langle u, \nabla U(x + t\|v\| u)\rangle^+$. If we scale both λ_0 and $\|v\|$ by the same constant $\alpha > 0$, then the intensity becomes $\lambda_\alpha(s) = \alpha(\lambda_0 + \|v\| \langle u, \nabla U(x + s\|v\| u)\rangle^+) = \alpha\lambda(t\alpha)$, where $s = \alpha t$. By (8), the first arrival time of the scaled process is the solution τ to the equation $\int_0^{\tau'} \lambda_\alpha(s)\, ds = -\log(U)$ where $U \sim \mathrm{U}(0, 1)$. Making the substitution $s = t/\alpha$ yields

$$\begin{aligned} -\log(U) &= \int_0^{\tau'} \lambda_\alpha(s)\, ds \\ &= \int_0^{\alpha\tau'} \frac{1}{\alpha} \lambda_\alpha(t/\alpha)\, dt \\ &= \int_0^{\alpha\tau'} \lambda(t)\, dt = \int_0^{\tau} \lambda(t)\, dt, \end{aligned}$$

where the last line expresses the equation for the first arrival time of the process with intensity $\lambda(t)$. Thus we see that if τ' is the first arrival time of the scaled process, then $\tau = \alpha\tau'$ is the first arrival time of the original process. We have $x + \tau' v' = x + (\tau/\alpha)v' = x + \tau v$, and so the trajectories of the two processes are identical. When an event occurs, it corresponds to either a re-sampling event or a reflection event. Using standard results concerning the Poisson process, we have that the probability that an event occurring at time τ is a re-sampling event is given by

$$\frac{\lambda_0}{\lambda_0 + \langle v, \nabla U(x(\tau))\rangle^+} = \frac{1}{1 + \frac{\|v\|}{\lambda_0} \langle u, \nabla U(x(\tau)\rangle^+}, \tag{59}$$

which again only depends on the ratio $\|v\| / \lambda_0$ and is unchanged under rescaling by α.

This insight does not guide us in the selection of λ_0, although it does help to explain the fact that the algorithm in most cases highly robust to this selection, even for values differing by orders of magnitude. When the velocity variables are drawn from a distribution like a multivariate Gaussian, $\|v\|$ is not fixed, and so the ratio $\|v\| / \lambda_0$ will change after every event. This is similar in flavour to contexts involving other algorithms where parameters are chosen randomly from some interval (e.g. when using HMC, it is common to randomly select the number of leapfrog steps per iteration from some integer lattice, i.e. l uniformly in $\{L - H, L + H\}$ for some integers $H < L$). This would lead us to expect that if the distribution ψ of v is such that $\|v\|$ is fixed, (say, if ψ is the uniform distribution on $\mathcal{S}^{d-1}$, the d-dimensional hypersphere) then the reflection algorithm will

be more sensitive to choice of λ_0, and indeed this does prove to be the case - see Figure 5 of Bouchard-Côté et al. [8]. We note that when $v \sim \mathrm{N}(0, I_d)$, then $\|v\| \sim \chi_d$, a Chi distribution with d degrees of freedom. As $d \to \infty$, the variance of this distribution stabilizes, never exceeding $1/2$, and so if we increase λ_0 with d to make the expectation of the ratio $\|v\| / \lambda_0$ constant, the variance of this quantity will tend to zero; thus we might expect the sensitivity to λ_0 to increase with dimension. In any particular case, this may be corrected for by introducing a different marginal distribution for the velocities, for example one could draw v as usual, and draw a quantity $s \sim \mathrm{U}(\mathbb{E}\|v\| - \alpha, \mathbb{E}\|v\| + \alpha)$ for some $0 < \alpha < \mathbb{E}\|v\|$ and then scale v to have norm s; in this case one could alter α to give $\|v\| / \lambda_0$ the desired variance. As long as the same scheme were observed at every re-sampling event, the algorithm would be correct.

We note that while the reflection algorithm is usually quite insensitive to small values of λ_0, performance generally degrades sharply for values above a certain problem-dependent threshold, above which the velocity variables will be re-sampled often and the dynamics of the chain will tend towards random-walk behaviour; c.f. Figures 5 and 13 of Bouchard-Côté et al. [8]. In our experience we have found that one or two trial runs often suffice to find a value of λ_0 which will yield near-optimal performance and thus, unlike other algorithms that are highly sensitive to parameter settings, e.g. HMC, where performance can vary drastically even under small perturbations of the tuning parameters (ϵ, L), it is unnecessary to devote much (if any) computation time to determining acceptable settings.

6.2.2 The Mass Matrix.

The mass matrix of the reflection algorithm plays a role very similar to the mass matrix in HMC; choosing M to be other than the identity will lead to certain direction of motion being favoured much more highly than others. In the case of HMC, it is known [31, 17] that careful tuning of the mass matrix can often lead to significant improvement. While in many cases HMC will perform very well with an identity mass matrix, for problems with high correlation between variables choosing a non-diagonal M is often essential. In Figure 13 below, we demonstrate the potential efficiency gains that are obtainable when M is properly chosen. Figure 13 shows the results of repeating the experiment from Section 4 with a 100-dimensional Gaussian target with a noisy covariance matrix $\Sigma = LL^T$ where $L_{ij} \sim \mathrm{N}(0, 1)$ using the reflection algorithm with an identity mass matrix, and with a mass matrix Σ, which corresponds to the inverse of the Hessian matrix of the energy function.

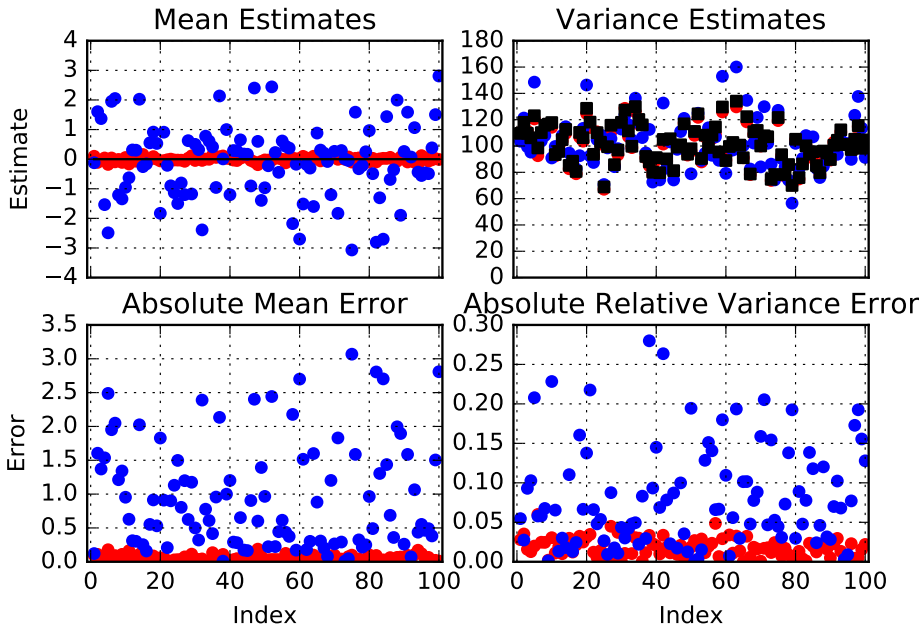

Figure 13: Clockwise from top left: estimates of the mean, estimates of the variance, absolute relative error of variance estimates, and absolute error of mean estimates for each component of a one-hundred dimensional Gaussian target distribution from trajectories of 50000 events for the reflection method with identity mass matrix (blue) and mass matrix given by the true covariance matrix (red).

As we see, using $M = \Sigma$ led to dramatic improvement in performance. Of course, this is an ideal scenario; in practice we will not have such precise knowledge of the true covariance matrix of the target. For the HMC algorithm, much work has been done with the goal of selecting M when knowledge of the target density is unavailable. Heuristics have been proposed, see for example Liu [24] and Neal [29, 30, 31], although these are not wholly satisfactory, as they rely upon knowledge of the scales of the variables, which will usually require preliminary runs of the algorithm to obtain [17]. Adaptive methods (see e.g. Andrieu and Thoms [2]) may be provide hope of a solution, although while setting parameters adaptively can often work well when the number of parameters is low, adaptively setting a mass matrix with $(d^2 + d)/2$ parameters is likely to be very costly - see Roberts and Rosenthal [35] for an instance of a proposal covariance matrix being set adaptively for a Metropolis-Hastings algorithm. In Girolami and Calderhead [17], the authors implement a scheme which they call 'Riemannian Manifold' HMC, in which the mass matrix is a function of the current position; specifically, inspired by geometric ideas introduced in Rao [34], they employ the Fisher-Rao metric tensor at x as $M(x)$. This defines a distance on the Riemannian manifold of the parameter space, and is equal to the expected Fisher information [17]. This induces a non-separable Hamiltonian, and the corresponding equations driving the dynamics are more difficult to handle. In the basic ECMC setting, this framework is infeasible, as the piecewise linear trajectories of the algorithms would not leave the target distribution invariant if the position variables were not marginally independent of the velocity variables; however, encouraged by the

success of their approach, we may hope that using a constant approximation to the expected information as a mass matrix for the reflection algorithm may yield significant improvements over the identity. Below we investigate.

6.3 Example: Real Data.

In this section we consider the performance of the reflection algorithm with varying mass matrix for logistic regression on two real datasets: the first consisting of steel plate faults data, which can be found at `https://archive.ics.uci.edu/ml/datasets/Steel+Plates+Faults`, and the second consisting of skin segmentation data, which can be found at `https:// archive.ics.uci.edu/ml/datasets/Skin+Segmentation`. For details, we refer the reader to the original papers: Buscema et al. [9] and Bhatt et al. [5]. The faults data set exhibits quasi-complete separation, so we preprocessed by removing several features; furthermore, we rescale both datasets so that each column of the design matrix has unit variance. After preprocessing, the faults dataset had 1941 observations with 23 covariates, while the skin dataset had 245057 observations with 3 covariates. We use the alias method on the faults data, and the alias method with control variates on the skin data.

As demonstrated in Girolami and Calderhead [17], the expected Fisher information for logistic regression is given by

$$\mathcal{I}(\beta) = X^T \Lambda X \tag{60}$$

where Λ is a diagonal matrix with n-th diagonal entry given by $\Lambda_{n,n} = s(\beta^T X_n^T)/(1 - s(\beta^T X_n^T)$ where $s(\cdot)$ is the logistic function and X_n is the n-th row of the design matrix. Since this is non-constant, we must make an approximation to it. Thus we consider the three following matrices:

$$\begin{aligned} G_1 &= X^T X, \\ G_2 &= I(\widehat{\beta}), \\ G_3 &= \mathrm{diag}(I(\widehat{\beta})), \end{aligned}$$

and use the mass matrices $M_i = G_i^{-1}$, and compare with $M_0 = I_d$. Note that it was necessary to rescale the mass matrices so that the diagonal entries had mean one; this is so that the expected ratio $\|v\| / \lambda_0$ was of similar order of magnitude, which ensures that keeping λ_0 constant across methods is appropriate. The results are given in Tables 6.3 and 6.3 below.

Tables 1 and 2 show mixed results. For the faults data, we see that using a mass matrix improves efficiency by a factor of at least 10 in each case, while the identity matrix works

Mass	Time (s)	Min ESS	Med ESS	Max ESS	Min ESS/s	Relative Speed
I_d	168.9	15410	15420	15780	91.2	1
M_1	168.0	12990	18390	21770	77.3	0.85
M_2	169.4	9300	10210	15510	54.9	0.60
M_3	168.5	11550	13110	14280	68.5	0.75

Table 1: Effective Sample Sizes for the skin segmentation data. Each method ran for 2×10^6 iterations.

Mass	Time (s)	Min ESS	Med ESS	Max ESS	Min ESS/s	Relative Speed
I_d	325.7	28	30	37	0.0860	1
M_1	325.1	384	387	406	1.18	13.7
M_2	323.8	292	294	305	0.902	10.5
M_3	329.5	286	290	307	0.868	10.1

Table 2: Effective Sample Sizes for the steel plates faults data. Each method ran for 2×10^6 iterations.

best for the skin data. Naturally (as in this case we can only use an approximation to the Fisher information), the closer the posterior is to a constant curvature surface, the better we expect this method to work.

Since the mass matrix $M_1 = X^T X$ is the top performing non-identity mass matrix in each case, we recommend giving it a trial run when using ECMC for logistic regression in practice. In cases when the Laplace approximation at the maximum likelihood estimator is good, we would suggest trying M_2.

6.4 Example: Poisson-Gaussian Markov Random Field.

We turn our attention to the problem of sampling from the distribution of a latent Gaussian field arising from a Poisson-Gaussian Markov random field model (also referred to as a log-Gaussian Cox point process). We use a lower-dimensional version of the model previously analysed in Christensen et al. [10], Girolami and Calderhead [17] and Wang et al. [40]. Specifically, we consider a dataset $\mathbf{Y} = \{y_{ij}\}$ consisting of counts at locations $(i, j) : i, j = 1, 2, \dots d$ on a $d \times d$ grid for $d = 30$; the problem is therefore of dimension $d^2 = 900$. The counts y_{ij} follow a Poisson distribution and are conditionally independent given a latent intensity process $\Lambda = \{\lambda_{ij}\}$ with means given by $s\lambda_{ij} = s \exp\{x_{ij}\}$ where $s = 1/d^2$, and $\mathbf{X} = \{x_{ij}\}$ is a Gaussian process with mean function $\mathbb{E}\mathbf{X} = \mu\mathbf{1}$ and covariance function

$$\Sigma_{(i,j),(i',j')} = \sigma^2 \exp\left\{-\delta(i, i', j, j')/30\beta\right\}, \tag{61}$$

where $\delta(i, i', j, j') = \sqrt{(i - i')^2 + (j - j')^2}$. Following Christensen et al. [10], we set $\sigma^2 = 1.91$ and $\mu = \log(126) - \sigma^2/2$, and we set $\beta = 1/6$; to ease the computational demands of the problem, we treat these parameters as fixed. The energy function

$U(\mathbf{x}) = -\log(\mathbf{x}|\mathbf{y}, \mu, \sigma, \beta)$ is easily seen to be proportional to

$$\sum_{i,j}(-y_{ij}x_{ij} + s\exp\{x_{ij}\}) + \frac{1}{2}(\mathbf{x} - \mu\mathbf{1})^T\Sigma^{-1}(\mathbf{x} - \mu\mathbf{1}) \tag{62}$$

$$= U_1(\mathbf{x}) + U_2(\mathbf{x}). \tag{63}$$

We have $\nabla U_2(x) = \Sigma^{-1}(\mathbf{x} - \mu\mathbf{1})$, while

$$\nabla_{ij}U_1(\mathbf{x}) = -y_{ij} + s\exp\{x_{ij}\}. \tag{64}$$

To simulate from the non-homogeneous Poisson process with intensity given by $\langle v, U(x(t))\rangle^+$, we use the superposition principle (10) using $U = \sum_{ij}(U_{11}^{ij} + U_{12}^{ij}) + U_2$ with $U_{11}^{ij} = -y_{ij}$ and $U_{12}^{ij} = d\exp\{x_{ij}\}$. We see that $U_2(x)$ is the energy function of a Gaussian distribution with mean $\mu\mathbf{1}$ and covariance matrix Σ, so we may simulate $\tau^{(2)}$ using (13). The intensities for U_{11}^{ij} and U_{12}^{ij} are given as functions of t by $-v_{ij}y_{ij}$ and $s\exp\{x_{ij} + tv_{ij}\})$ respectively, and so, using (8), we see that we may simulate $\tau_{ij}^{(11)}$ and $\tau_{ij}^{(12)}$ exactly by letting

$$\tau_{ij}^{(11)} = \begin{cases} \frac{\log(U)}{y_{ij}v_{ij}} & \text{if} \quad v_{ij} < 0 \\ \infty & \text{else,} \end{cases}$$

$$\tau_{ij}^{(12)} = \begin{cases} \frac{1}{v_{ij}}\left(\log\left(\frac{-\log(U)}{s} + \exp(x_{ij})\right) - x_{ij}\right) & \text{if} \quad v_{ij} > 0 \\ \infty & \text{else,} \end{cases}$$

where $U \sim \mathrm{U}(0, 1)$.

Below in Figure 14 we show the latent field, latent process, and observed data used for our example.

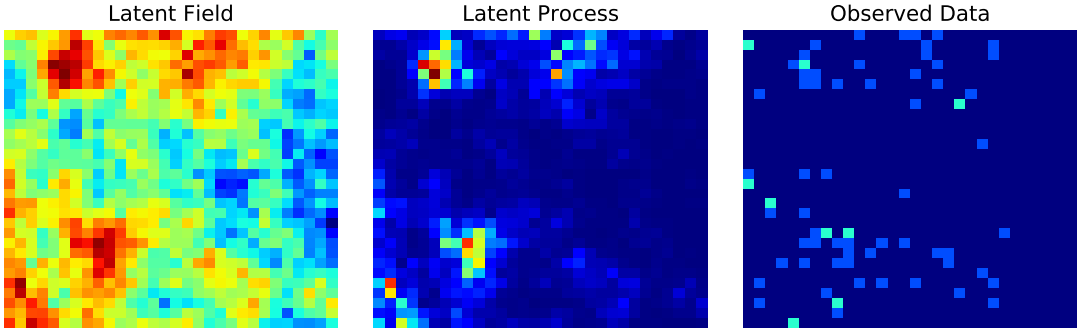

Figure 14: From left: latent random field $\mathbf{X}$, latent process Λ, and observed data $\mathbf{Y}$ with $d = 30$.

For this problem, as demonstrated in [17], the expected Fisher information is constant across the state space, and is given by

$$-\mathbb{E}_{\mathbf{x},\mathbf{y}}[\nabla_{\mathbf{x}}^2 U(\mathbf{x})] = L + \Sigma^{-1}, \tag{65}$$

Method	Time (s)	Min ESS	Med ESS	Max ESS	Min ES-S/s	Relative Speed
HMC	795.6	1070	4480	15910	1.34	1
RMHMC	783.5	6870	13780	20000	8.76	6.54
R-ECMC	1020.2	9640	10910	14160	9.44	7.04
R-ECMC (M)	1195.3	21340	23310	29280	17.85	13.3

Table 3: Effective Sample Sizes for a 30×30 random field. Row labels indicate HMC (identity mass), RMHMC (mass as above), R-ECMC (reflection algorithm with identity mass), R-ECMC (M) - with mass matrix as above.

where L is a diagonal matrix with entries $L_{ii} = m \exp\{\mu + \Sigma_{ii}\}$. Below we consider the performance of the reflection algorithm for sampling from the distribution of the latent field $\mathbf{X}$ using an identity mass matrix, and also using the matrix $M = (L+\Sigma^{-1})^{-1}$. Table 1 below compares the performance of these two instances of the reflection algorithm with the basic HMC algorithm, and with the RMHMC algorithm of Girolami and Calderhead [17]; note that in this instance, because the metric tensor is flat, RMHMC corresponds to an HMC algorithm with mass matrix M^{-1}. For the HMC methods, 20000 iterations were taken after 1000 iterations of burn-in, while the ECMC methods used 125000 events after 25000 burn-in events. The refreshment intensity was set to $\lambda_0 = 10$; the for the HMC methods we chose l steps chosen uniformly from $\{1, \ldots, L\}$ with stepsize ϵ, using $(L, \epsilon) = (100, 0.15)$ for HMC and $(L, \epsilon) = (50, 0.3)$ for RMHMC. These values were chosen after numerous trial runs, using ESS/s to select L and acceptance ratio to select ϵ - though we make no claim that these values are optimal.

Following Girolami and Calderhead [17], we use the minimum ESS/s across all variables as the performance metric. As we see in Table 6.4, the ECMC methods are most effective, although notably the reflection algorithm with identity mass yields the lowest maximum ESS/s, which explains the regions of high posterior variance seen in Figure 15 below. As expected, using the expected Fisher information as the mass matrix brings significant improvement to the reflection algorithm, albeit at the price of a higher computation time. This is of course no surprise, as the modified reflections (58) require a computation time in $O(d^2)$, while with diagonal mass they require only $O(d)$ time (see second equality in (2)). Thus we expect that using a non-diagonal mass will bring less benefit in very-high dimensions; however, it will still likely bring improvement if posterior correlations are very high. In the latter case, the best option may be to seek a parametrization under which variables are approximately independent in the posterior.

We close this section with the remark that, while the simulation recipe that we have employed produces highly competitive results, it is possible that the computation time per iteration could be significantly reduced by simulating the event times from the energy component $U_1(x)$ using numerical optimization methods to find a solve the equations (9). This would preclude the need to simulate d^2 candidate event-times and take the minimum of them, which is clearly the most computationally demanding step involved in

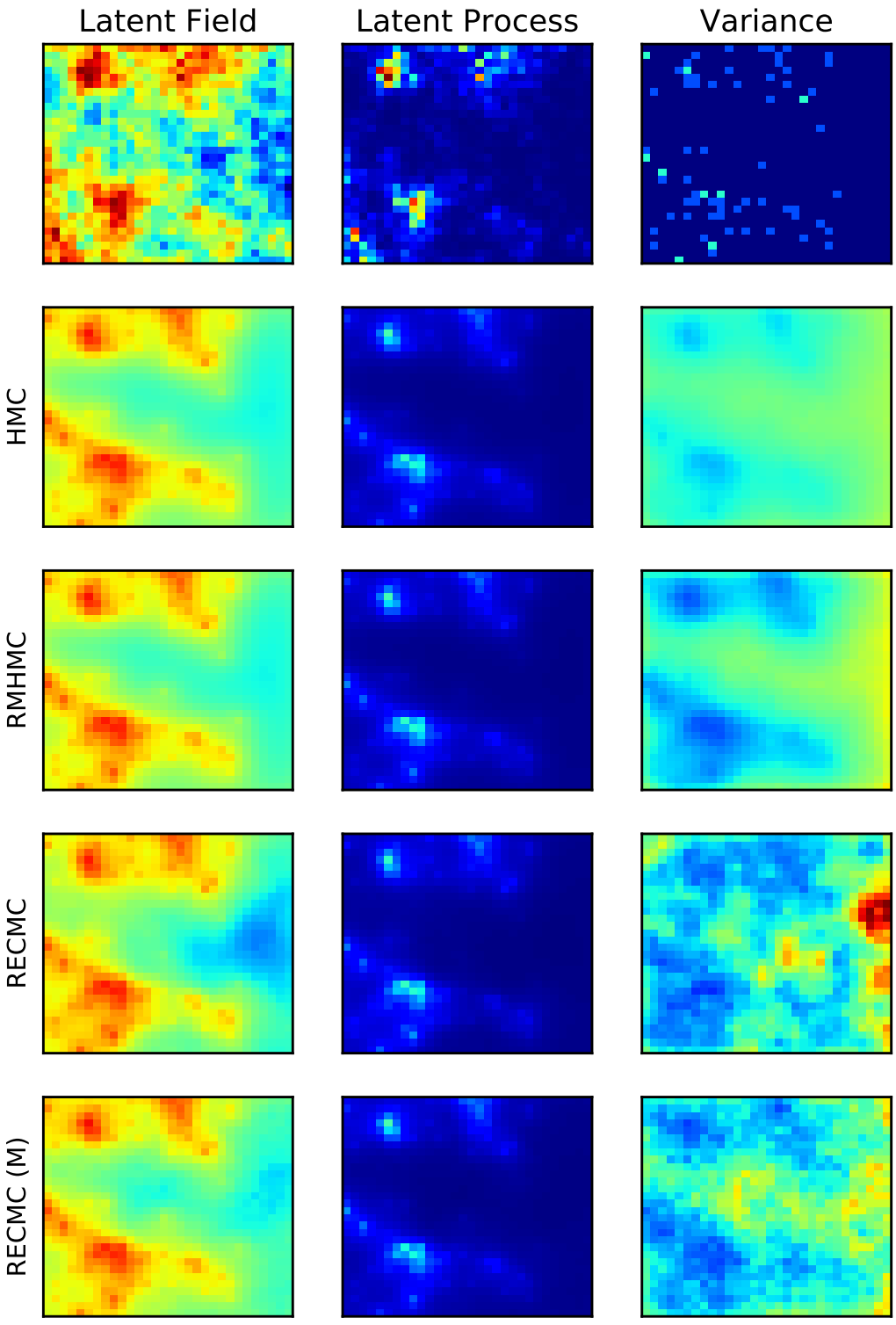

Figure 15: From left: posterior means of the latent random field $\mathbf{X}$,the latent process Λ, and the posterior variances of the latent field for HMC, RMHMC, and the reflection algorithm with identity mass (R-ECMC) and with mass as indicated in the text (R-ECMC (M)). Top row shows true latent field, process, and observed data.

the simulation.

"We can see but a short distance ahead, but we can see plenty that there needs to be done."

- Alan Turing

7 Conclusions and Further Work.

We conclude by noting that for a method so young (in relative terms), event-chain Monte Carlo methods are highly promising. They have been shown to be highly competitive with state-of-the-art HMC methods in several scenarios (RMHMC in this work, and a variety of HMC methods in Bouchard-Côté et al. [8]), and to be amenable to modifications that greatly facilitate big-data inference (this work and [8, 7]). It is to be hoped that with further study, new variants and modifications will be discovered that will bring ECMC even closer to mainstream use. Being simple to tune (as we saw above), ECMC has a great advantage over other efficient MCMC methods which require much labour before they can be made to run efficiently, e.g. HMC; while our final examples show that choosing an appropriate mass matrix can improve the algorithm significantly, in some cases it performs well (or better) even without this tuning.

As for the two ECMC algorithms we have considered, based on our experiments we conclude that the reflection algorithm is superior. It is in many cases considerably faster, as it avoids the need to simulate a candidate event-time for each dimension. Furthermore, it is more flexible, as the mass matrix allows for knowledge about the correlation between variables to be taken into account, while the flip algorithm can account for at most relative scale. We therefore recommend that future effort be directed towards the improvement of the reflection algorithm.

Geometric type methods akin to those employed in Girolami and Calderhead [17] are a promising avenue for future research, although the reflection algorithm would need to be generalized to allow for a joint density of the form $\rho(x, v) = \psi(v|x)\pi(x)$, which will be a challenge. It is also appealing to extend the algorithm so as to be able to sample efficiently from distributions arising from hierarchical models (indeed, work on this is already underway). In our final example, it was seen that the reflection algorithm performed exceedingly well on a Poisson-Gaussian Markov random field. However, the example was simplified tremendously by the hyper-parameters being fixed - a method to sample from the joint distribution of the latent field and the hyper-parameters is not so evident. In the discrete-time MCMC setting, Gibbs sampler style algorithms are able to handle such tasks; something similar could be achieved for ECMC by modifying the marginal distribution of the velocity variables in ECMC. For example, if $x' = (x, \alpha)$ where α is a vector of hyper-parameters, then if there were positive probability of drawing velocity vectors such as $v' = (v_1, 0)$ and $v'' = (0, v_2)$ where v_1,v_2 were of the same dimension as x, α respectively; this would yield a Gibbs flavoured set-up that would make sampling feasible for hierarchical models.

We finish by making some final comments on the limitations of the ECMC methods and variants which we have considered. As mentioned at the end of Section 5, the control

variate method - so successful for logistic regression - will not work unless the posterior resembles its Bernstein-von Mises approximation, which severely limits the usefulness of the method. This approximation will generally be excellent for large n, while for smaller n the sub-sampling methods are largely unnecessary; in other scenarios it may be altogether inaccurate, e.g. for multi-modal distributions. The lack of structural flexibility (i.e. for hierarchical models etc.) is another concern, although we expect that this will quickly be addressed. Finally, an obvious difficulty is the need to simulate the non-homogeneous Poisson process. This feature of the algorithm means that each new distribution encountered presents a potentially serious obstacle - in some cases there may simply be no efficient way to draw the event-times. In some ways however, this is a less serious drawback than the tuning difficulties of HMC, because once a method is devised to sample from an intensity arising from a given distribution, tweaking the parameters involved will not alter the simulation method, whereas different model parameters/dimensions can mean totally different optimal tuning parameter settings for HMC.

References

[1] C. Andrieu and G. O. Roberts. The pseudo-marginal approach for efficient Monte Carlo computations. *The Annals of Statistics*, 37(2):697–725, 2009.

[2] C. Andrieu and J. Thoms. A turotial on adaptive MCMC. *Statistics and Computing*, 18:343–373, 2008.

[3] R. Azais, J.-B. Bardet, A. Génadot, N. Krell, and P.-A. Zitt. Piecewise deterministic Markov processes - recent results. 2013. doi: http://www.arxiv.org/abs/1309.6061.

[4] R. Bardenet, A. Doucet, and C. Holmes. On Markov chain Monte Carlo methods for tall data. 2015. doi: http://www.arxiv.org/abs/1505.02827v1.

[5] R. Bhatt, G. Sharma, A. Dhall, and S. Chaudhury. Efficient skin region segmentation using low complexity fuzzy decision tree models. *IEEE-INDICON*, 2010.

[6] J. Bierkens and A. Duncan. Limit theorems for the zigzag process. 2016. doi: http://arxiv.org/pdf/1607.08845v1.pdf.

[7] J. Bierkens, P. Fearnhead, and G. Roberts. The zig-zag process and super-efficient sampling for Bayesian analysis of big data. 2016. doi: http://www.arxiv.org/abs/1607.03188v1.

[8] A. Bouchard-Côté, S. J. Vollmer, and A. Doucet. The bouncy particle sampler: A non-reversible rejection-free Markov chain Monte Carlo method. 2016.

[9] M. Buscema, S. Terzi, and W. Tastle. A new meta-classifier. *NAFIPS 2010, Toronto, Canada*.

[10] O. F. Christensen, G. O. Roberts, and J. S. Rosenthal. Scaling limits for the transient phase of local Metropolis-Hastings algorithms. *Journal of the Royal Statistical Society, Series B.*, 67(2):253–268, 2005.

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

[12] L. Devroye. *Non-uniform Random Variate Generation*. Springer-Verlag, New York, 1986.

[13] P. Diaconis, S. Holmes, and R. Neal. Analysis of a non-reversible Markov chain sampler. *Annals of Applied Probability*, 10:726–752, 2000.

[14] S. Duane, A. D. Kennedy, B. J. Pendleton, and D. Roweth. Hybrid Monte Carlo. *Phys. Lett. B.*, 195:216–222, 1987.

[15] A. E. Gelfand and A. F. M. Smith. Sampling-based approaches to calculating marginal densities. *J. Amer. Statist. Assoc.*, 85:398–409, 1990. doi: http://www.ams.org/mathscinet-getitem?mr=1141740.

[16] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images. *IEEE Trans. Pattern Anal. Machine Intelligence*, 6: 721–741, 1984.

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

[18] W. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57:97–109, 1970.

[19] M. D. Hoffman and A. Gelman. The no-U-turn sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research*, 15: 1351–1381, 2014.

[20] C. Hwang, S. Hwang-Ma, and S. Sheu. Accelerating Gaussian diffusions. *The Annals of Applied Probability.*, 3:897 – 913, 1993.

[21] R. Johnson. Asymptotic expansions associated with posterior distributions. *Annals of Mathematical Statistics*, 43:851–864, 1970.

[22] S. C. Kapfer and W. Krauth. Cell-veto Monte Carlo algorithm for long-range systems. 2016. doi: https://arxiv.org/abs/1606.06780.

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

[24] J. Liu. *Monte Carlo strategies in scientific computing.* New York: Springer, 2001.

[25] D. MacLaurin and R. P. Adams. Firefly Monte Carlo: Exact MCMC with subsets of data. *Proceedings of the conference on Uncertainty in Artificial Intelligence (UAI).*, 2014.

[26] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equations of state calculation by fast computing machines. *J. Chem. Phys.*, 21:1087–1092, 1953.

[27] M. Michel, S. C. Kapfer, and W. Krauth. Generalized event-chain Monte Carlo: Constructing rejection-free global balance algorithms from infinitesimal steps. *J. Chem. Phys.*, 140(054116), 2014. doi: http://dx.doi.org/10.1063/1.4863991.

[28] M. Michel, J. Mayer, and W. Krauth. Event-chain Monte Carlo for classical continuous spin models. 2015. doi: http://www.arxiv.org/abs/1508.06541.

[29] R. Neal. Probabilistic inference using Markov Chain Monte Carlo methods. *Technical Report.*, 1993.

[30] R. Neal. *Bayesian Learning for Neural Networks.* New York: Springer, 1996.

[31] R. Neal. MCMC using Hamiltonian dynamics. In *Handbook of Markov Chain Monte Carlo.* Chapman & Hall/CRC, 2011.

[32] Y. Nishikawa, M. Michel, W. Krauth, and K. Hukushima. Event-chain algorithms for the Heisenberg model: Evidence for $z \approx 1$ dynamic scaling. *Phys. Rev. E.*, 112, 2015.

[33] E. A. J. F. Peters and G. de With. Rejection free Monte Carlo sampling for general potentials. *Physical Review E*, 85(026703), 2012. doi: http://dx.doi.org/10.1103/PhysRevE.85.026703.

[34] C. R. Rao. Information and accuracy attainable in the estimation of statistical parameters. *Bull. Calc. Math. Soc.*, 37:81–91, 1945.

[35] G. Roberts and J. Rosenthal. Examples of adaptive MCMC. *Techical Report, University of Toronto*, 2006.

[36] J. Shao. *Mathematical Statistics.* Spriger-Verlag, New York., 2nd edition, 2003.

[37] R. Shariff, A. György, and C. Szepesvári. Exploiting symmetries to construct efficient MCMC algorithms with an application to SLAM. *AISTATS*, 38, 2015.

[38] K. S. Turitsyn, M. Chertkov, and M. Vucelja. Irreversible Monte Carlo algorithms for efficient sampling. *Physica D.*

[39] A. van der Vaart. *Asymptotic Statistics.* Cambridge University Press, 1998.

[40] Z. Wang, S. Mohamed, and N. D. Freitas. Adaptive Hamiltonian and Riemann manifold Monte Carlo. *Proceedings of the 30th International Conference on Machine Learning*, pages 1462–1470, 2013.

8 Appendix A: Expectations and ESS.

8.1 On Estimating Expectations and the Effective Sample Size.

The main objective of performing MCMC is the calculation of expectations of arbitrary functions with respect to the target distribution of interest. In our case the target distribution will be the marginal of the position variables x - $\pi(dx)$ - and so for a given ECMC trajectory $\Xi(t) = (X(t), V(t))$ on $[0, T]$ and function $\varphi : \mathbb{R}^d \to \mathbb{R}$, the expectation that we wish to evaluate can be expressed as

$$\pi(\varphi) = \mathbb{E}_\pi[\varphi] = \int_{\mathbb{R}^d} \varphi(x)\, \pi(dx), \tag{66}$$

and by the results in Theorems 2.1 and 2.2, we may estimate this using

$$\widehat{\pi(\varphi)} = \frac{1}{T} \int_0^T \varphi(x(t))\, dt. \tag{67}$$

Given a 'skeleton' of n points consisting of the event times and the corresponding positions and velocities $\left\{t^{(i)}, X^{(i)}, V^{(i)}\right\}_{i \geq 0}^{n}$, i.e. the output of the ECMC algorithms, the path integral (67) may be expressed as the sum of integrals along straight line segments

$$\frac{1}{T} \sum_{i=1}^{n-1} \int_0^{\tau^{(i)}} \varphi\left(x^{(i-1)} + t v^{(i-1)}\right) dt \tag{68}$$

where $\tau^{(i)} = T^{(i)} - T^{(i-1)}$. In many cases, such as when estimating the moments of a component of x (i.e. $\varphi : \mathbb{R}^d \to \mathbb{R}$, $x \mapsto x_i^\alpha$ for $\alpha \in \mathbb{R}$), these integrals will be available in closed form. When this is not the case, there are two options. The first is to approximate the univariate integrals in (68) using numerical methods, e.g. quadrature. The alternative is to approximate using an evenly spaced grid of time points, i.e. set

$$\widehat{\pi(\varphi)} = \frac{1}{L} \sum_{l=0}^{L-1} \varphi(l\Delta), \tag{69}$$

where $\Delta > 0$ is the width of the time intervals and $L = 1 + \lfloor T/\Delta \rfloor$ ([8]). Letting $P_t((x, v), \cdot)$ denote the continuous time Markov kernel of the ECMC algorithm, we remark ([7]) that (69) effectively corresponds to a Monte Carlo estimate of the expectation with respect to the discrete time Markov chain with transition kernel $\widetilde{P}((x, v), \cdot) = P_\Delta((x, v), \cdot)$. We echo Bierkens et al. [7] in emphasizing that (69) is no longer a Monte Carlo estimate if the grid size is not uniform. In particular, it is invalid to simply use the event times and positions as Monte Carlo samples - as pointed out in [7], these points are of course heavily biased towards the tails of the distribution, where flipping/reflection events become more likely.

These two approximation methods indicate two corresponding methods of estimating the effective sample size of a trajectory. If one uses a discretely subsampled set of N

points as Monte Carlo samples, then one may of course simply estimate the ESS as

$$N_{eff} = \frac{N}{\left(1 + 2\sum\limits_{k=1}^{\infty} \rho_k\right)}, \tag{70}$$

and use traditional methods to estimate $(1 + 2\sum_{k=1}^{\infty} \rho_k)$, the integrated autocorrelation time (IACT). When the integral (67) is analytically tractable, it is convenient to estimate the ESS using the following method, which is detailed in Bierkens et al. [7] - we closely follow their exposition below.

Suppose that the central limit theorem holds for continuous trajectory $\{\varphi(x(t))\}_{t\geq 0}$, i.e. that for $t \to \infty$ we have

$$\frac{1}{\sqrt{t}} \int_0^t \{\varphi(x(s)) - \pi(\varphi)\}\, ds \;\to_{\mathcal{D}}\; \text{Normal}(0, \sigma_\varphi^2), \tag{71}$$

where the convergence is in distribution, and where σ_φ^2 denotes the asymptotic variance. The quantity σ_φ^2 can be estimated by dividing an observed trajectory $\{\varphi(x(t))\}_{0\leq t\leq \tau}$ into B batches of length τ/B as follows: for sufficiently large batch length the quantity

$$Y_b = \sqrt{\frac{B}{\tau}} \int_{(b-1)\tau/B}^{b\tau/B} \varphi(x(s))\, ds \tag{72}$$

for $b = 1, \ldots, B$ is approximately distributed as $\mathrm{N}(\sqrt{\tau/B}\pi(\varphi), \sigma_\varphi^2)$. Assuming further that the Y_b's are approximately independent, which is not unreasonable if the batch lengths are large, then the estimate

$$\widehat{\sigma_\varphi^2} = \frac{1}{B-1} \sum_{b=1}^{B} \left(Y_b - \widehat{Y}\right)^2 \tag{73}$$

where $\widehat{Y} = \left(\sum_{b=1}^{B} Y_b\right)/B$ is consistent for σ_φ^2. Using the mean and variance estimates of φ

$$\widehat{\pi(\varphi)} = \frac{1}{\tau} \int_0^\tau \varphi(x(s))\, ds \tag{74}$$

$$\widehat{\mathrm{Var}_\pi(\varphi)} = \frac{1}{\tau} \int_0^\tau \varphi(x(s))^2\, ds - \left(\widehat{\pi(\varphi)}\right)^2, \tag{75}$$

we may estimate the effective sample size using

$$N_{eff} = \frac{\tau \widehat{\mathrm{Var}_\pi(\varphi)}}{\widehat{\sigma_\varphi^2}}. \tag{76}$$

9 Appendix B: Python Code.

```
1
2
3 ### Flip algorithm for Gaussian distributions
4 ### assumes mean is zero
5 ### requires dimenson 'd', inverse of covariance
6 ### matrix 'Zinv', and the gamma parameter 'gamma'
7
8
9 ### initialize the position
10 x_0 = np.array([np.random.standard_normal((d,))])
11 ## initialize the 'velocity'
12 v_0 = (2*np.floor(2*np.random.random_sample(d)) - 1)*speeds
13
14 ## keep track of event times
15 T = 0
16 Time1 = np.array(np.zeros((Niter +1,)))
17
18 ##initialize all
19 X = np.array(np.zeros((Niter+1,d)))
20 V = np.array(np.zeros((Niter+1,d)))
21 X[0,:] = x_0
22 V[0,:] = v_0
23 x = x_0
24 v = v_0
25
26 ##if desired, keep track of event types
27 forceFlip = 0
28 trueFlip = 0
29
30 startTime = time.time()
31
32 for i in range(1,Niter+1):
33
34     tauList = np.array(np.zeros((d,)))
35     tau2List = np.array(np.random.exponential(1/gamma, d))
36     for j in range(0, d):
37
38         discrim = max(0,v[j]*np.dot(x,Zinv[j,:]))**2 -2*v[j]*np.dot(v,Zinv[j
    ,:])*math.log(np.random.rand(1))
39         if discrim > 0:
40
41             tauList[j] = (-np.dot(x,Zinv[j,:])/np.dot(v,Zinv[j,:])
42             + 1/(v[j]*np.dot(v,Zinv[j,:]))*math.sqrt(discrim))
43         else:
44             tauList[j] = tau2List[j]
45     tau1 = min(tauList)
46     tau2 = min(tau2List)
47     tauList = np.minimum(tauList,tau2List)
48     if tau1 < tau2:
49         trueFlip = trueFlip + 1
```

```
    else:
        forceFlip = forceFlip + 1
    tau = min(tau1,tau2)
    x = x + tau*v
    v = v*(1 - 2*(tauList == min(tauList)))

    X[i,:] = x
    V[i,:] = v
    Time1[i] = Time1[i-1] + tau

### compute the total computation time
timeFlip = time.time() - startTime

### total 'time'
t = Time1[-1]

#### estimate first two moments from entire chain
Lag_time = Time1[1:Niter +1] - Time1[0:Niter]

firstMoment = ((1/t)*(np.dot(Lag_time,X[0:Niter,:])
+ (1/2)*np.dot(Lag_time**2,V[0:Niter])))

secondMoment = ((1/t) * (np.dot(Lag_time,X[0:Niter,:]**2)
+ np.dot(Lag_time**2,X[0:Niter,:]*V[0:Niter])
+ (1/3)*np.dot(Lag_time**3,V[0:Niter]**2)))

mu_hat1 = firstMoment
sigSq_hat1 = secondMoment - firstMoment**2


### Reflection algorithm for Gaussian distributions
### shown is the version that uses a non-diagonal mass matrix
### requires refreshment parameter 'Lref', inverse of
### target covariance matrix 'Zinv', mass matrix 'M', and Cholesky
### decomposition of M 'rootM'.


### initialization
x_0 = np.array([np.random.standard_normal((d,))])
v_0 = np.dot(rootM,np.random.standard_normal((d,)))
T = 0
Time = np.array(np.zeros((Niter +1,)))
X = np.zeros((Niter +1,d))
V = np.zeros((Niter +1,d))
X[0,:] = x_0
V[0,:] = v_0
x = x_0
v = v_0



### keep track of event types
```

```
Refresh = 0
Bounce = 0
startTime = time.time()

for i in range(1,Niter+1):


    if np.dot(v,np.dot(Zinv,x.T)) >= 0:
        t1 = ((-np.dot(v,np.dot(Zinv,x.T))
        + math.sqrt(np.dot(v,np.dot(Zinv,x.T))**2
        - 2*np.dot(v,np.dot(Zinv,v.T))*math.log(np.random.rand(1))))
        / np.dot(v,np.dot(Zinv,v.T)))
    else:
        t1 = ((-np.dot(v,np.dot(Zinv,x.T))
        + math.sqrt(-2*np.dot(v,np.dot(Zinv,v.T))
        *math.log(np.random.rand(1))))
        / np.dot(v,np.dot(Zinv,v.T)))

    t2 = np.random.exponential(1/Lref)

    t = min(t1,t2)
    x = x + t*v

    if t1 <= t2:
        Bounce = Bounce + 1
        gradU = np.dot(x,Zinv)
        v = v - 2*np.dot(gradU,v.T)*np.dot(gradU,Z.T)/np.linalg.norm(np.dot(
    rootM.T,gradU.T))**2

    else:
        Refresh = Refresh + 1
        v = np.dot(rootM,np.random.standard_normal((d,)))

    X[i,:] = x
    V[i,:] = v
    Time[i] = Time[i-1] + t


### running time
timeRefl = time.time() - startTime

### total 'time'
t = Time[-1]

### compute all the moments
Lag_time = Time[1:Niter +1] - Time[0:Niter]
firstMoment = ((1/t)*(np.dot(Lag_time,X[0:Niter,:])
+ (1/2)*np.dot(Lag_time**2,V[0:Niter])))
secondMoment = ((1/t) * (np.dot(Lag_time,X[0:Niter,:]**2)
+ np.dot(Lag_time**2,X[0:Niter,:]*V[0:Niter])
+ (1/3)*np.dot(Lag_time**3,V[0:Niter]**2)))

```

```
mu_hat1 = firstMoment
sigSq_hat1 = secondMoment - firstMoment**2


### algorithms for logistic regression


import numpy as np
import numpy.random as npr
import math
import pandas as pd
import time
import matplotlib.pyplot as plt
import statsmodels.api as sm
from matplotlib.patches import Polygon


### simulate from a Poisson process with intensity a + bt
def affinePois(a,b):
    return((1/b)*(-a + math.sqrt(a**2 - 2*b*math.log(np.random.rand(1)))))


def logisticFun(a):
  return math.exp(a)/(1 + math.exp(a))
### gradient function (obs J)
def gradU(x,y,J,iota):
    return(iota[J,:]*(logisticFun((iota[J,:]*x).sum()) - y[J]))

### This code belongs to Ryan Adams, and can be found at
##  https://hips.seas.harvard.edu/blog/2013/03/03/
##  the-alias-method-efficient-sampling-with-many-discrete-outcomes/
def alias_setup(probs):
    K       = len(probs)
    q       = np.zeros(K)
    J       = np.zeros(K, dtype=np.int)

    # Sort the data into the outcomes with probabilities
    # that are larger and smaller than 1/K.
    smaller = []
    larger  = []
    for kk, prob in enumerate(probs):
        q[kk] = K*prob
        if q[kk] < 1.0:
            smaller.append(kk)
        else:
            larger.append(kk)

    # Loop though and create little binary mixtures that
    # appropriately allocate the larger outcomes over the
    # overall uniform mixture.
    while len(smaller) > 0 and len(larger) > 0:
        small = smaller.pop()
```

```
205         large = larger.pop()
206 
207         J[small] = large
208         q[large] = q[large] - (1.0 - q[small])
209 
210         if q[large] < 1.0:
211             smaller.append(large)
212         else:
213             larger.append(large)
214 
215     return J, q
216 
217 def alias_draw(J, q):
218     K  = len(J)
219 
220     # Draw from the overall uniform mixture.
221     kk = int(np.floor(npr.rand()*K))
222 
223     # Draw from the binary mixture, either keeping the
224     # small one, or choosing the associated larger one.
225     if npr.rand() < q[kk]:
226         return kk
227     else:
228         return J[kk]
229 
230 
231 ### flip method, naive sub-sampling
232 ### parameters are as follows:
233 ### R - num obs, d - dimension, Niter - num iterations
234 ### y - observed data, iota - design matrix
235 ### x_star - mle, gamma - refreshment parameter
236 ### parameters are the same for the other methods.
237 
238 def cycleZZnaive(R,d,Niter,y,iota,x_star,gamma):
239     gammas = np.ones((d,))*gamma
240 
241     bounds = np.max(iota, axis = 0)
242 
243     x_0 = x_star
244     v_0 = 2*np.floor(2*np.random.random_sample(d)) - 1
245 
246     Time = np.array(np.zeros((Niter +1,)))
247 
248     X = np.array(np.zeros((Niter+1,d)))
249     V = np.array(np.zeros((Niter+1,d)))
250     X[0,:] = x_0
251     V[0,:] = v_0
252     x = x_0
253     v = v_0
254 
255     startTime = time.time()
256 
```

```
257     for i in range(1, Niter + 1):
258 
259         tauList = np.zeros((d,))
260         for j in range(0,d):
261             tauList[j] = np.random.exponential
262             (1/(R*bounds[j] + gammas[j]))
263 
264         j_0 = int(tauList.argmin())
265         tau = tauList[j_0]
266         x = x + tau*v
267         Time[i] = Time[i-1] + tau
268         ### naive subsampling
269         if np.random.random_sample(1) < (R*bounds[j_0])
270         /((R*bounds[j_0] + gammas[j_0])):
271             k = int(np.floor(np.random.rand(1)*R))
272 
273             if np.random.rand(1) < (max(0,v[j_0]*
274             (gradU(x,y,k,iota)[j_0])))/(bounds[j_0]):
275                 v = v*(1 - 2*(tauList == min(tauList)))
276         else:
277             v[j_0] = -v[j_0]
278 
279         X[i,:] = x
280         V[i,:] = v
281 
282     tZZ = time.time() - startTime
283 ### the rest is contained in each function call, but is
284 ###shown only in this example
285 
286     t = Time[-1]
287 
288     ## compute moments
289     Lag_time = Time[1:Niter +1] - Time[0:Niter]
290 
291     firstMoment = ((1/t)*(np.dot(Lag_time,X[0:Niter,:])
292     + (1/2)*np.dot(Lag_time**2,V[0:Niter])))
293 
294     secondMoment = ((1/t) * (np.dot(Lag_time,X[0:Niter,:]**2)
295     + np.dot(Lag_time**2,X[0:Niter,:]*V[0:Niter])
296     + (1/3)*np.dot(Lag_time**3,V[0:Niter]**2)))
297 
298 
299     mu_hat = firstMoment
300     sigSq_hat = secondMoment - firstMoment**2
301 
302     realVar = sigSq_hat
303 
304     ### set number of batches
305     B = 200
306     batchTime = t/B
307 
308 
```

```
309
310     batchIndices = np.zeros((B+1,))
311     index = 0
312     for i in range(1,B+1):
313         index = index + np.array((Time[index:]
314          <= batchTime*i).nonzero()).max()
315         batchIndices[i] = index
316
317
318     ## compute the mean in each batch
319
320     batchMeans = np.zeros((B,d))
321
322
323     if (batchIndices[B] == Niter - 1):
324         for j in range(0,B):
325             firstMoment = ((B/t)*(np.dot(Lag_time
326             [int(batchIndices[j]):int(batchIndices[j+1])+1],
327             X[int(batchIndices[j]):int(batchIndices[j+1])+1,:])
328             + (1/2)*np.dot(Lag_time[int(batchIndices[j])
329             :int(batchIndices[j+1])+1]**2,V[int(batchIndices[j])
330             :int(batchIndices[j+1])+1])))
331             mu_hat = firstMoment
332             batchMeans[j,:] = mu_hat
333
334     else:
335         for j in range(0,B):
336
337             if j < B-1:
338
339                 firstMoment = ((B/t)*(np.dot(Lag_time
340                 [int(batchIndices[j]):int(batchIndices[j+1])+1]
341                 ,X[int(batchIndices[j]):int(batchIndices[j+1])+1,:])
342                 + (1/2)*np.dot(Lag_time[int(batchIndices[j])
343                 :int(batchIndices[j+1])+1]**2,V[int(batchIndices[j])
344                 :int(batchIndices[j+1])+1])))
345             else:
346                 firstMoment = ((B/t)*(np.dot
347                 (Lag_time[int(batchIndices[j]):int(batchIndices[j+1])+1]
348                 ,X[int(batchIndices[j]):int(batchIndices[j+1]),:])
349                 + (1/2)*np.dot(Lag_time[int(batchIndices[j])
350                 :int(batchIndices[j+1])+1]**2,V[int(batchIndices[j])
351                 :int(batchIndices[j+1])])))
352
353
354     Yvec = math.sqrt(B/t)*batchMeans
355     Ybar = np.mean(Yvec, axis = 0)
356     Y = (Yvec - Ybar)**2
357
358     sigHat = (1/(B-1))*np.sum(Y, axis = 0)
359
360
```

```
361     sampleSizes = t*(realVar/sigHat)
362     ESSs = np.mean(sampleSizes)/tZZ
363 
364     return (ESSs,np.mean(sampleSizes),tZZ)
365 
366 ### flip method, informed sub-sampling
367 def cycleZZalias(R,d,Niter,y,iota,x_star,gamma):
368     gammas = np.ones((d,))*gamma
369 
370     data = np.vstack((y,iota.T)).T
371 
372     c0 = data[:,0] == 0
373     c1 = data[:,0] == 1
374 
375     iota0 = data[c0,1:(d+1)].sum(axis = 0)
376     iota1 = data[c1,1:(d+1)].sum(axis = 0)
377 
378     aliasVector0 = (iota/iota0)*(1-y.reshape(R,1))
379     aliasVector1 = (iota/iota1)*y.reshape(R,1)
380 
381 
382     J_zero = np.zeros([d,R])
383     Q_zero = np.zeros([d,R])
384 
385     J_one = np.zeros([d,R])
386     Q_one = np.zeros([d,R])
387 
388     for ii in range(d):
389         J_zero[ii,:], Q_zero[ii,:] = alias_setup(aliasVector0[:,ii])
390         J_one[ii,:], Q_one[ii,:] = alias_setup(aliasVector1[:,ii])
391 
392     x_0 = x_star
393     v_0 = 2*np.floor(2*np.random.random_sample(d)) - 1
394 
395     Time = np.array(np.zeros((Niter +1,)))
396 
397     X = np.array(np.zeros((Niter+1,d)))
398     V = np.array(np.zeros((Niter+1,d)))
399     X[0,:] = x_0
400     V[0,:] = v_0
401     x = x_0
402     v = v_0
403 
404     startTime = time.time()
405     for i in range(1,Niter+1):
406 
407 
408         chi = np.array(np.zeros([d]))
409         for ii in range(0,d):
410             if v[ii] < 0:
411                 chi[ii] = np.abs(v[ii])*iota1[ii]
412 
```

```
413             else:
414                 chi[ii] = np.abs(v[ii])*iota0[ii]
415 
416         tauList1 = np.zeros((d,))
417         tauList2 = np.zeros((d,))
418         for ii in range(0,d):
419             tauList1[ii] = np.random.exponential(1/chi[ii])
420             tauList2[ii] = np.random.exponential(1/gammas[ii])
421         j_0 = int(tauList1.argmin())
422         j_1 = int(tauList2.argmin())
423 
424         tau = min(tauList1[j_0],tauList2[j_1])
425         x = x + tau*v
426         Time[i] = Time[i-1] + tau
427 
428         if tau == tauList2[j_1]:
429             v[j_1] = -v[j_1]
430 
431         else:
432             if v[j_0] < 0 :
433                 r = int(alias_draw(J_one[j_0,:],Q_one[j_0,:]))
434             else:
435                 r = int(alias_draw(J_zero[j_0,:], Q_zero[j_0,:]))
436 
437         if np.random.random_sample(1) < max(0,v[j_0]*gradU(x,y,r,iota)[j_0])
    /(abs(v[j_0])*iota[r,j_0]):
438             v[j_0] = -v[j_0]
439 
440 
441         X[i,:] = x
442         V[i,:] = v
443 
444     tZZ = time.time() - startTime
445 
446     ### here goes the rest
447 
448     return (ESSs,np.mean(sampleSizes),tZZ)
449 
450 
451 ### flip method, control variates
452 def cycleZZcv(R,d,Niter,y,iota,x_star,gamma):
453     gammas = np.ones((d,))*gamma
454 
455     lipKs = np.zeros((d,))
456     dataNorms = np.linalg.norm(iota, axis =1)
457 
458     boundMat = (dataNorms*iota.T).T
459 
460     for i in range(0,d):
461         lipKs[i] = R*(1/4)*max(boundMat[:,i])
462 
463     gradRefs = np.array(np.zeros((R,d)))
```

```
464
465     for i in range(0,R):
466         gradRefs[i,:] = gradU(x_star,y,i,iota)
467
468     refGrad = np.sum(gradRefs, axis = 0)
469     #x_0 = x_star
470     x_0 = x_star
471     v_0 = 2*np.floor(2*np.random.random_sample(d)) - 1
472
473     Time = np.array(np.zeros((Niter +1,)))
474
475     X = np.array(np.zeros((Niter+1,d)))
476     V = np.array(np.zeros((Niter+1,d)))
477     X[0,:] = x_0
478     V[0,:] = v_0
479     x = x_0
480     v = v_0
481
482     startTime = time.time()
483     for i in range(1, Niter + 1):
484
485         A = (v*refGrad)*(v*refGrad > 0) + np.linalg.norm(x - x_star)*lipKs
486         B = math.sqrt(d)*lipKs
487         tauList = np.zeros((d,))
488         tauList2 = np.zeros((d,))
489         for j in range(0,d):
490             tauList[j] = affinePois(A[j],B[j])
491             tauList2[j] = np.random.exponential(1/gammas[j])
492
493         j_0 = int(tauList.argmin())
494         j_1 = int(tauList2.argmin())
495
496         tau = min(tauList[j_0],tauList2[j_1])
497
498
499         x = x + tau*v
500         Time[i] = Time[i-1] + tau
501         ### naive subsampling
502
503         if tau == tauList[j_0]:
504             k = int(np.floor(np.random.rand(1)*R))
505
506             if np.random.rand(1) < (R*max(0,v[j_0]*(refGrad[j_0]/R
507             + gradU(x,y,k,iota)[j_0] - gradRefs[k,j_0])))/(A[j_0] + tau*B[
    j_0]):
508
509                 v[j_0] = -v[j_0]
510         else:
511             v[j_1] = -v[j_1]
512
513         X[i,:] = x
514         V[i,:] = v
```

```
515
516     tZZ = time.time() - startTime
517
518     return (ESSs,np.mean(sampleSizes),tZZ)
519
520
521 ### flip method, informed sub-sampling, control variates
522 def cycleZZalcv(R,d,Niter,y,iota,gamma,x_star):
523
524     gammas = np.ones((d,))*gamma
525
526     dataNorms = np.linalg.norm(iota, axis =1)
527
528     C_Mat = (1/4)*(dataNorms*iota.T).T
529
530     C_sums = np.sum(C_Mat, axis = 0)
531
532     C_probvec = C_Mat/C_sums
533
534     gradRefs = np.array(np.zeros((R,d)))
535
536     for i in range(0,R):
537         gradRefs[i,:] = gradU(x_star,y,i,iota)
538
539
540     J_zero = np.zeros([d,R])
541     Q_zero = np.zeros([d,R])
542
543
544     for ii in range(0,d):
545         J_zero[ii,:], Q_zero[ii,:] = alias_setup(C_probvec[:,ii])
546
547
548     x_0 = x_star
549     v_0 = 2*np.floor(2*np.random.random_sample(d)) - 1
550
551     Time = np.array(np.zeros((Niter +1,)))
552
553     X = np.array(np.zeros((Niter+1,d)))
554     V = np.array(np.zeros((Niter+1,d)))
555     X[0,:] = x_0
556     V[0,:] = v_0
557     x = x_0
558     v = v_0
559
560     startTime = time.time()
561     for i in range(1,Niter+1):
562
563         nx = np.linalg.norm(x-x_star)
564         A = C_sums*nx
565         B = C_sums*math.sqrt(d)
566
```

```
567         tauList1 = np.zeros((d,))
568         tauList2 = np.zeros((d,))
569         for ii in range(0,d):
570             tauList1[ii] = affinePois(A[ii],B[ii])
571             tauList2[ii] = np.random.exponential(1/gammas[ii])
572         j_0 = int(tauList1.argmin())
573         j_1 = int(tauList2.argmin())
574 
575         tau = min(tauList1[j_0],tauList2[j_1])
576         x = x + tau*v
577         Time[i] = Time[i-1] + tau
578 
579         if tau == tauList2[j_1]:
580             v[j_1] = -v[j_1]
581 
582         else:
583             r = int(alias_draw(J_zero[j_0,:], Q_zero[j_0,:]))
584 
585         if np.random.random_sample(1) < max(0,v[j_0]*(gradU(x,y,r,iota)[j_0]
    - gradRefs[r,j_0]))/(nx*C_Mat[r,j_0] + tau*math.sqrt(d)*C_Mat[r,j_0]):
586             v[j_0] = -v[j_0]
587 
588 
589         X[i,:] = x
590         V[i,:] = v
591 
592     tZZ = time.time() - startTime
593 
594 
595     return (ESSs,np.mean(sampleSizes),tZZ)
596 
597 
598 #### reflection, naive subsampling
599 def cycleBPSnaive(R,d,Niter,y,iota,x_star,Lref):
600 
601     bounds = np.max(iota, axis = 0)
602 
603     x_0 = x_star
604     v_0 = np.array(np.random.standard_normal(d,))
605     Time = np.array(np.zeros((Niter +1,)))
606     X = np.array(np.zeros((Niter+1,d)))
607     V = np.array(np.zeros((Niter+1,d)))
608     X[0,:] = x_0
609     V[0,:] = v_0
610     x = x_0
611     v = v_0
612 
613     startTime = time.time()
614     for i in range(1, Niter + 1):
615 
616         tau = np.random.exponential(1/(R*np.dot(abs(v),bounds)))
617         tau2 = np.random.exponential(1/Lref)
```

```
618         tau = min(tau,tau2)
619         x = x + tau*v
620         Time[i] = Time[i-1] + tau
621         if tau == tau2:
622             v = np.array(np.random.standard_normal(d,))
623         else:
624             k = int(np.floor(np.random.rand(1)*R))
625             gU = iota[k,:]*(logisticFun((iota[k,:]*x).sum()) - y[k])
626             if np.random.rand(1) < max(0,np.dot(v,gU))/(np.dot(abs(v),bounds
    ))):
627                 v = v - 2*np.dot(gU,v)/(np.linalg.norm(gU)**2)*gU
628
629         X[i,:] = x
630         V[i,:] = v
631
632     tBPS = time.time() - startTime
633
634     return (ESSs,np.mean(sampleSizes),tBPS)
635
636
637 ### reflection, informed subsampling
638 def cycleBPSalias(R,d,Niter,y,iota,x_star,Lref):
639
640     data = np.vstack((y,iota.T)).T
641
642     c0 = data[:,0] == 0
643     c1 = data[:,0] == 1
644
645     iota0 = data[c0,1:(d+1)].sum(axis = 0)
646     iota1 = data[c1,1:(d+1)].sum(axis = 0)
647
648     aliasVector0 = (iota/iota0)*(1-y.reshape(R,1))
649     aliasVector1 = (iota/iota1)*y.reshape(R,1)
650
651
652     J_zero = np.zeros([d,R])
653     Q_zero = np.zeros([d,R])
654
655     J_one = np.zeros([d,R])
656     Q_one = np.zeros([d,R])
657
658     for ii in range(d):
659         J_zero[ii,:], Q_zero[ii,:] = alias_setup(aliasVector0[:,ii])
660         J_one[ii,:], Q_one[ii,:] = alias_setup(aliasVector1[:,ii])
661
662     x_0 = x_star
663     v_0 = np.random.standard_normal((d,))
664
665     Time = np.array(np.zeros((Niter +1,)))
666
667     X = np.array(np.zeros((Niter+1,d)))
668     V = np.array(np.zeros((Niter+1,d)))
```

```
669     X[0,:] = x_0
670     V[0,:] = v_0
671     x = x_0
672     v = v_0
673
674
675     startTime = time.time()
676
677     for i in range(1,Niter+1):
678
679         chi = 0
680         q = np.array(np.zeros([d]))
681         for ii in range(0,d):
682             if v[ii] < 0:
683                 chi = chi + np.abs(v[ii])*iota1[ii]
684                 q[ii] = np.abs(v[ii])*iota1[ii]
685             else:
686                 chi = chi + np.abs(v[ii])*iota0[ii]
687                 q[ii] = np.abs(v[ii])*iota0[ii]
688         q = q/q.sum()
689
690
691         tau = np.random.exponential(1/(Lref + chi))
692
693         x = x + tau*v
694         Time[i] = Time[i-1] + tau
695         u = np.random.random_sample(1)
696         if u < chi/(chi + Lref): ## ii) if j = 1
697         ## draw k from q(k)
698               k = int(np.random.multinomial(1, q, size=1).ravel().nonzero()
    [0])
699               ## draw r from q(r|k)
700               if v[k] < 0 :
701                   r = int(alias_draw(J_one[k,:],Q_one[k,:]))
702               else:
703                   r = int(alias_draw(J_zero[k,:], Q_zero[k,:]))
704
705               gradU = (logisticFun((iota[r,:]*x).sum()) - y[r])*iota[r,:]
706               chi_r = np.dot((v*((-1)**y[r]) >= 0), (iota[r,:]*abs(v)).T)
707
708               if np.random.random_sample(1) < max(0,np.dot(gradU,v))/chi_r :
709
710                   v = v - 2*(gradU*v).sum()/(np.linalg.norm(gradU)**2)*gradU #
    # Else
711
712         else:
713             v = np.random.standard_normal((d,))
714
715         X[i,:] = x
716         V[i,:] = v
717
718
```

```
719     tBPS = time.time() - startTime
720 
721     t = Time[-1]
722 
723     return (ESSs,np.mean(sampleSizes),tBPS)
724 
725 
726 ### reflection method, control variates
727 def cycleBPScv(R,d,Niter,y,iota,Lref,x_star):
728 
729     lipKs = np.zeros((d,))
730     dataNorms = np.linalg.norm(iota, axis =1)
731 
732     boundMat = (dataNorms*iota.T).T
733 
734     for i in range(0,d):
735         lipKs[i] = R*(1/4)*max(boundMat[:,i])
736 
737     gradRefs = np.array(np.zeros((R,d)))
738 
739     for i in range(0,R):
740         gradRefs[i,:] = gradU(x_star,y,i,iota)
741 
742     refGrad = np.sum(gradRefs, axis = 0 )
743 
744     x_0 = x_star
745     v_0 = np.array(np.random.standard_normal(d,))
746     Time = np.array(np.zeros((Niter +1,)))
747     X = np.array(np.zeros((Niter+1,d)))
748     V = np.array(np.zeros((Niter+1,d)))
749     X[0,:] = x_0
750     V[0,:] = v_0
751     x = x_0
752     v = v_0
753 
754     startTime = time.time()
755     for i in range(1, Niter + 1):
756 
757         A = max(0,np.dot(v,refGrad)) + np.linalg.norm(x - x_star)*np.dot(
    lipKs,np.abs(v))
758         B = np.linalg.norm(v)*np.dot(lipKs,np.abs(v))
759         tau1 = affinePois(A,B)
760         tau2 = np.random.exponential(1/Lref)
761         tau = min(tau1,tau2)
762         x = x + tau*v
763         Time[i] = Time[i-1] + tau
764 
765         if tau2 < tau1:
766             v = np.array(np.random.standard_normal(d,))
767         ### naive subsampling
768         else:
769             k = int(np.floor(np.random.rand(1)*R))
```

```
770             Ek = (refGrad/R + gradU(x,y,k,iota) - gradRefs[k,:])
771             if np.random.rand(1) < R*max(0,np.dot(v,Ek))/(A + tau*B):
772                 v = v - 2*np.dot(Ek,v)/(np.linalg.norm(Ek)**2)*Ek
773 
774         X[i,:] = x
775         V[i,:] = v
776 
777     tBPS = time.time() - startTime
778     t = Time[-1]
779 
780 
781     return (ESSs,np.mean(sampleSizes),tBPS)
782 
783 
784 ### reflection with control variates - informed ss
785 def cycleBPSalcv(R,d,Niter,y,iota,Lref,x_star):
786 
787     dataNorms = np.linalg.norm(iota, axis =1)
788 
789     C_Mat = (1/4)*(dataNorms*iota.T).T
790 
791     C_sums = np.sum(C_Mat, axis = 0)
792 
793     C_probvec = C_Mat/C_sums
794 
795     gradRefs = np.array(np.zeros((R,d)))
796 
797     for i in range(0,R):
798         gradRefs[i,:] = gradU(x_star,y,i,iota)
799 
800 
801     J_zero = np.zeros([d,R])
802     Q_zero = np.zeros([d,R])
803 
804 
805     for ii in range(0,d):
806         J_zero[ii,:], Q_zero[ii,:] = alias_setup(C_probvec[:,ii])
807 
808 
809     x_0 = x_star
810     v_0 = np.random.standard_normal((d,))
811 
812     Time = np.array(np.zeros((Niter +1,)))
813 
814     X = np.array(np.zeros((Niter+1,d)))
815     V = np.array(np.zeros((Niter+1,d)))
816     X[0,:] = x_0
817     V[0,:] = v_0
818     x = x_0
819     v = v_0
820 
821 
```

```
822     startTime = time.time()
823     for i in range(1,Niter+1):
824         nv = np.linalg.norm(v)
825         nx = np.linalg.norm(x - x_star)
826         A = np.dot(C_sums,abs(v))*nx
827         B = np.dot(C_sums,abs(v))*nv
828 
829         q = np.array(np.zeros([d]))
830         for ii in range(0,d):
831             q[ii] = abs(v[ii])*C_sums[ii]
832 
833         q = q/q.sum()
834 
835 
836         tau1 = affinePois(A,B)
837         tau2 = np.random.exponential(1/(Lref))
838 
839         tau = min(tau1,tau2)
840         x = x + tau*v
841         Time[i] = Time[i-1] + tau
842 
843         if tau == tau1: ## ii) if j = 1
844         ## draw k from q(k)
845              k = int(np.random.multinomial(1, q, size=1).ravel().nonzero()
    [0])
846              ## draw r from q(r|k)
847              r = int(alias_draw(J_zero[k,:], Q_zero[k,:]))
848 
849              Er = (gradU(x,y,r,iota) - gradRefs[r,:])
850              if np.random.random_sample(1) < max(0,np.dot(Er,v))/(np.dot(
    C_Mat[r,:],abs(v))*(tau*nv + nx)):
851                  v = v - 2*np.dot(Er,v)/(np.linalg.norm(Er)**2)*Er
852         else:
853             v = np.random.standard_normal((d,))
854 
855         X[i,:] = x
856         V[i,:] = v
857 
858 
859     tBPS = time.time() - startTime
860 
861 
862     return (ESSs,np.mean(sampleSizes),tBPS)
863 
864 
865 
866 
867 ### one example with a mass matrix will suffice
868 ### 'M' is the mass matrix, 'rootM' is the Cholesky decomposition
869 
870 
871 def cycleBPSalcvM(R,d,Niter,y,iota,Lref,x_star,M,rootM):
```

```
872
873     dataNorms = np.linalg.norm(iota, axis =1)
874
875     C_Mat = (1/4)*(dataNorms*np.abs(iota.T)).T
876
877     C_sums = np.sum(C_Mat, axis = 0)
878
879     C_probvec = C_Mat/C_sums
880
881     gradRefs = np.array(np.zeros((R,d)))
882
883     for i in range(0,R):
884         gradRefs[i,:] = gradU(x_star,y,i,iota)
885
886     J_zero = np.zeros([d,R])
887     Q_zero = np.zeros([d,R])
888
889
890
891
892
893
894     for ii in range(0,d):
895         J_zero[ii,:], Q_zero[ii,:] = alias_setup(C_probvec[:,ii])
896
897
898     x_0 = x_star
899     v_0 = np.dot(rootM,np.random.standard_normal((d,)).T)
900
901     Time = np.array(np.zeros((Niter +1,)))
902
903     X = np.array(np.zeros((Niter+1,d)))
904     V = np.array(np.zeros((Niter+1,d)))
905     X[0,:] = x_0
906     V[0,:] = v_0
907     x = x_0
908     v = v_0
909
910
911     startTime = time.time()
912     for i in range(1,Niter+1):
913         nv = np.linalg.norm(v)
914         nx = np.linalg.norm(x - x_star)
915         A = np.dot(C_sums,abs(v))*nx
916         B = np.dot(C_sums,abs(v))*nv
917
918         q = np.array(np.zeros([d]))
919         for ii in range(0,d):
920             q[ii] = abs(v[ii])*C_sums[ii]
921
922         q = q/q.sum()
923
```

```
924
925         tau1 = affinePois(A,B)
926         tau2 = np.random.exponential(1/(Lref))
927
928         tau = min(tau1,tau2)
929         x = x + tau*v
930         Time[i] = Time[i-1] + tau
931
932         if tau == tau1: ## ii) if j = 1
933         ## draw k from q(k)
934               k = int(np.random.multinomial(1, q, size=1).ravel().nonzero()
    [0])
935               ## draw r from q(r|k)
936               r = int(alias_draw(J_zero[k,:], Q_zero[k,:]))
937
938               Er = (gradU(x,y,r,iota) - gradRefs[r,:])
939               if np.random.random_sample(1) < max(0,np.dot(Er,v))/(np.dot(
    C_Mat[r,:],abs(v))*(tau*nv + nx)):
940                   v = v - 2*np.dot(Er,v)/(np.linalg.norm(np.dot(rootM.T,Er.T))
    **2)*np.dot(M,Er)
941           else:
942               v = np.dot(rootM,np.random.standard_normal((d,)).T)
943
944         X[i,:] = x
945         V[i,:] = v
946
947 ### code for random field example
948
949
950 import numpy as np
951 import time
952 import math
953
954 ##hyperparameters
955 d = 30
956 sig2 = 1.91
957 mu = np.log(126) - sig2/2
958 beta = 1/6
959 s = 1/d**2
960
961 Lref = 5
962
963
964 ## create the 900x900 cov matrix
965
966 Z = np.zeros((d**2,d**2))
967
968 startTime = time.time()
969 for n in range(0,d**2):
970     ni = np.ceil((n+1)/d)
971     nj = (n+1) % d
972     if nj == 0:
```

```
            nj = d
        for m in range(0,d**2):
            mi = np.ceil((m+1)/d)
            mj = (m+1) % d
            if mj == 0:
                mj = d

            Z[n,m] = np.sqrt((ni - mi)**2 + (nj - mj)**2)

matTime = time.time() - startTime


### Fisher inf.
Z = sig2*np.exp(-Z/(beta*d))
Zinv = np.linalg.inv(Z)
L = np.linalg.cholesky(Z)


### make the mass matrix

Lambda = np.zeros((d**2,d**2))
for i in range(0,d**2):
    Lambda[i,i] = s*math.exp(mu + Z[i,i])

G = Lambda + Zinv
M = np.linalg.inv(G)

for i in range(0,d**2):
    for j in range(0,d**2):
        if M[i,j] < 10**(-5):
            M[i,j] = 0


rootM = np.linalg.cholesky(M)

### generate the latent field X and data Y

X =  np.dot(L,np.random.standard_normal((d**2,)).T) + mu

Y = np.zeros((d**2,))
for i in range(0,d**2):
    Y[i] = np.random.poisson(s*np.exp(X[i]))

latProc = s*np.exp(X)



### batch means estimator for HMC ESS

def batchMeansNeff(X,N,B):  ## assumes N/B is an integer
    bMeans = np.zeros((B,))
    m = N/B ## batch size
```

```
    for i in range(0,B):
        bMeans[i] = np.sum(X[i*m:(i+1)*m])/m

    s = np.var(X,ddof = 1)
    s_batch = m*np.var(bMeans, ddof = 1)

    return(N*(s/s_batch))



### function to generate an arrival time from a Gaussian
### for simulation of event time of U_2(x) (see section 6.4)
def genGaussianTime(x,v,mu,Zinv):
    x = x - mu
    if np.dot(v,np.dot(Zinv,x.T)) >= 0:
        t = ((-np.dot(v,np.dot(Zinv,x.T))
        + math.sqrt(np.dot(v,np.dot(Zinv,x.T))**2
        - 2*np.dot(v,np.dot(Zinv,v.T))*math.log(np.random.rand(1))))
        / np.dot(v,np.dot(Zinv,v.T)))
    else:
        t = ((-np.dot(v,np.dot(Zinv,x.T))
        + math.sqrt(-2*np.dot(v,np.dot(Zinv,v.T))
        *math.log(np.random.rand(1))))
        / np.dot(v,np.dot(Zinv,v.T)))
    return(t)


### no mass (moment calculations, ess calculations left out
### as they are identical to the logistic case

def cycleRF(Y,Z,Zinv,mu,Niter,Lref):

    x_0 = np.random.standard_normal((d**2,))
    v_0 = np.random.standard_normal((d**2,))

    Time = np.array(np.zeros((Niter +1,)))

    X = np.array(np.zeros((Niter+1,d**2)))
    V = np.array(np.zeros((Niter+1,d**2)))
    X[0,:] = x_0
    V[0,:] = v_0
    x = x_0
    v = v_0

    startTime = time.time()

    for i in range(1,Niter+1):

        t1 = np.random.exponential(1/Lref)
        t2 = genGaussianTime(x,v,mu,Zinv)

        tauList = np.zeros((d**2,))
```

```
1077         for j in range(0,d**2):
1078 
1079             if v[j] > 0:
1080                 tauList[j] = (1/v[j])*(math.log(-math.log(np.random.
     random_sample(1))/s + math.exp(x[j])) - x[j])
1081             elif Y[j]>0:
1082                 tauList[j] = math.log(np.random.random_sample(1))/(Y[j]*v[j
     ])
1083             else:
1084                 tauList[j] = np.inf
1085 
1086         t3 = min(tauList)
1087         tau = min(t1,t2,t3)
1088         x = x + tau*v
1089         Time[i] = Time[i-1] + tau
1090 
1091         if tau == t1:
1092             v = np.random.standard_normal((d**2,))
1093 
1094         else:
1095             gradU = np.dot(Zinv,(x - mu)) - Y + s*np.exp(x)
1096             v = v - 2*(np.dot(gradU,v)/np.linalg.norm(gradU)**2)*gradU
1097 
1098         X[i,:] = x
1099         V[i,:] = v
1100 
1101     tBPS = time.time() - startTime
1102 
1103     return (ESSs,sampleSizes,tBPS,Min,Med,Max,means,Vars)
1104 
1105 
1106 
1107 def cycleRFwMass(Y,Z,Zinv,mu,Niter,Lref,M,rootM):
1108 
1109     x_0 = np.random.standard_normal((d**2,))
1110     v_0 = np.dot(rootM,np.random.standard_normal((d**2,)).T)
1111 
1112     Time = np.array(np.zeros((Niter +1,)))
1113 
1114     X = np.array(np.zeros((Niter+1,d**2)))
1115     V = np.array(np.zeros((Niter+1,d**2)))
1116     X[0,:] = x_0
1117     V[0,:] = v_0
1118     x = x_0
1119     v = v_0
1120 
1121     startTime = time.time()
1122 
1123     for i in range(1,Niter+1):
1124 
1125         t1 = np.random.exponential(1/Lref)
1126         t2 = genGaussianTime(x,v,mu,Zinv)
```

```
1127
1128         tauList = np.zeros((d**2,))
1129         for j in range(0,d**2):
1130
1131             if v[j] > 0:
1132                 tauList[j] = (1/v[j])*(math.log(-math.log(np.random.
    random_sample(1))/s + math.exp(x[j])) - x[j])
1133             elif Y[j] > 0:
1134                 tauList[j] = math.log(np.random.random_sample(1))/(Y[j]*v[j
    ])
1135             else: tauList[j] = np.inf
1136
1137         t3 = min(tauList)
1138         tau = min(t1,t2,t3)
1139         x = x + tau*v
1140         Time[i] = Time[i-1] + tau
1141
1142         if tau == t1:
1143             v = np.dot(rootM,np.random.standard_normal((d**2,)))
1144
1145         else:
1146             gradU = np.dot(Zinv,(x - mu)) - Y + s*np.exp(x)
1147             v = v - 2*(np.dot(gradU,v)/np.linalg.norm(np.dot(rootM.T,gradU.T
    ))**2)*np.dot(M,gradU.T)
1148
1149         X[i,:] = x
1150         V[i,:] = v
1151
1152     tBPS = time.time() - startTime
1153
1154
1155     return (ESSs,sampleSizes,tBPS,Min,Med,Max,means,Vars)
1156
1157 ### batch means for HMC ESS calculations
1158
1159 def batchMeans(X,N,B):  ## assumes N/B is an integer
1160     bMeans = np.zeros((B,))
1161     m = N/B ## batch size
1162     for i in range(0,B):
1163         bMeans[i] = np.sum(X[i*m:(i+1)*m])/m
1164
1165     s = np.var(X,ddof = 1)
1166     s_batch = m*np.var(bMeans, ddof = 1)
1167
1168     return(N*(s/s_batch))
1169
1170
1171
1172
1173
1174 ### HMC, RMHMC code, modified from pseudocode given in 'MCMC using
    Hamiltonian Dynamics' by Radford Neal
```

```
1175
1176 def HMCitM(U, grad_U, epsilon, L, current_q,d,Minv,rootM):
1177     q = current_q
1178     p = np.dot(rootM,np.random.standard_normal((d,)))
1179     # independent standard normal variates
1180     current_p = p
1181     # Make a half step for momentum at the beginning
1182     p = p- epsilon*grad_U(q)/2
1183     # Alternate full steps for position and momentum
1184     for i in range(1,L+1):
1185
1186
1187         q = q + epsilon*np.dot(Minv,p)
1188         # Make a full step for the momentum, except at end of trajectory
1189         if (i!=L):
1190             p = p - epsilon*grad_U(q)
1191
1192     # Make a half step for momentum at the end.
1193     p = p - epsilon*grad_U(q)/2
1194     # Negate momentum at end of trajectory to make the proposal symmetric
1195     p = -p
1196     # Evaluate potential and kinetic energies at start and end of trajectory
1197     current_U = U(current_q)
1198     current_K = np.dot(current_p,np.dot(Minv,current_p))/2  ## identity mass
      matrix
1199     proposed_U = U(q)
1200     proposed_K = np.dot(p,np.dot(Minv,p))/2
1201     # Accept or reject the state at end of trajectory, returning either
1202     # the position at the end of the trajectory or the initial position
1203     if (np.log(np.random.random_sample(1)) <
1204     (current_U-proposed_U+current_K-proposed_K)):
1205         return (q) # accept
1206
1207     else:
1208         return (current_q) # reject
1209
1210 def HMCit(U, grad_U, epsilon, L, current_q,d):
1211     q = current_q
1212     p = np.random.standard_normal((d,)) # independent standard normal
     variates
1213     current_p = p
1214     # Make a half step for momentum at the beginning
1215     p = p- epsilon*grad_U(q)/2
1216     # Alternate full steps for position and momentum
1217     for i in range(1,L+1):
1218
1219
1220         q = q + epsilon*p
1221         # Make a full step for the momentum, except at end of trajectory
1222         if (i!=L):
1223             p = p - epsilon*grad_U(q)
1224
```

```
    # Make a half step for momentum at the end.
    p = p - epsilon*grad_U(q)/2
    # Negate momentum at end of trajectory to make the proposal symmetric
    p = -p
    # Evaluate potential and kinetic energies at start and end of trajectory
    current_U = U(current_q)
    current_K = np.sum(current_p**2)/2  ## identity mass matrix
    proposed_U = U(q)
    proposed_K = np.sum(p**2)/2
    # Accept or reject the state at end of trajectory, returning either
    # the position at the end of the trajectory or the initial position
    if (np.log(np.random.random_sample(1)) <
    (current_U-proposed_U+current_K-proposed_K)):
        return (q) # accept

    else:
        return (current_q)



### functions that HMC, RMHMC will call:
### energy, gradient respectively

def U(x): #### negative log density
    u1 = np.dot((x - mu),np.dot(Zinv,(x-mu)))/2
    u2 = -np.sum(Y*x) + s*np.sum(np.exp(x))
    return u1 + u2

def grad_U(x):
    return -Y +s*np.exp(x) + np.dot(Zinv,(x-mu))
```