Statistics and Computing (2020) 30:1325–1345
https://doi.org/10.1007/s11222-020-09948-4

Markov chain Monte Carlo algorithms with sequential proposals

Joonha Park[1] · Yves Atchadé[1]

Received: 19 August 2019 / Accepted: 10 May 2020 / Published online: 24 June 2020

Abstract
We explore a general framework in Markov chain Monte Carlo (MCMC) sampling where sequential proposals are tried as a candidate for the next state of the Markov chain. This sequential-proposal framework can be applied to various existing MCMC methods, including Metropolis–Hastings algorithms using random proposals and methods that use deterministic proposals such as Hamiltonian Monte Carlo (HMC) or the bouncy particle sampler. Sequential-proposal MCMC methods construct the same Markov chains as those constructed by the delayed rejection method under certain circumstances. In the context of HMC, the sequential-proposal approach has been proposed as extra chance generalized hybrid Monte Carlo (XCGHMC). We develop two novel methods in which the trajectories leading to proposals in HMC are automatically tuned to avoid doubling back, as in the No-U-Turn sampler (NUTS). The numerical efficiency of these new methods compare favorably to the NUTS. We additionally show that the sequential-proposal bouncy particle sampler enables the constructed Markov chain to pass through regions of low target density and thus facilitates better mixing of the chain when the target density is multimodal.

Keywords Markov chain Monte Carlo · No-U-Turn sampler · Hamiltonian Monte Carlo · Peskun ordering

1 Introduction

Markov chain Monte Carlo (MCMC) methods are widely used to sample from distributions with analytically tractable unnormalized densities. In this paper, we explore an MCMC framework in which proposals for the next state of the Markov chain are drawn sequentially. We consider the objective of obtaining samples from a target distribution on a measurable space $(\mathbb{X}, \mathcal{X})$ with density

$$\bar{\pi}(x) := \frac{\pi(x)}{Z}$$

with respect to a reference measure denoted by dx, where $\pi(x)$ denotes an unnormalized density, and Z denotes the corresponding normalizing constant. MCMC methods construct Markov chains such that, given the current state of the Markov chain $X^{(i)}$, the next state $X^{(i+1)}$ is drawn from a kernel which has the target distribution $\bar{\pi}$ as its invariant distribution. The widely used Metropolis–Hastings (MH) strategy constructs a kernel with a specified invariant distribution in the following two steps (Metropolis et al. 1953; Hastings 1970). First, a proposal Y is drawn from a proposal kernel, and second, the proposal is accepted as $X^{(i+1)}$ with a certain probability. When the proposal is not accepted, the next state of the chain is set equal to the current state $X^{(i)}$. The acceptance probability depends on the target density and the proposal kernel density at $X^{(i)}$ and Y in a way that ensures that $\bar{\pi}$ is a stationary density of the constructed Markov chain.

The typical size of proposal increments and the mean acceptance probability affect the rate of mixing of the constructed Markov chain and thus the numerical efficiency of the algorithm. There is often a balance to be made between the size of proposal increments and the mean acceptance probability. Theoretical studies on this trade-off have been carried out for several widely used algorithms, such as random walk Metropolis (Roberts et al. 1997), Metropolis adjusted Langevin algorithm (MALA) (Roberts and Rosenthal 1998), or Hamiltonian Monte Carlo (HMC) (Beskos et al. 2013), in an asymptotic scenario where the target density is given by the product of d identical copies of a one dimensional density and where d tends to infinity. These results suggest that the optimal balance can be made by aiming at a certain value of the mean acceptance probability which depends on

Electronic supplementary material The online version of this article (https://doi.org/10.1007/s11222-020-09948-4) contains supplementary material, which is available to authorized users.

✉ Joonha Park
joonhap@bu.edu

[1] Boston University, Boston, USA

the algorithm but not on the target density, provided that the marginal density satisfies some regularity conditions.

Alternative methods to the basic Metropolis–Hastings strategy have been proposed to improve the numerical efficiency beyond the optimal balance between the proposal increment size and the acceptance probability. The multiple-try Metropolis method by Liu et al. (2000) makes multiple proposals given the current state of the Markov chain and selects one of them as a candidate for the next state of the Markov chain. Calderhead (2014) proposed a different algorithm that makes multiple proposals and allows more than one of them to be taken as the samples in the Markov chain. Since multiple proposals can be made independently in these methods, parallelization can increase computational efficiency. These methods make a preset number of proposals conditional on the current state in the Markov chain at each iteration.

Developments in various other directions have been made to improve the numerical efficiency of MCMC sampling. Adaptive MCMC methods use transition kernels that adapt over time using the information about the target distribution provided by the past history of the constructed chain (Haario et al. 2001; Andrieu and Thoms 2008). The update scheme for the transition kernel is designed to induce a sequence of transition kernels that converges to one that is efficient for the target distribution. The convergence of the law of the constructed chain and the rate of convergence have been studied under certain sets of conditions (Haario et al. 2001; Atchadé and Rosenthal 2005; Andrieu and Moulines 2006; Andrieu and Atchade 2007; Roberts and Rosenthal 2007; Atchadé and Fort 2010, 2012). Note however that the performance of an adaptive MCMC algorithm is limited by the efficiencies of the candidate transition kernels. In a different approach, Goodman and Weare (2010) proposed using ensemble samplers that construct Markov chains that are equally efficient for all target distributions that are affine transformations of each other. These methods draw information about the shape of the target distribution from parallel chains which jointly target the product distribution given by identical copies of the target density.

There also exist a class of methods that address difficulties in sampling from multimodal distributions using local proposals. Methods in this class include parallel tempering (Geyer 1991; Hukushima and Nemoto 1996), simulated tempering (Marinari and Parisi 1992), and the equi-energy sampler (Kou et al. 2006). In these methods, the mixing of the constructed Markov chain is aided by a set of other Markov chains that target alternative distributions for which the moves between separated modes happen more frequently. The equi-energy sampler bears a similarity with the approach of slice sampling, where a new sample is obtained within a randomly chosen level set of the target density (Roberts and Rosenthal 1999; Mira et al. 2001; Neal 2003).

In this paper, we explore a novel approach where proposals are drawn sequentially conditional on the previous proposal in each iteration. The proposal draws continue until a desired number of "acceptable" proposals are made, so the total number of proposals is variable. A key element in this approach is that the decision of acceptance or rejection of proposals are coupled via a single uniform$(0, 1)$ random variable drawn at the start of each iteration. This feature makes a straightforward generalization of the Metropolis–Hastings acceptance or rejection strategy. The approach is applicable to a wide range of commonly used MCMC algorithms, including ones that use proposal kernels with well defined densities and others that use deterministic proposal maps, such as Hamiltonian Monte Carlo (Duane et al. 1987) or the recently proposed bouncy particle sampler (Peters et al. 2012; Bouchard-Côté et al. 2018). We will demonstrate that the sequential-proposal approach is flexible; it is possible to make various modifications in order to develop methods that possess specific strengths.

The advantage of the sequential-proposal approach can be explained using Peskun-Tierney ordering (Peskun 1973; Tierney 1998; Andrieu and Livingstone 2019). Suppose two transition kernels P_1 and P_2 defined on $(\mathbb{X}, \mathcal{X})$ are reversible with respect to $\bar{\pi}$:

$$\int \mathbb{1}_{A\times B}(x, y) P_j(x, dy)\pi(x)dx = \int \mathbb{1}_{B\times A}(x, y) P_j(x, dy)\pi(x)dx, \quad \forall A, B \in \mathcal{X},\ j = 1, 2.$$

The transition kernel P_1 is said to dominate P_2 off the diagonal if

$$P_1(x, A \setminus \{x\}) \geq P_2(x, A \setminus \{x\}), \qquad \forall x \in \mathbb{X},\ \forall A \in \mathcal{X}.$$

For a $\mathcal{X}$-measurable function f such that $\int f^2(x)\bar{\pi}(x)dx < \infty$, consider an estimator of $\int f(x)\bar{\pi}(x)dx$ given by $\hat{I}(f) := \frac{1}{M}\sum_{i=1}^{M} f(X^{(i)})$. Define a scaled asymptotic variance of the estimator with respect to the kernels P_j, $j = 1, 2$ by

$$v(f, P_j) := \lim_{M\to\infty} M\mathrm{Var}_{P_j}\left(\hat{I}(f)\right), \qquad j = 1, 2,$$

where $X^{(0)}$ is assumed to be drawn from the stationary density $\bar{\pi}$ and $X^{(i)}$ is drawn from $P_j(X^{(i-1)}, \cdot)$, $i \geq 1$. Provided that P_1 dominates P_2 off the diagonal, we have $v(f, P_1) \leq v(f, P_2)$ (Tierney 1998, Theorem 4). Suppose now we denote by $P_1(x, A \setminus \{x\})$ the conditional probability that $X^{(i)}$ is in $A \setminus \{x\}$ given $X^{(i-1)} = x$ when a sequential-proposal method is used and by $P_2(x, A \setminus \{x\})$ the same conditional probability when a standard MH method is used. Then

P_1 dominates P_2 off the diagonal, because the sequential-proposal method tries additional proposals when the first proposal is rejected. Due to Peskun-Tierney ordering, the asymptotic variance of the estimate $\hat{I}(f)$ from the sequential-proposal method is always smaller than or equal to that from the standard MH. In fact, a similar argument also motivated the development of the delayed rejection method (Tierney and Mira 1999; Green and Mira 2001), which we will show to be equivalent to the sequential-proposal approach under certain circumstances in terms of the law of the constructed Markov chains.

After the initial arXiv posting of this manuscript, we became aware of the extra chance generalized hybrid Monte Carlo (XCGHMC) of Campos and Sanz-Serna (2015) that develops a method equivalent to the sequential-proposal approach in the context of HMC. Our work departs from Campos and Sanz-Serna (2015) in two important ways. First, the current paper develops sequential-proposal MCMC more broadly, and as a general recipe for improving on MCMC algorithms. Second, in applying the idea to HMC our emphasis differs from that in Campos and Sanz-Serna (2015). One of the main challenges in using HMC in practice is the tuning of the number of leapfrog jumps. Motivated by this issue, we use the sequential-proposal MCMC framework to develop two novel HMC methods that automatically tune the lengths of the leapfrog trajectories in such a way that the well-known double-backing issue in HMC is avoided, in the same spirit as the No-U-Turn sampler (NUTS) of Hoffman and Gelman (2014). Our numerical results on a multivariate normal distribution and a real data example show that the efficiencies of these new methods compare favorably to that of the NUTS.

Our paper is organized as follows. In Sect. 2 we explain the sequential-proposal approach in the case where proposal kernels have well defined densities. We call the resulting methods as sequential-proposal Metropolis–Hastings algorithms. An equivalence between between our approach and the delayed rejection method of Mira (2001) under certain settings is discussed in this section. In Sect. 3, we explain how the sequential-proposal approach can be applied to a class of MCMC algorithms that use deterministic proposal maps. Based on this formulation, we develop two variants of the NUTS algorithm in Sect. 4. In Sect. 5 we propose a novel discrete time bouncy particle sampler method based on the sequential-proposal approach and demonstrate some desirable numerical properties that enable faster mixing of the Markov chain for multimodal target distributions. Section 6 gives a summarizing conclusion. Proofs of most theoretical results are given in Online Resource S1–S5.

Algorithm 1: A sequential-proposal Metropolis-Hastings algorithm

Input : Distribution for the maximum number of proposals and the number of accepted proposals, $\nu(N, L)$
Possibly asymmetric proposal kernel, $q(y_n \mid y_{n-1})$
Number of iterations, M

Output: A draw of Markov chain, $(X^{(i)})_{i\in 1:M}$

Initialize: Set $X^{(0)}$ arbitrarily
for $i \leftarrow 0 : M-1$ **do**
  Draw $(N, L) \sim \nu(\cdot, \cdot)$
  Draw $\Lambda \sim$ unif(0, 1)
  Set $X^{(i+1)} \leftarrow X^{(i)}$
  Set $Y_0 \leftarrow X^{(i)}$ and $n_a \leftarrow 0$
  for $n \leftarrow 1 : N$ **do**
    Draw $Y_n \sim q(\cdot \mid Y_{n-1})$
    if $\Lambda < \frac{\pi(Y_n)\prod_{j=1}^{n} q(Y_{j-1} \mid Y_j)}{\pi(Y_0)\prod_{j=1}^{n} q(Y_j \mid Y_{j-1})}$ **then** $n_a \leftarrow n_a + 1$
    if $n_a = L$ **then**
      Set $X^{(i+1)} \leftarrow Y_n$
      `break`
    end
  end
end

2 Sequential-proposal Metropolis–Hastings algorithms

We will first explain the sequential-proposal approach when the proposal kernel has well defined density with respect to the reference measure of the target density $\bar{\pi}$. In standard Metropolis–Hasting algorithms, given the current state $X^{(i)} = x$ at the i-th iteration of the algorithm, the proposal Y is drawn from a probability kernel with conditional density $q(y \mid x)$. The proposal $Y = y$ is accepted with the probability

$$\min\left(1, \frac{\pi(y)q(x \mid y)}{\pi(x)q(y \mid x)}\right).$$

This is often implemented by drawing a uniform random variable $\Lambda \sim$ unif(0, 1) and accepting the proposal by setting $X^{(i+1)} \leftarrow Y$ if and only if $\Lambda < \frac{\pi(y)q(x \mid y)}{\pi(x)q(y \mid x)}$. If Y is not accepted, the algorithm sets $X^{(i+1)} \leftarrow X^{(i)}$.

We will call Y_1 the first proposal drawn from $q(\cdot \mid X^{(i)})$. The proposal Y_1 is rejected if and only if a uniform random number $\Lambda \sim$ unif(0, 1) is greater than or equal to

$$\frac{\pi(Y_1)q(X^{(i)} \mid Y_1)}{\pi(X^{(i)})q(Y_1 \mid X^{(i)})}.$$

If rejected, a second proposal Y_2 is drawn from $q(\cdot \mid Y_1)$. The second proposal is accepted if and only if

$$\Lambda < \frac{\pi(Y_2)q(Y_1 \mid Y_2)q(X^{(i)} \mid Y_1)}{\pi(X^{(i)})q(Y_1 \mid X^{(i)})q(Y_2 \mid Y_1)}$$

using the same value of Λ used previously. If accepted, the algorithm sets $X^{(i+1)} \leftarrow Y_2$. The n-th proposal Y_n is deemed acceptable if and only if

$$\Lambda < \frac{\pi(Y_n)\prod_{j=1}^{n} q(Y_{j-1} \mid Y_j)}{\pi(Y_0)\prod_{j=1}^{n} q_j(Y_j \mid Y_{j-1})}, \tag{1}$$

where Y_0 denotes the current state of the Markov chain $X^{(i)}$. This procedure is repeated until an acceptable proposal is found or until a preset number N of proposals are all rejected, whichever is reached sooner. In the case where all N proposals are rejected, the algorithm sets $X^{(i+1)} \leftarrow X^{(i)}$. The algorithm reduces to a standard Metropolis–Hastings algorithm if we set $N = 1$.

More generally, a sequential-proposal Metropolis–Hastings algorithm can take the L-th acceptable proposal for some $L \geq 1$ as the next state of the Markov chain. If there are less than L acceptable proposals in the first N proposals, the Markov chain stays at its current position. The algorithmic parameters N and L may be randomly selected at each iteration, provided that they are independent of the proposals $\{Y_n\,;\, n \geq 1\}$ and Λ. For some algorithms such as the Hamiltonian Monte Carlo methods, moving as far as possible at each step can improve the numerical efficiency of the algorithms, so the choice $L \geq 1$ may be considered. A pseudocode for a sequential-proposal Metropolis algorithm is given in Algorithm 1.

We will now show that the sequential-proposal Metropolis–Hastings algorithm constructs a reversible Markov chain with respect to the target distribution with density $\bar{\pi}$. Throughout this paper, for two integers n and m, we will denote by $n:m$ the sequence $(n, n+1, \ldots, m)$ if $n \leq m$ and the sequence $(n, n-1, \ldots, m)$ if $n > m$. Also, given a sequence $(a_n)_{n\geq 0} = (a_0, a_1, a_2, \ldots)$, we will denote by $a_{n:m}$ the subsequence $(a_j)_{n\leq j\leq m}$.

Proposition 1 *Algorithm 1 constructs a reversible Markov chain* $\left(X^{(i)}\right)$ *with respect to the target density* $\bar{\pi}$.

Proof For simplicity, we prove the proposition for the case where $L = 1$ and the proposal kernel with conditional density $q(y \mid x)$ is symmetric in the sense that $q(y \mid x) = q(x \mid y)$ for all $x, y \in \mathbb{X}$. If the proposal density q is symmetric, (1) reduces to $\pi(Y_n)/\pi(Y_0)$. The proof for asymmetric proposal density q and general $L \geq 1$ is given in Online Resource S1.

We will show that the detailed balance equation

$$\mathcal{P}[X^{(i)} \in A,\ X^{(i+1)} \in B] = \mathcal{P}[X^{(i)} \in B,\ X^{(i+1)} \in A]$$

holds for every pair of measurable subsets A and B of $\mathbb{X}$, provided that $X^{(i)}$ is distributed according to $\bar{\pi}$. We will write $Y_0 := X^{(i)}$, and the subsequent proposals as $Y_1, Y_2, \ldots, Y_N$. The case where the n-th proposal Y_n is taken for $X^{(i+1)}$ will be considered; then the claim of detailed balance will follow by combining the cases for n in $1:N$ and the case where all proposals are rejected. Under the assumption that $X^{(i)}$ is distributed according to $\bar{\pi}$, the probability that $X^{(i)}$ is in A and the n-th proposal is in B and taken as $X^{(i+1)}$ is given by

$$\begin{aligned}
&\mathcal{P}[X^{(i)} \in A,\ X^{(i+1)} \in B,\ \text{the } n\text{-th proposal is taken as } X^{(i+1)}] \\
&\quad = \int \mathbb{1}_A(y_0)\mathbb{1}_B(y_n)\bar{\pi}(y_0)q(y_1 \mid y_0)\cdots q(y_n \mid y_{n-1}) \\
&\quad \cdot \mathbb{1}\left[\Lambda \geq \frac{\pi(y_1)}{\pi(y_0)}\right]\cdots\mathbb{1}\left[\Lambda \geq \frac{\pi(y_{n-1})}{\pi(y_0)}\right] \\
&\quad \mathbb{1}\left[\Lambda < \frac{\pi(y_n)}{\pi(y_0)}\right]\mathbb{1}[0 < \Lambda < 1]d\Lambda\, dy_0\, dy_1 \ldots dy_n,
\end{aligned} \tag{2}$$

where $\mathbb{1}_A$ denotes the indicator function for the set A and $\mathbb{1}[\cdot]$ denotes the indicator function of the event specified between the brackets. The quantity

$$\begin{aligned}
&\mathbb{1}\left[\Lambda \geq \frac{\pi(y_1)}{\pi(y_0)}\right]\cdots\mathbb{1}\left[\Lambda \geq \frac{\pi(y_{n-1})}{\pi(y_0)}\right] \\
&\quad \mathbb{1}\left[\Lambda < \frac{\pi(y_n)}{\pi(y_0)}\right]\mathbb{1}[0 < \Lambda < 1]
\end{aligned} \tag{3}$$

is equal to unity if and only if

$$\Lambda \geq \max_{k\in 1:n-1}\frac{\pi(y_k)}{\pi(y_0)} \quad \text{and} \quad \Lambda < \min\left(1, \frac{\pi(y_n)}{\pi(y_0)}\right). \tag{4}$$

It can be readily observed that for real numbers x, a, and b, the conditions $x \geq a$ and $x < b$ are satisfied if and only if $x \in [\min\{a, b\},\ b)$, where the interval length is given by $b - \min(a, b)$. Thus the interval length corresponding to the conditions (4) is given by

$$\min\left(1, \frac{\pi(y_n)}{\pi(y_0)}\right) - \min\left(1, \frac{\pi(y_n)}{\pi(y_0)}, \max_{k\in 1:n-1}\frac{\pi(y_k)}{\pi(y_0)}\right),$$

which gives the integral of (3) over Λ. It follows that (2) is equal to

$$\begin{aligned}
&\int \mathbb{1}_A(y_0)\mathbb{1}_B(y_n)\bar{\pi}(y_0)\prod_{k=1}^{n} q(y_k \mid y_{k-1})\cdot \\
&\qquad \left[\min\left(1, \frac{\pi(y_n)}{\pi(y_0)}\right) - \min\left(1, \frac{\pi(y_n)}{\pi(y_0)}, \max_{k\in 1:n-1}\frac{\pi(y_k)}{\pi(y_0)}\right)\right] dy_{0:n} \\
&\quad = \frac{1}{Z}\int \mathbb{1}_A(y_0)\mathbb{1}_B(y_n)\prod_{k=1}^{n} q(y_k \mid y_{k-1})\cdot[\min\{\pi(y_0), \pi(y_n)\} \\
&\qquad - \min\{\pi(y_0), \pi(y_n), \max_{k\in 1:n-1}\pi(y_k)\}\Big] dy_{0:n}.
\end{aligned} \tag{5}$$

If we change the notation of the dummy variables by writing $y_0 \leftarrow y_n, y_1 \leftarrow y_{n-1}, \ldots, y_n \leftarrow y_0$, then (5) is given by

$$\frac{1}{Z}\int \mathbb{1}_A(y_n)\mathbb{1}_B(y_0)\prod_{k=1}^{n} q(y_k \mid y_{k-1}) \left[\min\{\pi(y_n),\pi(y_0)\} - \min\{\pi(y_n),\pi(y_0), \max_{k\in 1:n-1}\pi(y_k)\}\right] dy_{0:n}, \tag{6}$$

where we have used the fact that the kernel density q is symmetric; that is, $q(y_{k-1} \mid y_k) = q(y_k \mid y_{k-1})$, for $k \in 1:n$. It is now obvious that (6) is equal to the quantity obtained by swapping the positions of A and B in (2). Thus we see that

$$\begin{aligned}&\mathcal{P}[X^{(i)} \in A,\ X^{(i+1)} \in B, \text{the } n\text{-th proposal is taken as } X^{(i+1)}]\\ &\quad = \mathcal{P}[X^{(i)} \in B,\ X^{(i+1)} \in A, \text{the } n\text{-th proposal is taken as } X^{(i+1)}].\end{aligned} \tag{7}$$

In the case where all N proposals are rejected, the algorithms sets $X^{(i+1)} \leftarrow X^{(i)}$. Thus,

$$\begin{aligned}&\mathcal{P}[X^{(i)} \in A,\ X^{(i+1)} \in B, \text{all } N \text{ proposals are rejected}]\\ &\quad = \mathcal{P}[X^{(i+1)} \in A,\ X^{(i)} \in B, \text{all } N \text{ proposals are rejected}]\end{aligned} \tag{8}$$

Thus summing (7) over all $n \in 1:N$ and adding (8) gives

$$\mathcal{P}[X^{(i)} \in A,\ X^{(i+1)} \in B] = \mathcal{P}[X^{(i)} \in B,\ X^{(i+1)} \in A].$$

□

A sequential-proposal Metropolis–Hastings algorithm can also employ proposal kernels that depend on the sequence of previous proposals. Suppose that proposals are sequentially drawn in such a way that the k-th candidate Y_k is drawn from a proposal kernel with density $q_k(\cdot \mid Y_{k-1}, \ldots, Y_0)$, where $Y_{k-1}, \ldots, Y_1$ denote the previous proposals and Y_0 denotes the current state $X^{(i)}$ in the Markov chain at the i-th iteration. The candidate Y_k is deemed acceptable if

$$\Lambda < \frac{\pi(Y_k)\prod_{j=1}^{k} q_j(Y_{k-j} \mid Y_{k-j+1:k})}{\pi(Y_0)\prod_{j=1}^{k} q_j(Y_j \mid Y_{j-1:0})}. \tag{9}$$

Proposals are sequentially drawn until L acceptable proposals are found. If there are less than L acceptable proposals among the first N proposals, the next state in the Markov chain is set to the current state, $X^{(i+1)} \leftarrow X^{(i)}$. Suppose now that the L-th acceptable state is obtained by the n-th proposal Y_n for some $n \le N$. In the case where the proposal kernel depends on the sequence of previous proposals, in order to take Y_n as the next state of the Markov chain, an additional condition needs to be checked, namely that there are exactly $L-1$ numbers $k \in 1:n-1$ that satisfy

$$\Lambda < \frac{\pi(Y_k)\prod_{j=1}^{n-k} q_j(Y_{k+j} \mid Y_{k+j-1:k})\prod_{j=n-k+1}^{n} q_j(Y_{n-j} \mid Y_{n-j+1:n})}{\pi(Y_0)\prod_{j=1}^{n} q_j(Y_j \mid Y_{j-1:0})}. \tag{10}$$

If this additional condition is satisfied, Y_n is taken as the next state of the Markov chain, that is $X^{(i+1)} \leftarrow Y_n$. Otherwise, the next state is set to the current state in the Markov chain, $X^{(i+1)} \leftarrow X^{(i)}$. A pseudocode for sequential-proposal Metropolis–Hastings algorithms that employ kernels dependent on the sequence of previous proposals are given in Online Resource S2. The role of the additional condition (10) is to establish detailed balance between $X^{(i)}$ and $X^{(i+1)}$ by creating a symmetry between the sequence of proposals $Y_0 \to Y_1 \to \cdots \to Y_n$ and the reversed sequence $Y_n \to Y_{n-1} \to \cdots \to Y_0$. To see this, we note that the candidate Y_n can be taken as the next state of the Markov chain only when there are exactly $L-1$ acceptable proposals among $Y_1, \ldots, Y_{n-1}$. The additional symmetry condition accounts for a mirror case where there are $L-1$ acceptable proposals among $Y_{n-1}, \ldots, Y_1$, assuming that these proposals are sequentially drawn in the reverse order starting from Y_n. A proof of detailed balance for this algorithm is also given in Online Resource S2. This algorithm reduces to Algorithm 1 in the case where the proposal kernel is dependent only on the most recent proposal. Heuristically, the symmetry condition (10) will be satisfied with high probability if the dependence of q_j on the previous proposals other than the most recent one is small and if $\pi(Y_k)$ varies greatly for the sequence of proposals $\{Y_n ; n \ge 1\}$. In this case, the difference between the acceptability condition (9) and the mirror condition (10) will be small, and the Y_k that satisfies (9) will be likely to satisfy (10).

2.1 Relationship with the delayed rejection method

Sequential-proposal Metropolis–Hastings algorithms in the case where $L = 1$ construct the same Markov chains as those constructed by delayed rejection methods (Tierney and Mira 1999; Mira 2001; Green and Mira 2001) when the proposal kernel depends only on the most recent proposal. A brief description of the delayed rejection method, following Mira (2001), is given as follows. Given the current state of the Markov chain y_0, the first candidate value y_1 is drawn from $q(\cdot \mid y_0)$ and accepted with probability

$$\alpha_1(y_0, y_1) = 1 \wedge \frac{\pi(y_1)q(y_0 \mid y_1)}{\pi(y_0)q(y_1 \mid y_0)},$$

where $a \wedge b := \min(a, b)$. If y_1 is rejected, a next candidate value y_2 is drawn from $q(\cdot \mid y_1)$. The acceptance probability for y_2 is given by

$$\alpha_2(y_0, y_1, y_2) = 1 \wedge \frac{\pi(y_2)q(y_1 \mid y_2)q(y_0 \mid y_1)\{1 - \alpha_1(y_2, y_1)\}}{\pi(y_0)q(y_1 \mid y_0)q(y_2 \mid y_1)\{1 - \alpha_1(y_0, y_1)\}}.$$

If $y_1, \ldots, y_{n-1}$ are rejected, y_n is drawn from $q(\cdot \mid y_{n-1})$ and accepted with probability

$$\alpha_n(y_{0:n}) = 1 \wedge \frac{\pi(y_n)\prod_{j=1}^{n} q(y_{j-1} \mid y_j)\prod_{j=1}^{n-1}\{1 - \alpha_j(y_{n:n-j})\}}{\pi(y_0)\prod_{j=1}^{n} q(y_j \mid y_{j-1})\prod_{j=1}^{n-1}\{1 - \alpha_j(y_{0:j})\}}.$$

If all proposals are rejected up to a certain number N, the next state of the Markov chain is set to the current state y_0. We show in Online Resource S3 the equivalence between the delayed rejection method and the sequential-proposal Metropolis–Hastings algorithm with $L = 1$ under the case where each proposal is made depending only on the most recent proposal. The delayed rejection method can also use proposal kernels dependent on the sequence of previous proposals to construct a reversible Markov chain with respect to the target distribution. In this case, however, the law of the constructed Markov chain will be different from that by a sequential-proposal Metropolis–Hastings algorithm.

In our view, there are several advantages sequential-proposal Metropolis–Hastings algorithms have over the delayed rejection method:

1. Sequential-proposal Metropolis–Hastings algorithms are more straightforward to implement than the delayed rejection method. The evaluation of $\alpha_n(y_{0:n})$ in delayed rejection involves the evaluation of a sequence of reversed acceptance probabilities $\{\alpha_j(y_{n:n-j}) ; j \in 1 : n-1\}$. This involves computation of a total of $\mathcal{O}(n^2)$ acceptance probabilities. In comparison, sequential-proposal Metropolis–Hastings algorithms only compare the ratio in (1) to a uniform random number Λ for the same task of checking the acceptability of y_n. The algorithmic simplicity of sequential-proposal Metropolis–Hastings facilitates the use of a large number of proposals in each iteration. Moreover, one may choose to take the L-th acceptable proposal for the next state of the Markov chain for a large $L > 1$.
2. The sequential-proposal MCMC framework can be readily applied to MCMC algorithms using deterministic maps for proposals, as explained in Sect. 3. In particular, the sequential-proposal MCMC framework applies to Hamiltonian Monte Carlo and the bouncy particle sampler methods, leading to improved the numerical efficiency. Applications to these algorithms are discussed in Sects. 4 and 5. We note that the delayed rejection method has been generalized to algorithms using deterministic maps in Green and Mira (2001), although only the case for the second proposal was discussed.
3. The conceptual simplicity of the sequential-proposal MCMC framework allows for various generalizations and modifications. For example, in Sect. 4.2, we develop sequential-proposal No-U-Turn sampler algorithms (Algorithms 5 and 6) that automatically adjust the lengths of trajectories leading to proposals in HMC, similarly to the No-U-Turn sampler algorithm proposed by Hoffman and Gelman (2014). The proofs of detailed balance for these algorithms can be obtained by making minor modifications to the proof for the sequential-proposal Metropolis–Hastings algorithms.

3 Sequential-proposal MCMC algorithms using deterministic kernels

The sequential-proposal MCMC framework can be applied to algorithms that use deterministic proposal kernels. MCMC algorithms that employ deterministic proposal kernels often target a distribution on an extended space $\mathbb{X} \times \mathbb{V}$ whose marginal distribution on $\mathbb{X}$ is equal to the original target distribution $\bar{\pi}$. An additional variable V drawn from a distribution on $\mathbb{V}$ serves as a parameter for the deterministic proposal kernel. In this section, we will explain a general class of MCMC algorithms using deterministic proposal kernels and show how the sequential-proposal scheme can be applied to these algorithms. Applications to specific algorithms, such as HMC or the bouncy particle sampler (BPS), are discussed in subsequent sections (Sects. 4 and 5).

We suppose that the extended target distribution on $\mathbb{X} \times \mathbb{V}$ has density $\Pi(x, v)$ with respect to a reference measure denoted by $dx\, dv$. We further assume that the original target density $\bar{\pi}$ equals the marginal density of Π, such that $\Pi(x, v) = \bar{\pi}(x)\psi(v ; x)$ for some $\psi(v ; x)$, the conditional density of v given x. We define a collection of deterministic maps $\mathcal{S}_\tau : \mathbb{X} \times \mathbb{V} \to \mathbb{X} \times \mathbb{V}$ for possibly various values of τ. In HMC and the BPS, $\mathcal{S}_\tau$ has an analogy with the evolution of a particle in a physical system for a time duration τ. In this analogy, the variable $x \in \mathbb{X}$ is considered as the position of a particle in the system and the variable $v \in \mathbb{V}$ as the velocity of the particle. The point $\mathcal{S}_\tau(x, v)$ then represents the final position-velocity pair of a particle that moves with initial position x and initial velocity v for time τ. We suppose that the map $\mathcal{S}_\tau$ for each τ satisfies the following condition:

Reversibility condition *There exists a velocity reflection operator $\mathcal{R}_x : \mathbb{V} \to \mathbb{V}$ defined for every point $x \in \mathbb{X}$ such that*

$$\mathcal{R}_x \circ \mathcal{R}_x = \mathcal{I}, \tag{11}$$

holds for every $x \in \mathbb{X}$ *and*

$$\frac{\psi(\mathcal{R}_x v\,;\, x)}{\psi(v\,;\, x)} \left|\frac{\partial \mathcal{R}_x v}{\partial v}\right| = 1 \tag{12}$$

holds for almost every $(x, v) \in \mathbb{X} \times \mathbb{V}$ *with respect to the reference measure* $dx\,dv$. *Furthermore, if we define a map* $\mathcal{T} : \mathbb{X} \times \mathbb{V} \to \mathbb{X} \times \mathbb{V}$ *as* $\mathcal{T}(x, v) := (x, \mathcal{R}_x v)$, *we have*

$$\mathcal{T} \circ \mathcal{S}_\tau \circ \mathcal{T} \circ \mathcal{S}_\tau = \mathcal{I}. \tag{13}$$

Similar sets of conditions appear routinely in the literature on MCMC (Fang et al. 2014; Vanetti et al. 2017) and on Hamiltonian dynamics (Leimkuhler and Reich 2004, Section 4.3). In (11) and (13), $\mathcal{I}$ denotes the identity map in the corresponding space $\mathbb{V}$ or $\mathbb{X} \times \mathbb{V}$, and the symbol $\circ$ denotes function composition. In (12), $\left|\frac{\partial \mathcal{R}_x v}{\partial v}\right|$ denotes the absolute value of the Jacobian determinant of the map $\mathcal{R}_x$ at v. The condition (12) is equivalent to the condition that

$$\int_A \Pi(x, v)dv = \int_{\mathcal{R}_x(A)} \Pi(x, v)dv \tag{14}$$

for every measurable subset A of $\mathbb{V}$ and for almost every $x \in \mathbb{X}$, due to the change of variable formula. The condition (13) can be understood as an abstraction of a property in Hamiltonian dynamics that if we reverse the velocity of a particle and advance in time, the particle traces back its past trajectory.

Given $X^{(i)} = x$ and $V^{(i)} = v$ at the start of the i-th iteration, an MCMC algorithm can make a deterministic proposal $\mathcal{S}_\tau(x, v)$, which is accepted with probability

$$\min\left(1, \frac{\Pi(\mathcal{S}_\tau(x, v))}{\Pi(x, v)} \left|\det D\mathcal{S}_\tau(x, v)\right|\right),$$

where D denotes the differential operator (i.e., $D\mathcal{S}_\tau(x, v) = \frac{\partial \mathcal{S}_\tau(x,v)}{\partial(x,v)}$). In algorithms such as HMC or the BPS, the extended target density $\Pi(x, v)$ is often taken as a product of independent densities, $\bar{\pi}(x)\psi(v)$, where a common choice for $\psi(v)$ is a multivariate normal density. The map $\mathcal{S}_\tau$ is often taken to preserve the reference measure, such that it has unit Jacobian determinant (i.e., $|\det D\mathcal{S}_\tau(x, v)| = 1$, for all (x, v)).

The sequential-proposal framework can be used to generalize MCMC algorithms using deterministic kernels in a similar way that it is applied to Metropolis–Hastings algorithms. A pseudocode of a sequential-proposal MCMC algorithm using a deterministic kernel is shown in Algorithm 2. Proposals are obtained sequentially as $(Y_n, W_n) \leftarrow \mathcal{S}_\tau(Y_{n-1}, W_{n-1})$, where we write $(Y_0, W_0) := (X^{(i)}, V^{(i)})$.

Algorithm 2: A sequential-proposal MCMC using a deterministic kernel

Input : Distribution of the maximum number of proposals and the number of accepted proposals, $\nu(N, L)$
Time step length distribution, $\mu(d\tau)$
Velocity distribution density, $\psi(v\,;\, x)$
Time evolution operators, $\{\mathcal{S}_\tau\}$
Velocity reflection operators, $\{\mathcal{R}_x\}$
Velocity refreshment probability, $p^{\mathrm{ref}}(x)$
Number of iterations, M

Output: A draw of Markov chain, $\left(X^{(i)}\right)_{i \in 1:M}$

Initialize: Set $X^{(0)}$ arbitrarily and draw $V^{(0)} \sim \psi(\cdot\,;\, X^{(0)})$.
for $i \leftarrow 0 : M-1$ **do**
  Draw $N, L \sim \nu(\cdot, \cdot)$
  Draw $\tau \sim \mu(\cdot)$
  Draw $\Lambda \sim \mathrm{unif}(0, 1)$
  Set $X^{(i+1)} \leftarrow X^{(i)}$ and $V^{(i+1)} \leftarrow \mathcal{R}_{X^{(i)}} V^{(i)}$
  Set $n_a \leftarrow 0$
  Set $(Y_0, W_0) \leftarrow (X^{(i)}, V^{(i)})$
  for $n \leftarrow 1 : N$ **do**
    Set $(Y_n, W_n) \leftarrow \mathcal{S}_\tau(Y_{n-1}, W_{n-1})$
    if $\Lambda < \frac{\pi(Y_n)\psi(W_n\,;\, Y_n)}{\pi(Y_0)\psi(W_0\,;\, Y_0)} \left|\det D\mathcal{S}_\tau^n(Y_0, W_0)\right|$ **then**
      $n_a \leftarrow n_a + 1$
      if $n_a = L$ **then**
        Set $(X^{(i+1)}, V^{(i+1)}) \leftarrow (Y_n, W_n)$
        `break`
      end
    end
  end
  With probability $p^{\mathrm{ref}}(X^{(i+1)})$, refresh $V^{(i+1)} \sim \psi(\cdot\,;\, X^{(i+1)})$
end

The pair (Y_n, W_n) is deemed acceptable if

$$\Lambda < \frac{\Pi(Y_n, W_n)}{\Pi(Y_0, W_0)} \left|\det D\mathcal{S}_\tau^n(Y_0, W_0)\right|,$$

where $\mathcal{S}_\tau^n = \mathcal{S}_\tau \circ \cdots \circ \mathcal{S}_\tau$ denotes a map obtained by composing $\mathcal{S}_\tau$ n times. If there are less than L acceptable proposals in the sequence of L proposals, the next state of the Markov chain is set to $(X^{(i+1)}, V^{(i+1)}) \leftarrow (X^{(i)}, \mathcal{R}_{X^{(i)}} V^{(i)})$. The velocity $V^{(i+1)}$ may be refreshed at the end of the iteration by drawing from $\psi(\cdot\,;\, X^{(i+1)})$ with a certain probability $p^{\mathrm{ref}}(X^{(i+1)})$ that may depend on $X^{(i+1)}$. The parameter τ for the evolution map $\mathcal{S}_\tau$ can be drawn randomly. The pseudocode in Algorithm 2 shows the case where τ is drawn once per iteration and the same value is used for all $n \in 1 : N$, but τ can also be drawn separately for each n, provided that the draws are independent of each other and of all other random draws in the algorithm.

We state the following result for Algorithm 2. The proof is given in Online Resource S4.

Proposition 2 *The extended target distribution with density* $\Pi(x, v)$ *is a stationary distribution for the Markov chain* $\left(X^{(i)}, V^{(i)}\right)_{i \in 1:M}$ *constructed by Algorithm 2. Furthermore, the Markov chain* $\left(X^{(i)}\right)_{i \in 1:M}$ *constructed by Algorithm 2, marginally for the* x*-component, is reversible with respect to the target distribution* $\bar{\pi}(x)$.

4 Connection to Hamiltonian Monte Carlo methods

4.1 Sequential-proposal Hamiltonian Monte Carlo

In this section, we consider applications of the sequential-proposal approach described in Sect. 3 to Hamiltonian Monte Carlo algorithms and discuss the numerical efficiency. We first briefly summarize basic features of HMC algorithms. A function on $\mathbb{X} \times \mathbb{V}$, called the Hamiltonian, is defined as the negative log density of the extended target density:

$$H(x, v) := -\log \Pi(x, v) = -\log \bar{\pi}(x) - \log \psi(v\,; x) \quad (15)$$

We assume both $\mathbb{X}$ and $\mathbb{V}$ are equal to the d dimensional Euclidean space $\mathbb{R}^d$. The velocity distribution $\psi(v\,; x)$ is often taken as a multivariate normal density independent of x,

$$\psi(v\,; x) \equiv \psi_C(v) := \frac{1}{\sqrt{(2\pi)^d\,|\det C|}} \exp\left\{-\frac{v^T C^{-1} v}{2}\right\}.$$

An analogy with a physical Hamiltonian system is drawn by interpreting the first term $-\log \bar{\pi}(x)$ as the static potential energy of a particle and the second term $-\log \psi(v)$ as the kinetic energy. In this analogy, the covariance matrix C can be interpreted as the inverse of the mass of the particle. Hamiltonian dynamics is defined as a solution to the Hamiltonian equation of motion (HEM):

$$\begin{aligned} \frac{dx}{dt} &= C\frac{\partial H}{\partial v} \\ \frac{dv}{dt} &= -C\frac{\partial H}{\partial x}. \end{aligned} \quad (16)$$

If we denote the solution to the HEM as $(x(t), v(t))$, the exact Hamiltonian flow S^*_τ defined by $\mathcal{S}^*_\tau(x(0), v(0)) := (x(\tau), v(\tau))$ satisfies the reversibility condition (12) and (13) when the velocity reflection operator is given by $\mathcal{R}_x(v) = -v$ for all $x \in \mathbb{X}$ and $v \in \mathbb{V}$. The map $\mathcal{S}^*_\tau$ preserves the Hamiltonian, that is, $H(x, v) = H(\mathcal{S}^*_\tau(x, v))$ for all $x \in \mathbb{X}$, $v \in \mathbb{V}$, and $\tau \geq 0$. The map $\mathcal{S}^*_\tau$ also preserves the reference measure $dx\,dv$, that is, $\left|\det D\mathcal{S}^*_\tau(x, v)\right| = 1$ for all $x \in \mathbb{X}$, $v \in \mathbb{V}$ and $\tau \geq 0$, which is known as Liouville's theorem (Liouville 1838).

A commonly used numerical approximation method for solving the HEM is called the leapfrog method (Duane et al. 1987; Leimkuhler and Reich 2004). One iteration of the leapfrog method approximates time evolution of a Hamiltonian system for duration ϵ by alternately updating the velocity and position (x, v) as follows:

$$\begin{aligned} v &\leftarrow v + \frac{\epsilon}{2} \cdot C \cdot \nabla \log \pi(x) \\ x &\leftarrow x + \epsilon v \\ v &\leftarrow v + \frac{\epsilon}{2} \cdot C \cdot \nabla \log \pi(x). \end{aligned} \quad (17)$$

Algorithm 3: Sequential-proposal HMC and leapfrog jump function

Input : Leapfrog step size, ϵ
Number of leapfrog jumps, l
Covariance of the velocity distribution, C

Output: A draw of Markov chain, $\left(X^{(i)}\right)_{i \in 1:M}$

1 Run Algorithm 2 with $\Pi(x, v) = \bar{\pi}(x)\psi_C(v)$, $p^{\mathrm{ref}}(x) = 1$, $\tau := (\epsilon, l)$, $\mathcal{S}_\tau(x, v) = \texttt{Leapfrog}(x, v, \epsilon, l, C)$, and $\mathcal{R}_x = -\mathcal{I}$.

2 **Function** `Leapfrog`(x, v, ϵ, l, C)
3 $\quad v \leftarrow v + \frac{\epsilon}{2} \cdot C \cdot \nabla \log \pi(x)$
4 $\quad x \leftarrow x + \epsilon v$
5 $\quad$ Set $j \leftarrow 1$
6 $\quad$ **while** $j < l$ **do**
7 $\quad\quad v \leftarrow v + \epsilon \cdot C \cdot \nabla \log \pi(x)$
8 $\quad\quad x \leftarrow x + \epsilon v$
9 $\quad\quad$ Set $j \leftarrow j + 1$
10 $\quad$ **end**
11 $\quad v \leftarrow v + \frac{\epsilon}{2} \cdot C \cdot \nabla \log \pi(x)$
12 **end**

We call the time increment ϵ the leapfrog step size.

A standard Hamiltonian Monte Carlo algorithm is a specific instance of MCMC algorithms using deterministic kernels described in Sect. 3, where the extended target density $\Pi(x, v)$ is given by $\bar{\pi}(x)\psi_C(v)$ and the proposal map $\mathcal{S}_\tau$ is given by l leapfrog jumps with step size ϵ, such that the time duration parameter τ can be understood as the pair (ϵ, l). The reversibility condition (11)–(13) is satisfied by this $\mathcal{S}_\tau$ with $\mathcal{R}_x = -\mathcal{I}$ for all $x \in \mathbb{X}$. Each step in the leapfrog method (17) preserves the reference measure $dx\,dv$, so we have $|\det D\mathcal{S}_\tau| \equiv 1$. It is common to refresh the probability at every iteration (i.e., $p^{\mathrm{ref}}(x) \equiv 1$).

Sequential-proposal HMC (Algorithm 3) is obtained as a specific case of sequential-proposal MCMC algorithms using deterministic kernels (Algorithm 2) under the same setting, $\Pi(x, v) = \bar{\pi}(x)\psi_C(v)$ and $\mathcal{S}_\tau = \mathcal{S}_{(\epsilon,l)}$. In other words, a proposal (Y_1, W_1) is made by making l leapfrog jumps of size ϵ starting from (Y_0, W_0), and if the proposal is rejected, a new proposal (Y_2, W_2) is made by making l leapfrog jumps from (Y_1, W_1). The procedure is repeated until L acceptable proposals are found, or until N proposals have been tried, whichever comes sooner. The leapfrog jump size ϵ and the unit number of jumps l may be re-drawn at every iteration or for every new proposal. As mentioned earlier, Campos and Sanz-Serna (2015) have proposed extra chance generalized hybrid Monte Carlo (XCGHMC), which is identical to the sequential-proposal approach, except possibly in the way the velocity is refreshed at the end of each iteration. In generalized HMC (Horowitz 1991), the velocity is partially

refreshed by setting

$$V^{(i+1)} = \sin\theta\, V + \cos\theta\, U,$$

where V is the velocity before refreshement, U is an independent draw from $\mathcal{N}(0, C)$, and θ is an arbitrary real number. It was shown in Campos and Sanz-Serna (2015) that Markov chains constructed by XCGHMC have the same law as those constructed by Look Ahead Hamiltonian Monte Carlo (LAHMC) developed by Sohl-Dickstein et al. (2014).

A major advantage of HMC algorithms over random walk based algorithms such as random walk Metropolis or Metropolis adjusted Langevin algorithms is that HMC can make a global jump in one iteration (Neal 2011; Betancourt 2017). The leapfrog method is able to build long trajectories that are numerically stable, provided that the target distribution satisfies some regulatory conditions and the leapfrog step size is less than a certain upper bound (Leimkuhler and Reich 2004). Since the solution to the HEM preserves the Hamiltonian, proposals obtained by a numerical approximation to the solution can be accepted with reasonably high probabilities. Given a fixed length of leapfrog trajectory, the number of leapfrog jumps is inversely proportional to the leapfrog jump size. Thus an increase in the leapfrog step size leads to a reduced number of evaluations of the gradient of the target density. On the other hand, decreasing the leapfrog step size tends to increase the mean acceptance probability. As $\epsilon \to 0$, the average increment in the Hamiltonian at the end of the leapfrog trajectory scales as ϵ^4 (Leimkuhler and Reich 2004). In an asymptotic scenario where the target distribution is given by a product of d independent, identical low dimensional distributions and d tends to infinity, the increment in the Hamiltonian converges in distribution to a normal distribution with mean $\mu\epsilon^4 d$ and variance $2\mu\epsilon^4 d$ for some constant $\mu > 0$ dependent on the target density $\bar{\pi}$ (Gupta et al. 1990; Neal 2011). Beskos et al. (2013) showed under some mild regulatory conditions on the target density that as $\epsilon = \epsilon_0 d^{-1/4}$ and $d \to \infty$, the mean acceptance probability tends to $a(\epsilon_0) := 2\Phi\left(-\epsilon_0^2\sqrt{\mu/2}\right)$ where $\Phi(\cdot)$ denotes the cdf of the standard normal distribution. The computational cost for obtaining an accepted proposal that is fixed distance away from the current state in HMC is approximately given by

$$\frac{1}{\epsilon_0 a(\epsilon_0)},$$

which is minimized when $a(\epsilon_0) = 0.651$ to three decimal places (Beskos et al. 2013; Neal 2011). Empirical results also support targeting the mean acceptance probability of around 0.65 (Sexton and Weingarten 1992; Neal 1994). HMC using sequential proposals can improve on the numerical efficiency by increasing the probability that the constructed Markov chain makes a nonzero move at each iteration. A numerical study in Sect. 4.3 shows that HMC with sequential proposals leads to higher effective sample sizes per computation time compared to the standard HMC on a toy model.

Algorithm 4: The No-U-Turn samplers by Hoffman and Gelman (2014)

Input : Leapfrog step size, ϵ

Output: A draw of Markov chain, $\left(X^{(i)}\right)_{i\in 1:M}$

Initialize: Set $X^{(0)}$ arbitrarily
for $i \leftarrow 0 : M-1$ **do**
  Draw $\Lambda \sim \text{unif}(0, 1)$ and $V \sim \mathcal{N}(0, I_d)$
  Start with an initial tree $T^0 := \{(X^{(i)}, V)\}$ having a single leaf
  for $j \geq 1$ **do**
    Draw $\sigma_j \sim \text{unif}(\{-1, 1\})$
    Make 2^{j-1} leapfrog jumps either forward or backward depending on σ_j, forming a new binary tree T' of the same size as T^{j-1}.
    if *every sub-binary tree of* T' *is such that the two leaves on the opposite sides do not satisfy the U-turn condition* (18) **then**
      Set $T^j \leftarrow T^{j-1} \cup T'$
    else
      `break`
    end
    if *the two opposite leaves of* T^j *satisfies the U-turn condition* **then**
      `break`
    end
  end
  Let T^{j_0} be the final binary tree constructed
  Naive NUTS (Algorithm 2 in Hoffman and Gelman (2014)): Take for $X^{(i+1)}$ one of the leaf nodes of T^{j_0} that are acceptable, i.e., $\frac{\Pi(x,v)}{\Pi(X^{(i)},V)} > \Lambda$, uniformly at random
  Efficient NUTS (Algorithm 3 in Hoffman and Gelman (2014)): Denote by $n_a(T)$ the number of acceptable leaf nodes in a binary tree T, and
  for $j \leftarrow j_0 : 0$ **do**
    With probability $1 \wedge \frac{n_a(T^j \setminus T^{j-1})}{n_a(T^{j-1})}$, take for $X^{(i+1)}$ one of the acceptable leaf nodes of $T^j \setminus T^{j-1}$ uniformly at random, and break out from **for** loop
  end
end

4.2 Sequential-proposal No-U-Turn sampler algorithms

As previously mentioned, a key advantage of HMC over random walk based methods comes from its ability to make long moves. If the number of leapfrog jumps is too small, the Markov chain from HMC may essentially behave like a random walk because the velocity is randomly refreshed before a long leapfrog trajectory is built. Conversely, if the number of leapfrog jumps is too large, the trajectory may double back on itself, since the solution to the Hamiltonian equation of motion is confined to a level set of the Hamiltonian. However, simply stopping the leapfrog jumps when the trajectory starts doubling back on itself generally destroys the detailed balance of the Markov chain with respect to the

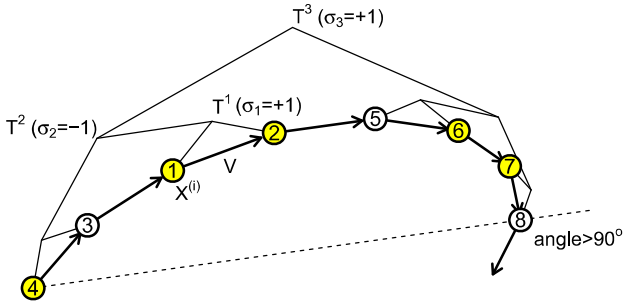


Fig. 1 An example diagram of a final binary tree constructed in an iteration of the NUTS algorithm by Hoffman and Gelman (2014). The numbered circles indicate the points along a leapfrog trajectory in the order they are added. The binary tree stops expanding at T^3 because there is a U-turn between leaf nodes 4 and 8. The next state of the Markov chain is selected randomly among the acceptable states, colored in yellow. (Color figure online)

target distribution. In order to solve this issue, Hoffman and Gelman (2014) proposed the No-U-Turn sampler (NUTS). In this section, we will briefly explain the NUTS algorithm and discuss the connection with sequential-proposal framework. In addition, we will propose two new algorithms that address the same issue of trajectory doubling.

In the No-U-Turn sampler, leapfrog trajectories are repeatedly extended twice in size in either forward or backward direction in the form of binary trees, until a "U-turn" is observed (see Fig. 1). The binary tree starts from the initial node $(X^{(i)}, V)$, where V is the velocity drawn from the standard multivariate normal distribution at the beginning of the i-th iteration. The direction of binary tree expansion is determined by a sequence of unif$(\{-1, 1\})$ variables denoted by $(\sigma_j)_{j\geq 0}$. The expansion of the binary tree stops if a U-turn is observed between the two leaf nodes on opposite sides of any of the sub-binary trees of the current tree. A position-velocity pair (x, v) and another pair (x', v') that is ahead of (x, v) on a leapfrog trajectory are said to satisfy the U-turn condition if either

$$(x' - x) \cdot v' \leq 0 \quad \text{or} \quad (x' - x) \cdot v \leq 0, \tag{18}$$

where $\cdot$ denotes the inner product in Euclidean spaces. If there is a U-turn within the lastly added half of the current binary tree, the other half without a U-turn is taken as the final binary tree. On the other hand, if a U-turn is only observed between the two opposite leaf nodes of the current binary tree but not within any of the sub-binary trees, the current binary tree is taken as the final binary tree. The next state of the Markov chain $X^{(i+1)}$ is set to one of the acceptable leaf nodes in the final binary tree. A leaf node (x, v) is deemed acceptable if $\frac{\Pi(x,v)}{\Pi(X^{(i)},V)} > \Lambda$. Here $\Pi(x, v) := \bar{\pi}(x)\psi_{I_d}(v)$, where ψ_{I_d} denotes the density of the d dimensional standard normal distribution. Hoffman and Gelman (2014) gives two versions of the NUTS algorithm. The naive version selects the next state of the Markov chain uniformly at random among the acceptable leaf nodes in the final binary tree. The efficient version preferentially selects a random leaf node in sub-binary trees that are added later. A pseudocode for these NUTS algorithms is given in Algorithm 4. By construction, for every leaf node in the final binary tree, it is possible to build the same final binary tree starting from that leaf node using a unique sequence of directions. Since each direction is drawn from unif$(\{-1, 1\})$, the probability of constructing the final binary tree is the same when started from any of its leaf nodes. This symmetric relationship ensures that the constructed Markov chain is reversible with respect to the target distribution.

The NUTS algorithm shares with the sequential-proposal MCMC framework the key feature that the decisions of acceptance or rejection of proposals are mutually coupled via a single uniform$(0, 1)$ random variable drawn at the start of each iteration. Furthermore, the naive version of the algorithm (Algorithm 2 in Hoffman and Gelman (2014)) can be viewed as a specific case of the sequential-proposal MCMC algorithm as follows. At each iteration a binary tree starting from $(X^{(i)}, V)$ is expanded until a U-turn is observed, as described above. Proposals are made sequentially by selecting one of the leaf nodes of the final binary tree uniformly at random. The first proposal that is acceptable is taken as the next state of the Markov chain. Since the next state of the Markov chain is then selected uniformly at random among the *acceptable* leaf nodes in the final binary tree, this sequential-proposal approach is equivalent to the naive NUTS.

There are two features of the NUTS algorithm that may, unfortunately, compromise the numerical efficiency. First, the point chosen for the next state of the Markov chain is generally not the farthest point on the leapfrog trajectory from the initial point. The NUTS typically constructs a leapfrog trajectory that is longer than the distance between the initial point and the point selected for the next state of the Markov chain due to the requirement of detailed balance. Second, the NUTS evaluates the log target density at every point on the constructed leapfrog trajectory to determine the acceptability. This can result in a substantial overhead if the computational cost of evaluating the log target density is at least comparable to that of evaluating the gradient of the log target density. We propose two alternative No-U-Turn sampling algorithms, which we call spNUTS1 and spNUTS2, addressing these two issues.

In spNUTS1, leapfrog trajectories are extended in one direction according to a given length schedule until a U-turn is observed, and only the endpoint of the trajectory is checked for acceptability. If the endpoint is not acceptable, a new trajectory is started from that point with a refreshed velocity. A pseudocode of spNUTS1 is given in Algorithm 5. At the start of each iteration, a velocity vector is drawn from a multivariate normal distribution $\mathcal{N}(0, C)$ where C is a $d \times d$

Algorithm 5: Sequential-proposal No-U-Turn sampler—Type 1 (spNUTS1)

Input : Leapfrog step size, ϵ
Unit number of leapfrog jumps, l
Covariance of velocity distribution, C
Scheduled checkpoints for a U-turn, $(b_j)_{j\in 1:\, j_{\max}}$
Distribution for the stopping value of cosine angle, ζ
Maximum number of proposals tried, N

Output: A draw of Markov chain, $\left(X^{(i)}\right)_{i\in 1:M}$

Initialize: Set $X^{(0)}$ arbitrarily
for $i \leftarrow 0 : M-1$ **do**
  Set $Y_0 \leftarrow X^{(i)}$ and draw $W_0 \sim \mathcal{N}(0, C)$
  Set $X^{(i+1)} \leftarrow X^{(i)}$
  Draw $\Lambda \sim$ unif(0, 1) and set $H_{\max} \leftarrow -\log \pi(Y_0) + \frac{1}{2}\|W_0\|_C^2 - \log \Lambda$
  for $n \leftarrow 1 : N$ **do**
    Draw $c \sim \zeta(\cdot)$
    $(Y_n, W_n') \leftarrow$ `spNUTS1Kernel`(Y_{n-1}, W_{n-1}, c)
    if $-\log \pi(Y_n) + \frac{1}{2}\|W_n'\|_C^2 < H_{max}$ **then**
      Set $X^{(i+1)} \leftarrow Y_n$
      `break`
    end
    Draw $U \sim \mathcal{N}(0, C)$ and set $W_n \leftarrow U \cdot \frac{\|W_n'\|_C}{\|U\|_C}$
  end
end
Function `spNUTS1Kernel`(x_0, v_0, c)
  for $k \leftarrow 1 : b_1$ **do**
    $(x_k, v_k) \leftarrow$ `Leapfrog`$(x_{k-1}, v_{k-1}, \epsilon, l, C)$
  end
  Set $j \leftarrow 1$
  while *cosAngle*$(x_{b_j} - x_0, v_0 ; C) > c$ *and* *cosAngle*$(x_{b_j} - x_0, v_{b_j} ; C) > c$ *and* $j < j_{max}$ **do**
    Set $j \leftarrow j + 1$
    for $k \leftarrow b_{j-1}+1 : b_j$ **do**
      $(x_k, v_k) \leftarrow$ `Leapfrog`$(x_{k-1}, v_{k-1}, \epsilon, l, C)$
    end
  end
  if *cosAngle*$(x_{b_j} - x_{b_j - b_{j'}}, v_{b_j} ; C) > c$ *and* *cosAngle*$(x_{b_j} - x_{b_j - b_{j'}}, v_{b_j - b_{j'}} ; C) > c$ *for all* $j' \in 1 : j-1$ **then**
    `return` (x_{b_j}, v_{b_j})
  else
    `return` (x_0, v_0)
  end
end

positive definite matrix. A leapfrog trajectory started from the current state of the Markov chain and the drawn velocity vector, denoted by (x_0, v_0), is repeatedly extended in units of l jumps. The position-velocity pair after lk leapfrog jumps is denoted by (x_k, v_k). We note that the leapfrog updates (17) should also use the same matrix C as the covariance of the velocity distribution. At preset checkpoints determined by a finite increasing sequence $(b_j)_{j\in 1:j_{\max}}$, the algorithm calculates the angles between the displacement $x_{b_j} - x_0$ and the velocities v_0 and v_{b_j}. In order to take into account the given covariance structure C, we define a C-norm of a vector $x \in \mathbb{R}^d$ as

$$\|x\|_C := \sqrt{x^T C^{-1} x},$$

and the cosine of the angle between two vectors x and x' as

$$\text{cosAngle}(x, x' ; C) := \frac{x^T C^{-1} x'}{\|x\|_C \cdot \|x'\|_C}. \tag{19}$$

The leapfrog trajectory stops at (x_{b_j}, v_{b_j}) if either of the following inequalities hold for a given c:

$$\begin{aligned} &\text{cosAngle}(x_{b_j} - x_0, v_0 ; C) \le c \quad \text{or} \\ &\quad \text{cosAngle}(x_{b_j} - x_0, v_{b_j} ; C) \le c. \end{aligned} \tag{20}$$

The value of c can be fixed at a constant value or randomly drawn for each trajectory. Algorithm 5 describes a case where c is randomly drawn from a distribution denoted by ζ. If the above stopping condition (20) is not satisfied until $j = j_{\max}$, the trajectory stops at $(x_{b_{j_{\max}}}, v_{b_{j_{\max}}})$.

The final state at the stopped trajectory (x_{b_j}, v_{b_j}) makes the first proposal (Y_1, W_1') (Fig. 2). It is taken as the next state in the Markov chain if the following two conditions are met. First, the state (x_{b_j}, v_{b_j}) has to be acceptable by satisfying

$$\begin{aligned} \log \Lambda < &\log \pi(x_{b_j}) + \log \psi_C(v_{b_j}) - \log \pi(x_0) \\ &- \log \psi_C(v_0). \end{aligned} \tag{21}$$

Since the Hamiltonian of the state (x_{b_j}, v_{b_j}) is given by $-\log \bar{\pi}(x_{b_j}) - \log \psi_C(v_{b_j})$, the acceptability criterion (21) can be interpreted as the increase in the Hamiltonian compared to the initial state (x_0, v_0) being at most $-\log \Lambda$. The second required condition is that

$$\begin{aligned} &\text{cosAngle}(x_{b_j} - x_{b_j - b_{j'}}, v_{b_j} ; C) > c \ \text{ and} \\ &\quad \text{cosAngle}(x_{b_j} - x_{b_j - b_{j'}}, v_{b_j - b_{j'}} ; C) > c \\ &\quad \text{for all } 1 \le j' \le j - 1. \end{aligned} \tag{22}$$

Since the trajectory has been extended to (x_{b_j}, v_{b_j}), the stopping condition (20) was not satisfied between the initial state (x_0, v_0) and any of the previously visited states $\{(x_{b_{j'}}, v_{b_{j'}}) ; 1 \le j' < j\}$. When the trajectory is viewed in the reverse order, an analogous situation is that the final state (x_{b_j}, v_{b_j}) and the intermediate states $\{(x_{b_j - b_{j'}}, v_{b_j - b_{j'}}) ; 1 \le j' < j\}$ satisfy (22). This symmetry condition is necessary to establish detailed balance of the Markov chain. If the symmetry condition is not satisfied, the next state of the Markov chain is set to the current state x_0. In the case where both the acceptability condition (21) and the symmetry condition (22) are satisfied, x_{b_j} is taken as the next state of the Markov chain. If the symmetry condition is satisfied but the acceptability condition is not, the algorithm starts a new leapfrog trajectory from x_{b_j} with a new velocity W_1. The new velocity W_1 is obtained by drawing a random vector from the velocity distribution ψ_C and rescaling it such that the C-norm is preserved:

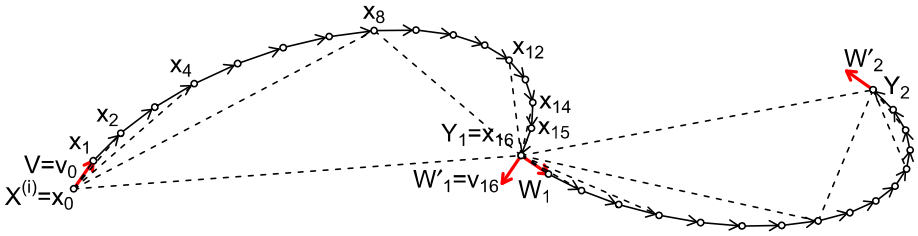


Fig. 2 An example diagram of an iteration in spNUTS1 where $b_j = 2^{j-1}$. The first proposal $(Y_1, W_1') = (x_{16}, v_{16})$ was rejected, and the second trajectory was started with a refreshed velocity W_1. The pairs of points for which the U-turn condition is checked are connected by dashed line segments

$\|W_1\|_C = \|W_1'\|_C$, or $\log \psi_C(W_1) = \log \psi_C(W_1')$. The new trajectory is extended until the stopping condition (20) with respect to the starting pair (Y_1, W_1) is satisfied. This procedure is repeated for subsequent proposals (Y_n, W_n'), $n \geq 2$. If all N proposals $\{(Y_n, W_n') ; n \in 1:N\}$ are unacceptable, the next state of the Markov chain is set to the current state $X^{(i)}$. The Markov chain also stays at its current state if the symmetry condition is not satisfied by at least one of the constructed leapfrog trajectories.

Algorithm 5 can be considered as a special case of sequential-proposal MCMC algorithms using deterministic kernels (Algorithm 2) where $L = 1$ and the proposal map $\mathcal{S}_\tau$ is given by a function that maps the starting state (x_0, v_0) to the final state (x_{b_j}, v_{b_j}) of the stopped trajectory, but with a differing feature that the direction of the velocity is randomly refreshed between proposals. In practice, extending the trajectory length at an exponential rate by setting, for example, $b_j = 2^{j-1}$ can lead to a high probability that the symmetry condition is satisfied, because the length of the interval $[b_{j-1}+1, b_j]$ in which the first U-turn occurs also increases exponentially. Hoffman and Gelman (2014) also remarked (based on Fig. 5 in their paper) that as the size of binary trees were repeatedly doubled, the U-turn condition was satisfied most of the time only by the two opposite leaf nodes of the final binary tree but not by the opposite leaf nodes of any of the sub-binary trees, for all examples they considered including ones arising from practical applications. The choice $b_j = 2^{j-1}$ also makes it easy to predict the checkpoints for the symmetry condition, $\{b_j - b_{j'} ; j' \leq j-1\}$. We note that the numerical efficiency of Algorithm 5, as well as that of the original NUTS algorithm, can be improved by tuning the covariance C (see the numerical results in Sect. 4.3).

The computational efficiency of Algorithm 5 may be compared favorably to that of the NUTS algorithm. Some numerical results are given in Sect. 4.3. Suppose that the average computational cost of evaluating the log target density is denoted by c_π and that of evaluating the gradient of the log target density by c_∇. Assume also that in the NUTS and spNUTS1 (Algorithm 5), the computational cost of checking the U-turn condition such as (20) is denoted by c_U. If a leapfrog trajectory stops after making ξ sets of l unit jumps on average, the NUTS algorithm evaluates the log target density ξ times and checks the U-turn condition ξ times on average. Thus the average computational cost for one iteration of the NUTS algorithm is given by $(lc_\nabla + c_\pi + c_U) \cdot \xi$. In comparison, spNUTS1 evaluates the log target density once and checks the U-turn condition $2\log_2 \xi + 1$ times if $b_j = 2^{j-1}$ for $j \in 1 : j_{\max}$. The average computational cost of obtaining a proposal in spNUTS1 is given by $lc_\nabla \xi + c_\pi + c_U(2\log_2 \xi + 1)$, and the average cost of finding a new state for the Markov chain different from the current state is roughly given by $\frac{1}{a \cdot \tilde{a}}\big(lc_\nabla \xi + c_\pi + c_U(2\log_2 \xi + 1)\big)$, where a denotes the mean acceptance probability of a proposal and $\tilde{a}$ denotes the average probability that the symmetry condition is satisfied. Both a and $\tilde{a}$ can be made close to unity in practice, so there is a computational gain in using spNUTS1 over the original NUTS if ξ is large and c_π is at least comparable to c_∇. The number l can be chosen to one unless there is an issue of numerical instability of leapfrog trajectories. We note that the increase in the cost by a factor of $\frac{1}{a}$ can be partially negated, in terms of the overall numerical efficiency, due to the fact if a proposal is deemed unacceptable, the next proposal can be further away from the initial state Y_0. The average distance between two consecutive states in the constructed Markov chain is a measure widely used to evaluate the numerical efficiency of an MCMC algorithm (Sherlock et al. 2010).

The proof of the following proposition is given in Online Resource S5.

Proposition 3 *The Markov chain* $\left(X^{(i)}\right)_{i \in 1:M}$ *constructed by the sequential-proposal No-U-Turn sampler of type 1 (spNUTS1, Algorithm 5) is reversible with respect to the target density* $\bar{\pi}$.

Another algorithm that automatically tunes the lengths of leapfrog trajectories, called spNUTS2, is given in Algorithm 6. Unlike spNUTS1, spNUTS2 applies the sequential-proposal scheme within one trajectory. The spNUTS2 algorithm takes the endpoint of the constructed leapfrog trajectory as a candidate for the next state of the Markov chain, as in spNUTS1. However, it evaluates the log target density at every point on the trajectory like the original NUTS. Starting

Algorithm 6: Sequential-proposal No-U-Turn sampler—Type 2 (spNUTS2)

Input : Leapfrog step size, ϵ
Unit number of leapfrog jumps, l
Covariance of velocity distribution, C
Maximum number of proposals, N
Scheduled checkpoints for a U-turn, $(b_j)_{j\in 1:\, j_{\max}}$
Distribution for the stopping value of cosine angle, ζ

Output: A draw of Markov chain, $\left(X^{(i)}\right)_{i\in 1:M}$

Initialize: Set $X^{(0)}$ arbitrarily
for $i \leftarrow 0 : M-1$ **do**
  Draw $V \sim \mathcal{N}(0, C)$
  Draw $c \sim \zeta(\cdot)$
  Draw $\Lambda \sim$ unif(0, 1) and set $\Delta \leftarrow -\log \Lambda$
  Set $\left(X^{(i+1)}, V^{(i+1)}\right) \leftarrow$ `spNUTS2Kernel`$(X^{(i)}, V, \Delta, \epsilon, C, c)$
end
Function `spNUTS2Kernel`$(x_0, v_0, \Delta, \epsilon, C, c)$
  Set $H_{\max} \leftarrow -\log \pi(x_0) + \frac{1}{2}\|v_0\|_C^2 + \Delta$
  for $k \leftarrow 1 : b_1$ **do**
    $(x_k, v_k, \mathbf{f}) \leftarrow$ `FindNextAcceptable`$(x_{k-1}, v_{k-1}, \epsilon, H_{\max}, C)$
    if $\mathbf{f} = 0$ **then** `return` (x_0, v_0) // the case where no acceptable states were found
  end
  Set $j \leftarrow 1$
  while *cosAngle*$(x_{b_j} - x_0, v_0\,; C) > c$ *and* *cosAngle*$(x_{b_j} - x_0, v_{b_j}\,; C) > c$ *and* $j < j_{max}$ **do**
    Set $j \leftarrow j + 1$
    for $k \leftarrow b_{j-1}+1 : b_j$ **do**
      $(x_k, v_k, \mathbf{f}) \leftarrow$ `FindNextAcceptable`$(x_{k-1}, v_{k-1}, \epsilon, H_{\max}, C)$
      if $\mathbf{f} = 0$ **then** `return` (x_0, v_0)
    end
  end
  if *cosAngle*$(x_{b_j} - x_{b_j - b_{j'}}, v_{b_j}\,; C) > c$ *and* *cosAngle*$(x_{b_j} - x_{b_j - b_{j'}}, v_{b_j - b_{j'}}\,; C) > c$ *for all* $j' \in 1 : j-1$ **then**
    `return` (x_{b_j}, v_{b_j})
  else
    `return` (x_0, v_0)
  end
end
Function `FindNextAcceptable`$(x, v, \epsilon, H_{\max}, C)$
  Set $(x_{\text{try}}, v_{\text{try}}) \leftarrow (x, v)$
  for $n \leftarrow 1 : N$ **do**
    $(x_{\text{try}}, v_{\text{try}}) \leftarrow$ `Leapfrog`$(x_{\text{try}}, v_{\text{try}}, \epsilon, l, C)$
    if $-\log \pi(x_{\text{try}}) + \frac{1}{2}\|v_{\text{try}}\|_C^2 < H_{\max}$ **then** `return` $(x_{\text{try}}, v_{\text{try}}, 1)$
  end
  `return` $(x, v, 0)$
end

from the current state of the Markov chain $X^{(i)} = x_0$ and a velocity vector v_0 randomly drawn from ψ_C, the algorithm extends a leapfrog trajectory in units of l leapfrog jumps. We will denote by (x_1, v_1) the first *acceptable* state along the trajectory that is a multiple of l leapfrog jumps away from the initial state (Fig. 3). Here, (x_1, v_1) is acceptable if

$$\Lambda < \frac{\pi(x_1)\psi_C(v_1)}{\pi(x_0)\psi_C(v_0)}.$$

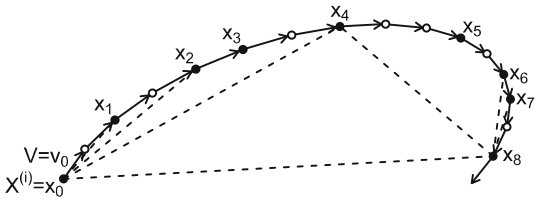


Fig. 3 An example diagram of an iteration in spNUTS2 where $b_j = 2^{j-1}$. Acceptable states are marked by filled circles and unacceptable ones by empty circles. The pairs of states for which the U-turn condition is checked are indicated by dashed line segments. The eighth acceptable state x_8 is taken as the next state of the Markov chain

In order to avoid indefinitely extending the trajectory when the leapfrog approximation is numerically unstable, the algorithm ends the attempt to find the next acceptable state if N consecutive states at intervals of l leapfrog jumps are all unacceptable. In this case, the next state of the Markov chain is set to (x_0, v_0). For $k \geq 2$, the state (x_k, v_k) is likewise found as the first acceptable state along the leapfrog trajectory that is a multiple of l jumps from (x_{k-1}, v_{k-1}). If for any $k \geq 1$ the next acceptable state is not found in N consecutive states visited after (x_{k-1}, v_{k-1}), the next state in the Markov chain is also set to (x_0, v_0). In practice, however, this situation can be avoided by taking the leapfrog step size ϵ reasonably small to ensure numerical stability and N large enough. The algorithm takes a preset increasing sequence of integers $(b_j)_{j\in 1:\, j_{\max}}$ and checks if the angles between the displacement vector $x_{b_j} - x_0$ and the initial and the last velocity vectors v_0 and v_{b_j} are below a certain level c. The trajectory is stopped at (x_{b_j}, v_{b_j}) if either

$$\begin{aligned}&\text{cosAngle}(x_{b_j} - x_0, v_0\,; C) \leq c \quad \text{or}\\ &\quad \text{cosAngle}(x_{b_j} - x_0, v_{b_j}\,; C) \leq c.\end{aligned} \tag{23}$$

Upon reaching $(x_{b_{j_{\max}}}, v_{b_{j_{\max}}})$, however, the trajectory stops regardless of whether (23) is satisfied for $j = j_{\max}$. As in spNUTS1, a symmetry condition is checked to ensure detailed balance. That is, the state (x_{b_j}, v_{b_j}) is taken as the next state in the Markov chain if and only if

$$\begin{aligned}&\text{cosAngle}(x_{b_j} - x_{b_j - b_{j'}}, v_{b_j}\,; C) > c \ \text{ and}\\ &\quad \text{cosAngle}(x_{b_j} - x_{b_j - b_{j'}}, v_{b_j - b_{j'}}\,; C) > c\\ &\quad \text{for all } 1 \leq j' \leq j - 1.\end{aligned} \tag{24}$$

If the symmetry condition is not satisfied, the next state of the Markov chain is set to (x_0, v_0). As in spNUTS1, the choice of $b_j = 2^{j-1}$, $j \in 1 : j_{\max}$, allows the symmetry condition in spNUTS2 to be satisfied with high probability and makes the checkpoints for the symmetry condition, $\{b_j - b_{j'}\,;\, j' \leq j - 1\}$, readily predictable.

When c_π, c_∇, and c_U denote the same average computational costs as before and $b_j = 2^{j-1}$, the average computa-

tional cost of finding a distinct sample point for the Markov chain using spNUTS2 is roughly given by $\frac{1}{\tilde{a}}\{(lc_\nabla + c_\pi)\cdot\xi + c_U(2\log_2(a\xi)+1)\}$, where ξ denotes the average length of stopped trajectories in units of l leapfrog jumps, $\tilde{a}$ the average probability that the symmetry condition is satisfied, and a the mean acceptance probability. Since $\tilde{a}$ can be close to unity and c_U is often smaller than c_π or c_∇ in practice, the computational cost of spNUTS2 per distinct sample is comparable to that of the NUTS. However, the overall numerical efficiency of spNUTS2 can be higher because the average distance between the current and the next state of the Markov chain can be larger.

The proof of the following proposition is also given in Online Resource S5.

Proposition 4 *The Markov chain* $\left(X^{(i)}\right)_{i\in 1:M}$ *constructed by the sequential-proposal No-U-Turn sampler of type 2 (spNUTS2, Algorithm 6) is reversible with respect to the target density* $\bar{\pi}$.

4.3 Numerical examples

4.3.1 Multivariate normal distribution

We used two examples to study the numerical efficiency of various algorithms discussed in this paper. We first considered a one hundred dimensional normal distribution $\mathcal{N}(0, \Sigma)$ where the covariance matrix Σ is diagonal and the marginal standard deviations form a uniformly increasing sequence from 0.01 to 1.00.

We compared the numerical efficiency of the following five algorithms: the standard HMC, HMC with sequential proposals (abbreviated as spHMC, which is equivalent to XCGHMC in Algorithm 3), the NUTS algorithm by Hoffman and Gelman (2014) (the efficient version), spNUTS1 (Algorithm 5), and spNUTS2 (Algorithm 6). All experiments were carried out using the implementation of the algorithms in R (R Core Team 2018). The source codes are available at https://github.com/joonhap/spMCMC. The covariance matrix C of the velocity distribution was set equal to the one hundred dimensional identity matrix. The leapfrog step size at the i-th iteration ϵ_i was adaptively tuned using the formula

$$\log\epsilon_{i+1} \leftarrow \log\epsilon_i + \frac{\lambda}{i^{0.7}}(a_i - a^*)$$

where the target acceptance probability a^* varied from 0.45 to 0.95. The acceptance probability at the i-th iteration a_i was computed using the state that was one leapfrog jump away from the current state of the Markov chain to ensure that the leapfrog jump size ϵ converges to the same value for the same target acceptance probability across the various algorithms. More details on adaptive tuning can be found in Online Resource S6. The adaptation started from the one hundredth iteration. When running HMC, spHMC, spNUTS1, and spNUTS2, the leapfrog step size was randomly perturbed at each iteration by multiplying to ϵ_i a uniform random number between 0.8 and 1.2. Randomly perturbing the leapfrog step size can improve the mixing of the Markov chain constructed by HMC algorithms (Neal 2011). For the NUTS, we found perturbing the leapfrog step size did not improve numerical efficiency and thus used ϵ_i. In HMC and spHMC, each proposal was obtained by making fifty leapfrog jumps. In spHMC, a maximum of $N = 10$ proposals were tried in each iteration and the first acceptable proposal was taken as the next state of the Markov chain (i.e., $L = 1$). In spNUTS1 and spNUTS2, the stopping condition was checked according to the schedule $b_j = 2^{j-1}$ for $j \in 1 : j_{\max}$ with $j_{\max} = 15$, and the unit number of leapfrog trajectories was set to one (i.e., $l = 1$ in Algorithms 5 and 6). The value c in the stopping condition (23) was randomly drawn from a uniform$(0, 1)$ distribution for each trajectory, as we found randomizing c yielded better numerical results than fixing it at zero. For the NUTS algorithm, randomizing c did not improve the numerical efficiency, so each trajectory was stopped when the cosine angle fell below zero (i.e., $c = 0$), as was in Hoffman and Gelman (2014). In spNUTS1, the maximum number of proposals N in each iteration was set to either one or five. In spNUTS2, a maximum of $N = 20$ consecutive states on a leapfrog trajectory were tried in each attempt to find an acceptable state. Every algorithm ran for $M = 20{,}200$ iterations. As measures of numerical efficiency, the effective sample size (ESS) of each component of the Markov chain was computed in two ways: the `effectiveSize` function in R package `coda` produced an estimate based on the spectral density at frequency zero of an autoregressive fit to the time series (Plummer et al. 2006), and the `ess` function in R package `mcmcse` produced an estimate based on a batch-mean approach (Flegal et al. 2017). The first two hundred states of the Markov chains were discarded when computing the effective sample sizes. Each experiment was independently repeated ten times. All computations were carried out using the Boston University Shared Computing Cluster.

Figure 4 shows both the minimum and the average effective sample size for the one hundred variables divided by the runtime in seconds when the covariance C of the velocity distribution was fixed at the identity matrix. We observed that there were large variations in the effective sample sizes among the $d = 100$ variables for the Markov chains constructed by HMC and spHMC, resulting in minimum ESSs much smaller than average ESSs. This happened due to the fact that for some variables the leapfrog trajectories with fifty jumps consistently tended to return to states close to the initial positions. The Markov chains mixed slowly in these variables. On the other hand, the leapfrog trajectories tended to reach the opposite side of the level set of Hamiltonian for

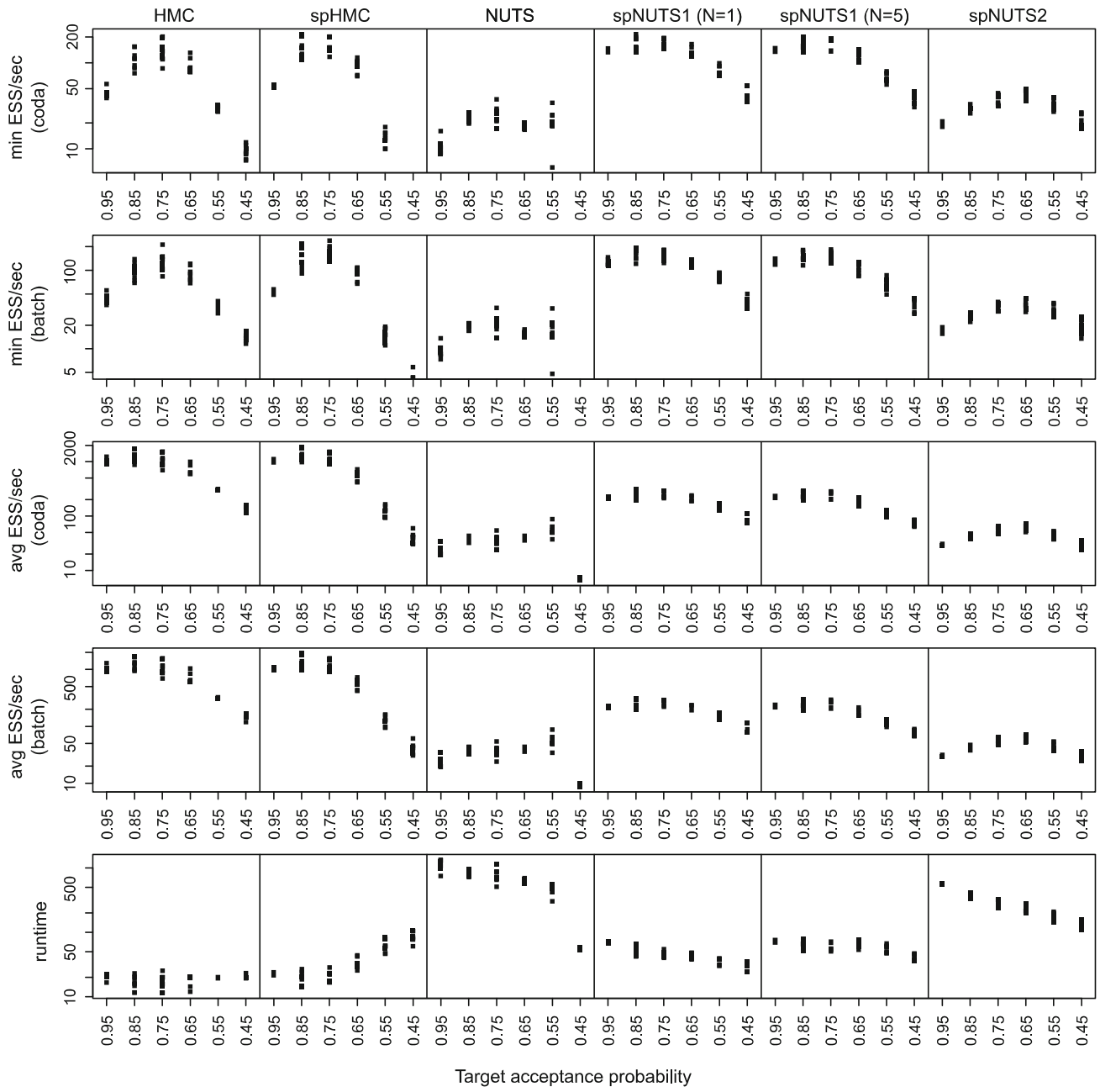


Fig. 4 The minimum and average effective sample sizes of constructed Markov chains across $d = 100$ variables per second of runtime for the target distribution $\mathcal{N}(0, \Sigma)$. The target acceptance probabilities are shown on the x-axis. The runtime in seconds are shown in the bottom row of plots. All y-axes are in logarithmic scales

some variables, for which the autocorrelation at lag one was close to -1. For these variables, the effective sample size was greater than the length of the Markov chain M. There were much variations in the effective sample size among the variables for the Markov chains constructed by spNUTS1 and spNUTS2 when the stopping cosine angle c was fixed at zero, but variations diminished when c was varied uniformly in the interval $(0, 1)$.

The highest value of the minimum ESS per second achieved by spHMC among various values of the target acceptance probability was higher than that by the standard HMC by about 7–12%. For this multivariate normal distribution, the number of leapfrog jumps $l = 50$ for HMC and spHMC was within the range of the average number of jumps in the leapfrog trajectories constructed by the NUTS, spNUTS1, and spNUTS2 algorithms. Thus the effective sam-

ple sizes by HMC and spHMC were comparable to those by the other three algorithms, but the runtimes tended to be shorter. The highest minimum ESS per second by spNUTS1 with $N = 5$ was 5.3 times higher than that by the NUTS. The highest minimum ESS per second by spNUTS2 was 32% higher than that by the NUTS. The runtimes of the NUTS were more than ten times longer than those of spNUTS1 and twice longer than those of spNUTS2. This happened because the evaluation of the gradient of the log target density took much less computation time than the evaluation of the log target density for this example.

Next we ran the NUTS, spNUTS1, and spNUTS2 algorithms for the same target distribution $\mathcal{N}(0, \Sigma)$ but with adaptively tuning the covariance C of the velocity distribution. The covariance C_i at the i-th iteration for $i \geq 100$ was set to a diagonal matrix whose diagonal entries were given by the marginal sample variances of the Markov chain constructed up to that point. We did not test HMC or spHMC when C was adaptively tuned, because the leapfrog step size, and thus the total length of the leapfrog trajectory with a fixed number of jumps, varied depending on the tuned values for C. Figure 5 shows the minimum and average ESS computed by the `coda` package divided by the runtime in seconds. The highest minimum ESS per second improved more than fifty times, compared to when the covariance C was fixed, for the NUTS. There was more than a five-fold improvement for spNUTS1, and more than a 25-fold improvement for spNUTS2. The highest minimum ESS per second by the NUTS was 19% higher than that by spNUTS1 ($N = 5$) and 86% higher than that by spNUTS2. The NUTS was relatively more efficient when C was adaptively tuned because the trajectories were built using fewer leapfrog jumps. The computational advantage of spNUTS1 that the log target density is not evaluated at every leapfrog jump is relatively small when there are only few jumps per trajectory. When C is close to Σ, the sampling task is essentially equivalent to sampling from the standard normal distribution, in which case larger leapfrog step sizes may be used. The trajectories were made of five to eight leapfrog jumps at the most efficient target acceptance probability when C was adaptively tuned. In comparison, the number of leapfrog jumps in a trajectory was between 80 and 250 when C was not adaptively tuned.

4.3.2 Bayesian logistic regression model

We also experimented the numerical efficiency of the NUTS, spNUTS1, and spNUTS2 using the posterior distribution for a Bayesian logistic regression model. The Bayesian logistic regression model and the data we used are identical to those considered by Hoffman and Gelman (2014). The German credit dataset from the UCI repository (Dua and Graff 2017) consists of twenty four attributes of individuals and one binary variable classifying those individuals' credit. The posterior density is proportional to

$$\pi(\alpha, \beta \mid x, y) \propto \exp\left\{-\sum_{i=1}^{1000} \log(1 + \exp\{-y_i(\alpha + x_i \cdot \beta\}) - \frac{\alpha^2}{200} - \frac{\|\beta\|_2^2}{200}\right\},$$

where x_i denotes the twenty four dimensional covariate vector for the i-th individual and y_i denotes the classification result taking a value from ± 1. We did not normalize the covariates to zero mean and unit variance as in Hoffman and Gelman (2014), because we let C be adaptively tuned. The covariance C was set to a diagonal matrix having as its diagonal entries the marginal sample variances of the constructed Markov chain up to the previous iteration. All algorithms were run under the same settings as those used for the multivariate normal distribution example.

Figure 6 shows the minimum and average ESS across $d = 25$ variables per second of runtime. The minimum ESS per second by spNUTS1 at the most efficient target acceptance probability was 2.6 times higher than that by the NUTS and 1.7 times higher than that by spNUTS2. The differences in the numerical efficiency was led mostly by the differences in the runtime. The numbers of leapfrog jumps in stopped trajectories tended to be larger than those for the normal distribution example due to the correlations between the variables in this Bayesian logistic regression model; the numbers of leapfrog jumps were about fifty for the NUTS, twenty seven for spNUTS1, and twenty two for spNUTS2.

5 Connection to the bouncy particle sampler

Recently, a non-reversible, piecewise deterministic MCMC sampling method called the bouncy particle sampler (BPS) has been proposed (Peters et al. 2012; Bouchard-Côté et al. 2018). The BPS constructs a rejection free, continuous time Markov chain. A key advantage of the BPS is that it allows for local updates of the target variables, meaning that the algorithm can update one subset of the target variables at a time while the rest of the variables evolve according to a flow that is easy to compute. However, this algorithm has a limitation in terms of the target distributions it can be used for, because the user needs to be able to draw the arrival times of a non-homogeneous Poisson process which have a rate depending on the gradient of the target density. In this section, we present a new, discrete time version of BPS, which is not rejection free. This discrete time BPS is readily applicable to any target distribution with evaluable unnormalized density. We note that there exists an alternative discrete time BPS, which was given in Vanetti et al. (2017, Algorithm 4).

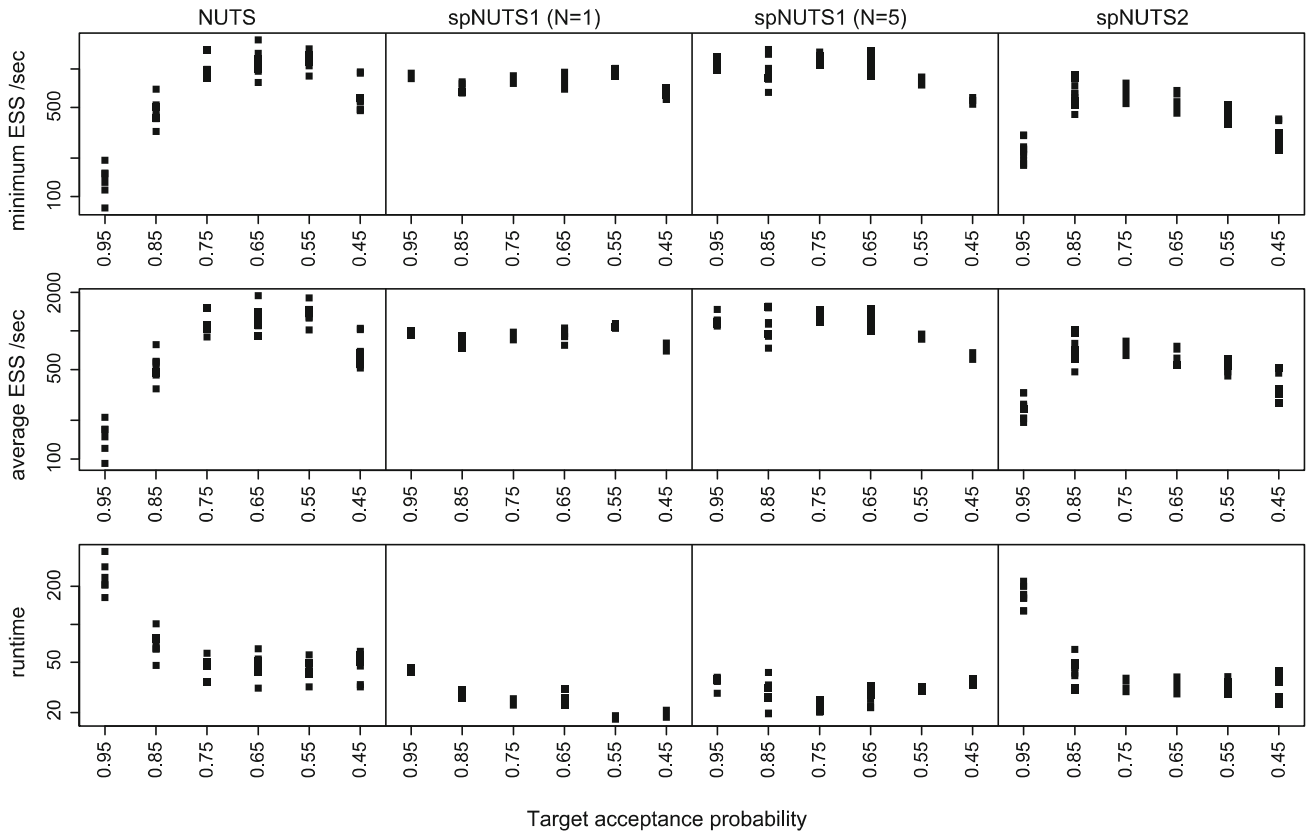


Fig. 5 The minimum and average effective sample sizes per second of runtime for the target distribution $\mathcal{N}(0, \Sigma)$ when the covariance C of the velocity distribution was adaptively tuned. The target acceptance probabilities are shown on the x-axis

We describe the discrete time BPS algorithm we propose within the framework of MCMC algorithms using deterministic kernels (see Sect. 3). We assume that the sample space $\mathbb{X}$ and the velocity space $\mathbb{V}$ are both given by $\mathbb{R}^d$. As in Sect. 3, the density of velocity distribution is denoted by $\psi(v\,;x)$. We suppose that for each $x \in \mathbb{X}$, there exists a *linear* operator $\mathcal{R}_x : \mathbb{V} \to \mathbb{V}$, which satisfies $\mathcal{R}_x \circ \mathcal{R}_x = \mathcal{I}$ and $\psi(\mathcal{R}_x v\,;x) = \psi(v\,;x)$. The time evolution map $\mathcal{S}_\tau$ is defined as

$$\mathcal{S}_\tau(x, v) := (x - \mathcal{R}_x v\tau, -\mathcal{R}_x v).$$

The self-inverse property of the linear operator $\mathcal{R}_x$ ensures that the absolute determinant of $\mathcal{R}_x$ when viewed as a matrix is equal to unity. This translates into unit Jacobian determinant of the map $\mathcal{R}_x$, so the condition (12), $\frac{\psi(\mathcal{R}_x v\,;x)}{\psi(v\,;x)}\left|\frac{\partial \mathcal{R}_x v}{\partial v}\right| = 1$, is satisfied. It can also be readily checked that the condition (13) is satisfied for $\mathcal{S}_\tau$ and that

$$\left|\frac{\partial \mathcal{S}_\tau(x, v)}{\partial(x, v)}\right| = 1, \quad \forall (x, v) \in \mathbb{X} \times \mathbb{V}.$$

A sequential-proposal discrete time BPS can be obtained as a specific case of Algorithm 2 where $\mathcal{S}_\tau$ and $\mathcal{R}_x$ are given as above. Algorithm 7 gives a pseudocode for the sequential-proposal discrete time BPS.

A convenient choice for ψ is a multivariate Gaussian density

$$\psi(v\,;x) = \psi_C(v) := \frac{1}{\sqrt{2\pi}^d\,|\det C|^{1/2}} \exp\{-v^T C^{-1} v\},$$

where C is a positive definite matrix. In this case, the conditions (11) and (12) hold if and only if

$$\mathcal{R}_x = C^{1/2}(I - 2P)C^{-1/2}$$

for a symmetric projection matrix P, that is $PP = P$ and $P^T = P$. A matrix P is a symmetric projection matrix in $\mathbb{R}^d$ if and only if it is a projection onto the linear span of a subset of an orthonormal basis of $\mathbb{R}^d$, that is $P = \sum_{j \in A} e^j (e^j)^T$ for some $A \subseteq \{1, 2, \ldots, d\}$ and some orthonormal basis $(e^1, \ldots, e^d)$.

A possible choice for $\mathcal{R}_x$ includes $-\mathcal{I}$, in which case the proposal map is given by $\mathcal{S}_\tau(x, v) = (x + v\tau, v)$. Since the map $\mathcal{S}_\tau$ can be readily evaluated, this choice has a computational advantage when multiple sequential proposals are made. Another sensible choice for the operator $\mathcal{R}_x$ is the

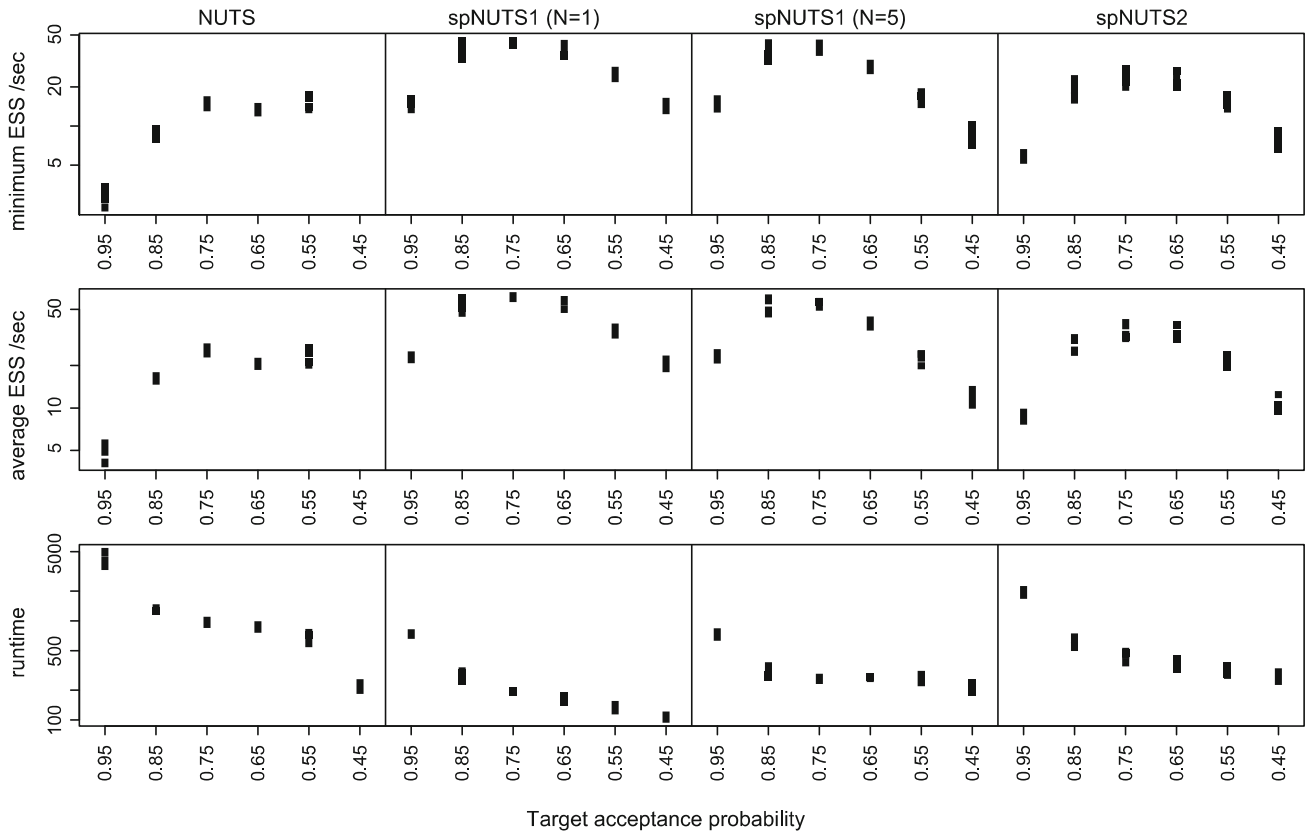


Fig. 6 The minimum and average effective sample sizes per second of runtime across $d = 25$ variables for the posterior distribution for the Bayesian logistic regression model in Sect. 4.3.2

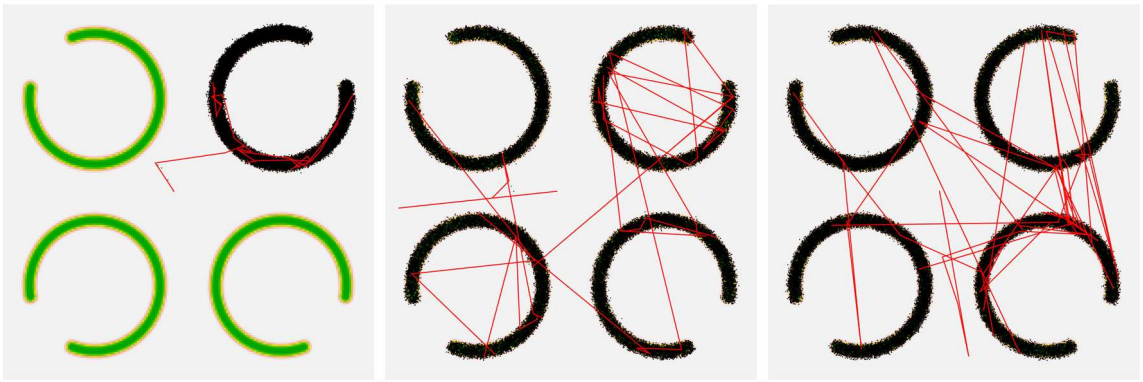


Fig. 7 Numerical results of the sequential-proposal discrete time BPS (Algorithm 7) for the model with four "C"s. The first 120,000 states in the constructed Markov chain are shown as black dots. The trajectory connecting every fourth point is shown by red segments up to one hundred points. Left, $N = 1$; middle, $N = 10$; right, $N = 20$

reflection on the hyperplane perpendicular to the gradient of the log target density under the metric given by C^{-1}. We write $U(x) := -\log \pi(x)$ and denote the velocity reflection operator by $R_{\nabla U(x)}$. This velocity reflection operator can be written as

$$R_{\nabla U(x)} v := v - 2\frac{\langle \nabla U(x), v\rangle_C}{\|\nabla U(x)\|_C^2}\nabla U(x), \tag{25}$$

where $\langle u, w\rangle_C := u^T C^{-1} v$. This $R_{\nabla U(x)}$ is the reflection operator used by the original BPS algorithm of Peters et al. (2012). Since

$$\mathcal{S}_\tau^2(x, v) = (x - \mathcal{R}_x v\tau + \mathcal{R}_{x-\mathcal{R}_x v}\mathcal{R}_x v\tau,\ \mathcal{R}_{x-\mathcal{R}_x v}\mathcal{R}_x v),$$

we have, in the case where τ is small such that $\nabla U(x - \mathcal{R}_x v\tau) \approx \nabla U(x)$ and $\mathcal{R}_{x-\mathcal{R}_x v} \approx \mathcal{R}_x$,

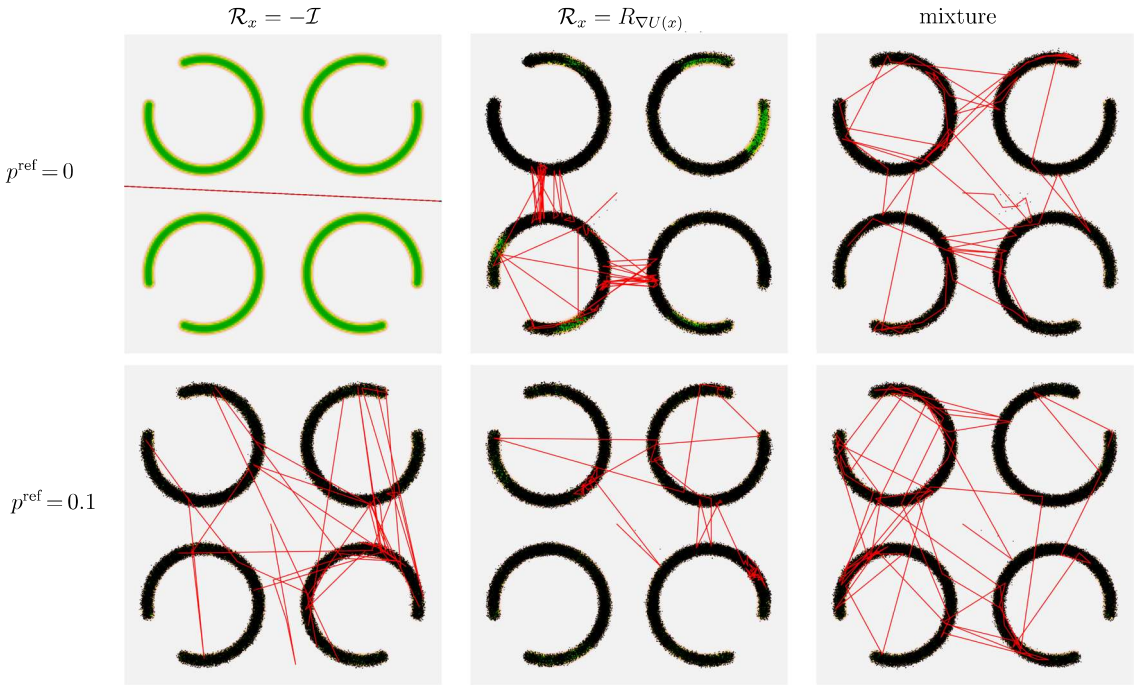
$$\mathcal{S}_\tau^2(x, v) \approx \left(x + 2\frac{\langle \nabla U(x), v\rangle_C}{\|\nabla U(x)\|_C^2} \nabla U(x),\ v \right).$$

Therefore, repeated application of the map $\mathcal{S}_\tau$ has an approximate net effect of moving the particles along the gradient of the log target density.

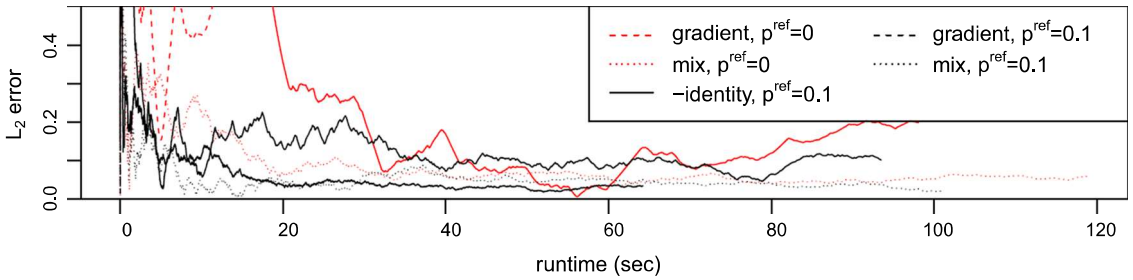
5.1 Numerical examples

To graphically illustrate the performance of the sequential-proposal discrete time BPS (Algorithm 7), we created a target distribution defined on a unit square. The regions of high likelihood density look like four open rings, or four rotated letters of "C", as shown in Figure 7. We applied the sequential-proposal discrete time BPS on this model with varying algorithmic parameters. In every experiment, we ran the algorithm up to 120,000 iterations, where the number of acceptable proposals L was fixed at one and the jump size τ at each iteration varied uniformly between 0.08 and 0.12.

Figure 7 shows the 120,000 sampled points as black dots. The target density is represented by a color map on a green-white scale at the background. Starting from the initial point, the trajectory of every fourth point is shown by red segments. We varied the maximum number of proposals N from one to ten and twenty. We used the reflection operator $\mathcal{R}_x = -\mathcal{I}$ and the velocity refreshment probability $p^{\text{ref}} \equiv 0.1$ for this experiment. In the case of $N = 1$, where no subsequent proposals were made if the first proposal was rejected, there was



(a) The first 120,000 states in the constructed Markov chain are shown as black dots. The trajectory connecting every fourth point is shown by red segments up to one hundred points.



(b) The distance between the sample mean and the center of the distribution as a function of runtime. In the legend, "gradient" means $\mathcal{R}_x = R_{\nabla U(x)}$, "−identity" means $\mathcal{R}_x = -\mathcal{I}$, and "mix" indicates the case where these two operators are used with equal probability.

Fig. 8 Numerical results for the four "C" model with varying p^{ref} and reflection operators

Algorithm 7: Sequential-proposal discrete time bouncy particle sampler

Input : Distribution of the maximum number of proposals and the number of accepted proposals, $\nu(N, L)$
Time step length distribution, $\mu(d\tau)$
Velocity distribution density, $\psi(v\,; x)$
Reflection operators $\{\mathcal{R}_x\}$, $\{\mathcal{R}'_x\}$
Velocity refreshment probability, $p^{\text{ref}}(x)$
Number of iterations, M

Output: A draw of Markov chain, $\left(X^{(i)}\right)_{i=1,\ldots,M}$

Initialize: Set $X^{(0)}$ arbitrarily and draw $V^{(0)} \sim \psi(\cdot\,; X^{(i+1)})$.
for $i \leftarrow 0 : M-1$ **do**
  Draw $N, L \sim \nu(\cdot, \cdot)$
  Draw $\tau \sim \mu(\cdot)$
  Draw $\Lambda \sim \text{unif}(0, 1)$
  Set $X^{(i+1)} \leftarrow X^{(i)}$ and $V^{(i+1)} \leftarrow \mathcal{R}_{X^{(i)}} V^{(i)}$
  Set $n_a \leftarrow 0$
  Set $(Y_0, W_0) \leftarrow (X^{(i)}, V^{(i)})$
  for $n \leftarrow 1 : N$ **do**
    Set $(Y_n, W_n) = (Y_{n-1} - \mathcal{R}_{Y_{n-1}} W_{n-1}\tau,\ -\mathcal{R}_{Y_{n-1}} W_{n-1})$
    if $\Lambda < \dfrac{\pi(Y_n)}{\pi(X)}$ **then** $n_a \leftarrow n_a + 1$
    if $n_a = L$ **then**
      Set $(X^{(i+1)}, V^{(i+1)}) \leftarrow (Y_n, W_n)$
      `break`
    end
  end
  With probability $p^{\text{ref}}(X^{(i+1)})$, refresh $V^{(i+1)} \sim \psi(\cdot\,; X^{(i+1)})$
end

no jumps between "C"s. As we increased N to ten and twenty, the jump between the "C"s happened more frequently, and mixing happened faster. The sample means clearly did not converge to the true mean when $N = 1$, but the sample means converged to the true mean with similar rates when $N = 10$ or $N = 20$.

Figure 8 shows the numerical results for the original four "C" model for various choices of velocity reflection operator and velocity refreshment probability. The maximum number of proposals N was fixed at twenty. The left column shows the results when the reflection operator $\mathcal{R}_x = -\mathcal{I}$ was used. In the middle column, the reflection operator $R_{\nabla U(x)}$ defined in (25) was used. In the right column, the reflection operator was randomly chosen between $-\mathcal{I}$ and $R_{\nabla U(x)}$ with equal probability whenever the reflection operator was used by the algorithm. The reflection operator $\mathcal{R}_x = -\mathcal{I}$ resulted in a non-ergodic Markov chain when we did not refresh the velocity (top row, left). When $p^{\text{ref}} = 0.1$, the choice of $R_{\nabla U(x)}$ resulted in a slower convergence to the target distribution compared to the case $\mathcal{R}_x = -\mathcal{I}$. The speed of convergence was improved if the algorithm used both $-\mathcal{I}$ and $R_{\nabla U(x)}$ with equal probability. From these results, we see that it is crucial to occasionally refresh the velocity for certain choices of $\mathcal{R}_x$ and that using a mixture of different velocity reflection operators can speed up the mixing of the Markov chain.

6 Conclusion

The sequential-proposal MCMC framework is readily applicable to a wide range of MCMC algorithms. The flexibility and simplicity of the framework allow for various adjustments to the algorithms and offer possibilities of developing new ones. In this paper, we showed that the numerical efficiency of MCMC algorithms can be improved by using sequential proposals. In particular, we developed two novel NUTS-type algorithms, which showed higher numerical efficiency than the original NUTS by Hoffman and Gelman (2014) on two examples we examined. The spNUTS1 algorithm can be numerically more efficient than the original NUTS and spNUTS2 when the evaluation of the target density takes an amount of time that is at least comparable to that for the evaluation of the gradient of the log target density. However, the computational gain for spNUTS1 can be smaller when automatic differentiation is used to evaluate the target density and its gradient together, as the probabilistic programming language Stan does (Carpenter et al. 2017). Finally, in Sect. 5, we applied the sequential-proposal framework to the bouncy particle sampler (BPS) and demonstrated an advantageous property that the sequential-proposal BPS could readily make jumps between multiple modes. The possibilities of other applications of the sequential-proposal MCMC framework can be explored in future research.

Acknowledgements This work was supported by National Science Foundation Grants DMS-1513040 and DMS-1308918. The authors thank Edward Ionides, Aaron King, and Stilian Stoev for comments on an earlier draft of this manuscript. The authors also thank Jesús María Sanz-Serna for informing us about related references.

Availability of data and materials The German credit dataset used in Sect. 4.3.2 is available from the UCI repository https://archive.ics.uci.edu/ml/datasets/statlog+(german+credit+data).

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

Code availability The source codes used in this work are available at https://github.com/joonhap/spMCMC.

References

Andrieu, C., Atchade, Y.: On the efficiency of adaptive MCMC algorithms. Electron. Commun. Probab. **12**, 336–349 (2007)
Andrieu, C., Livingstone, S.: Peskun-Tierney ordering for Markov chain and process Monte Carlo: beyond the reversible scenario. (2019). arXiv preprint arXiv:1906.06197
Andrieu, C., Moulines, É.: On the ergodicity properties of some adaptive MCMC algorithms. Ann. Appl. Probab. **16**, 1462–1505 (2006)
Andrieu, C., Thoms, J.: A tutorial on adaptive MCMC. Stat. Comput. **18**, 343–373 (2008)

Atchadé, Y., Fort, G.: Limit theorems for some adaptive MCMC algorithms with subgeometric kernels. Bernoulli **16**, 116–154 (2010)
Atchadé, Y.F., Fort, G.: Limit theorems for some adaptive MCMC algorithms with subgeometric kernels: Part II. Bernoulli **18**, 975–1001 (2012)
Atchadé, Y.F., Rosenthal, J.S.: On adaptive Markov chain Monte Carlo algorithms. Bernoulli **11**, 815–828 (2005)
Beskos, A., Pillai, N., Roberts, G., Sanz-Serna, J.-M., Stuart, A.: Optimal tuning of the hybrid Monte Carlo algorithm. Bernoulli **19**, 1501–1534 (2013)
Betancourt, M.: A conceptual introduction to Hamiltonian Monte Carlo. (2017). arXiv preprint arXiv:1701.02434
Bouchard-Côté, A., Vollmer, S.J., Doucet, A.: The bouncy particle sampler: a nonreversible rejection-free Markov chain Monte Carlo method. J. Am. Stat. Assoc. **113**, 855–867 (2018)
Calderhead, B.: A general construction for parallelizing Metropolis-Hastings algorithms. Proc. Natl. Acad. Sci. **111**, 17408–17413 (2014)
Campos, C.M., Sanz-Serna, J.: Extra chance generalized hybrid Monte Carlo. J. Comput. Phys. **281**, 365–374 (2015)
Carpenter, B., Gelman, A., Hoffman, M.D., Lee, D., Goodrich, B., Betancourt, M., Brubaker, M., Guo, J., Li, P., Riddell, A.: Stan: a probabilistic programming language. J. Stat. Softw. **76**, 1 (2017)
Dua, D., Graff, C.: UCI machine learning repository. (2017). http://archive.ics.uci.edu/ml. Accessed 17 Jan 2020
Duane, S., Kennedy, A.D., Pendleton, B.J., Roweth, D.: Hybrid Monte Carlo. Phys. Lett. B **195**, 216–222 (1987)
Fang, Y., Sanz-Serna, J.M., Skeel, R.D.: Compressible generalized hybrid Monte Carlo. J. Chem. Phys. **140**, 174108 (2014)
Flegal, J.M., Hughes, J., Vats, D., Dai, N.: mcmcse: Monte Carlo Standard Errors for MCMC. Riverside, CA, Denver, CO, Coventry, UK, and Minneapolis, MN. R package version 1.3-2 (2017)
Geyer, C.J.: Markov chain Monte Carlo maximum likelihood. Retrieved from the University of Minnesota Digital Conservancy, Interface Foundation of North America (1991)
Goodman, J., Weare, J.: Ensemble samplers with affine invariance. Commun. Appl. Math. Comput. Sci. **5**, 65–80 (2010)
Green, P.J., Mira, A.: Delayed rejection in reversible jump Metropolis-Hastings. Biometrika **88**, 1035–1053 (2001)
Gupta, S., Irbäc, A., Karsch, F., Petersson, B.: The acceptance probability in the hybrid Monte Carlo method. Phys. Lett. B **242**, 437–443 (1990)
Haario, H., Saksman, E., Tamminen, J.: An adaptive Metropolis algorithm. Bernoulli **7**, 223–242 (2001)
Hastings, W.K.: Monte Carlo sampling methods using Markov chains and their applications. Biometrika **57**, 97–109 (1970)
Hoffman, M.D., Gelman, A.: The No-U-turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo. J. Mach. Learn. Res. **15**, 1593–1623 (2014)
Horowitz, A.M.: A generalized guided Monte Carlo algorithm. Phys. Lett. B **268**, 247–252 (1991)
Hukushima, K., Nemoto, K.: Exchange Monte Carlo method and application to spin glass simulations. J. Phys. Soc. Jpn. **65**, 1604–1608 (1996)
Kou, S., Zhou, Q., Wong, W.H., et al.: Equi-energy sampler with applications in statistical inference and statistical mechanics. Ann. Stat. **34**, 1581–1619 (2006)
Leimkuhler, B., Reich, S.: Simulating Hamiltonian dynamics, vol. 14. Cambridge University Press, Cambridge (2004)
Liouville, J.: Note on the theory of the variation of arbitrary constants. Journal de Mathématiques Pures et Appliquées **3**, 342–349 (1838)
Liu, J.S., Liang, F., Wong, W.H.: The multiple-try method and local optimization in Metropolis sampling. J. Am. Stat. Assoc. **95**, 121–134 (2000)
Marinari, E., Parisi, G.: Simulated tempering: a new Monte Carlo scheme. EPL (Europhys. Lett.) **19**, 451 (1992)
Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., Teller, E.: Equation of state calculations by fast computing machines. J. Chem. Phys. **21**, 1087–1092 (1953)
Mira, A.: On metropolis-hastings algorithms with delayed rejection. Metron **59**, 231–241 (2001)
Mira, A., Møller, J., Roberts, G.O.: Perfect slice samplers. J. R. Stat. Soc. Ser. B (Stat. Methodol.) **63**, 593–606 (2001)
Neal, R.M.: An improved acceptance procedure for the hybrid Monte Carlo algorithm. J. Comput. Phys. **111**, 194–203 (1994)
Neal, R.M.: Slice sampling. Ann. Stat. **31**, 705–767 (2003)
Neal, R.M.: MCMC using Hamiltonian dynamics. In: Brooks, S., Gelman, A., Jones, G., Meng, X.-L. (eds.) Handbook of Markov chain Monte Carlo, pp. 113–162. CRC press (2011)
Peskun, P.H.: Optimum Monte-Carlo sampling using markov chains. Biometrika **60**, 607–612 (1973)
Peters, E.A., et al.: Rejection-free Monte Carlo sampling for general potentials. Phys. Rev. E **85**, 026703 (2012)
Plummer, M., Best, N., Cowles, K., Vines, K.: CODA: convergence diagnosis and output analysis for MCMC. R News **6**, 7–11 (2006)
R Core Team.: R: a Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna (2018). https://www.R-project.org/
Roberts, G.O., Rosenthal, J.S.: Optimal scaling of discrete approximations to Langevin diffusions. J. R. Stat. Soc. Ser. B (Stat. Methodol.) **60**, 255–268 (1998)
Roberts, G.O., Rosenthal, J.S.: Convergence of slice sampler Markov chains. J. R. Stat. Soc. Ser. B (Stat. Methodol.) **61**, 643–660 (1999)
Roberts, G.O., Rosenthal, J.S.: Coupling and ergodicity of adaptive Markov chain Monte Carlo algorithms. J. Appl. Probab. **44**, 458–475 (2007)
Roberts, G.O., Gelman, A., Gilks, W.R.: Weak convergence and optimal scaling of random walk Metropolis algorithms. Ann. Appl. Probab. **7**, 110–120 (1997)
Sexton, J., Weingarten, D.: Hamiltonian evolution for the hybrid Monte Carlo algorithm. Nucl. Phys. B **380**, 665–677 (1992)
Sherlock, C., Fearnhead, P., Roberts, G.O.: The random walk Metropolis: Linking theory and practice through a case study. Stat. Sci. **25**, 172–190 (2010)
Sohl-Dickstein, J., Mudigonda, M., DeWeese, M.R.: Hamiltonian Monte Carlo without detailed balance. (2014). arXiv preprint arXiv:1409.5191
Tierney, L.: A note on Metropolis-Hastings kernels for general state spaces. Ann. Appl. Probab. **8**, 1–9 (1998)
Tierney, L., Mira, A.: Some adaptive Monte Carlo methods for Bayesian inference. Stat. Med. **18**, 2507–2515 (1999)
Vanetti, P., Bouchard-Côté, A., Deligiannidis, G., Doucet, A.: Piecewise deterministic Markov chain Monte Carlo. (2017) arXiv preprint arXiv:1707.05296