The Hamiltonian Monte Carlo and the No-U-Turn Sampler

Jingyue Lu   Marco Palma

October 16, 2017

Abstract

We present the R package 'NUTS', which contains functions for Hamiltonian Monte Carlo and one of its extensions called No-U-Turn Sampler with Dual Averaging.

1 Introduction

This project investigates the No-U-Turn Sampler (NUTS). M.D. Hoffman and A. Gelman introduced NUTS in 2011 to address the tuning issues involved in Hamiltonian Monte Carlo (HMC) algorithm. To be more specific, we start by introducing the theoretical ideas behind HMC and NUTS. Then we implement both methods in R to compare their performances. Especially, we are interested in how effective NUTS is as an extension of HMC.

2 Hamiltonian Monte Carlo

The Hamiltonian Monte Carlo (also known as hybrid Monte Carlo, introduced by Duane et al., 1987) is a Markov chain Monte Carlo method that avoids simple random work behaviour by proposing remote states for Metropolis algorithm, thereby achieving rapid convergence to a target distribution. The idea of HMC is to introduce a d-dimensional vector r called momentum (where d is the dimension of the parameter vector θ), independently drawn from a distribution we can easily sample from (usually a standard multivariate normal). The unnormalized joint density can be written as

$$p(\theta, r) \propto \exp\left\{\mathcal{L}(\theta) - \frac{1}{2} r \cdot r\right\}$$

where $\mathcal{L}$ is the unnormalized logarithm of the joint posterior density of θ and $\cdot$ is an inner product.

Samples of (θ, r) are obtained by using the Störmer-Verlet *leapfrog integrator*. Given the gradient $\nabla_\theta \mathcal{L}$ and the step size ϵ, the updates proceed as follows:

$$r^{t+\epsilon/2} = r^t + \epsilon/2 \nabla_\theta \mathcal{L}(\theta^t) \quad \theta^{t+\epsilon} = \theta^t + \epsilon r^{t+\epsilon/2} \quad r^{t+\epsilon} = r^{t+\epsilon/2} + (\epsilon/2) \nabla_\theta \mathcal{L}(\theta^{t+\epsilon}).$$

The leapfrog procedure is applied L times in order to generate a pair $(\tilde{\theta}, \tilde{r})$. Then the proposal for the m-th iteration of the Markov chain is $(\tilde{\theta}, -\tilde{r})$ (where the minus sign for r is only used to guarantee time reversibility). According to Metropolis ratio, the proposal is accepted with a probability

$$\alpha = \min\left\{1, \frac{p(\tilde{\theta}, \tilde{r})}{p(\theta^{m-1}, r_0)}\right\}$$

where r_0 is the resampled momentum at the m-th iteration.

Despite the increase in efficiency, the usability and the performances of HMC are affected not only by the computation of $\nabla_\theta \mathcal{L}$ (that can be addressed via numerical procedures) but also by the step size ϵ and number of steps L chosen within the leapfrog. Indeed, when ϵ is too large the acceptance rates will be low, whereas for small values of ϵ there will be a waste of computation because of the tiny steps. In terms of L, if it is too small the samples will be close to each other, but if it is too large the trajectory in the parameters space will loop back to the previous steps.

3 No-U-Turn Sampler

NUTS addresses the problem of tuning the number of steps L. In short, NUTS repeatedly doubles the length of the current trajectory until increasing L no longer leads to an increased distance between the initial θ and a newly proposed $\tilde{\theta}$. That is, the $\tilde{\theta}$ makes a "U-turn".

3.1 Derivation of simplified NUTS algorithm

The derivation of NUTS algorithm can be divided into two parts: the conditions this algorithm has to satisfy in order to be theoretically sound and the criteria NUTS uses to stop the doubling procedure.

To simplify the derivation of NUTS, Hoffman and Gelman introduced a slice variable u. Simplified NUTS considers the augmented model

$$p(\theta, r, u) \propto \mathbb{I}\left[u \in \left[0, \exp\left\{\mathcal{L}(\theta) - \frac{1}{2} r \cdot r\right\}\right]\right],$$

where $\mathbb{I}[\cdot]$ is 1 if $u \in \left[0, \exp\{\mathcal{L}(\theta) - \frac{1}{2} r \cdot r\}\right]$ is true and 0 otherwise. Introducing u renders the conditional probabilities $p(u|\theta, r) \sim \text{Unif}(u; [0, \exp\{\mathcal{L}(\theta) - \frac{1}{2} r \cdot r\}])$ and $p(\theta, r|u) \sim \text{Unif}(\theta', r' | \exp\{\mathcal{L}(\theta) - \frac{1}{2} r \cdot r\} \geq u)$ and hence simplifies the simulation.

In terms of theoretical requirements, the simplified NUTS algorithm must not only leave the target distribution invariant but also guarantee the time reversibility. Under several mild conditions, NUTS uses the following procedure to sample θ^{t+1} from θ^t to achieve invariant target distribution:

1. sample $r \sim \mathcal{N}(0, I)$,

2. sample $u \sim \mathrm{Unif}\left([0, \exp\left\{\mathcal{L}(\theta^t) - \frac{1}{2} r \cdot r\right\}]\right)$,
3. sample $\mathcal{B}$, $\mathcal{C}$ from their conditional distribution $p(\mathcal{B}, \mathcal{C} | \theta^t, r, u, \epsilon)$,
4. sample $\left(\theta^{t+1}, r\right)$ uniformly from the set $\mathcal{C}$.

Here, $\mathcal{C}$ is a set of candidate position-momentum states while $\mathcal{B}$ is the set of all position-momentum states computed by leapfrog integrator during each NUTS iteration. Clearly, $\mathcal{C} \subseteq \mathcal{B}$. For the purpose of this project, we omit the proof of the validity of the procedure but only state the key observations and results. We first point out that steps 1, 2, 3 constitute a valid Gibbs sampling update for r, u, $\mathcal{B}$, $\mathcal{C}$. Secondly, as a result of the prequiresites of the above procedure, we use the following condition to determine whether a state in $\mathcal{B}$ is also in $\mathcal{C}$:

$$(\theta', r') \in \mathcal{C}, \qquad \text{if} \quad u \leq \exp\left\{\mathcal{L}(\theta') - \frac{1}{2} r' \cdot r'\right\} \tag{C.1}$$

For what concerns time reversibility, it is important as it ensures the algorithm converge to the target distribution. NUTS uses a recursive algorithm to preserve it. Recall that, to find a trajectory length for each NUTS interation, NUTS doubles the current trajectory repeatedly until an u-turn is encountered. To simulate forward and backward movement in time, during each doubling, NUTS allows the new subtrajectory to start from either the leftmost or rightmost point of the old trajectory and use leapfrog to trace a path either running backwards or forwards respectively. The proccess continues until stopping criteria are met. To illustrate, we assume the starting point is (θ_0^1, r), where the subscript is the number of step and the superscript is the index of the point in that step. Also, let $v \in \{1, -1\}$. v is randomly choosen in each step to present the direction of movement. We are only interested in the path for θ.

To simulate forward and backward movement in time, during each doubling, a new subtrajectory is traced by leapfrog either forward from the rightmost state or backward from the leftmost state until the stopping criteria are met. To illustrate, we assume the starting point is (θ_0^1, r), where the subscript is the number of doubling steps and the superscript is the index of the state in that doubling step. Also, let $v \in \{1, -1\}$ be randomly chosen in each doubling step to determine the direction of movement. We are interested in the path for θ:
Step j=1: (v= 1) $\Rightarrow \theta_0^1 \rightarrow \theta_1^1$.
Step j=2: (v= 1) $\Rightarrow \theta_0^1 \rightarrow \theta_1^1 \rightarrow \theta_2^1 \rightarrow \theta_2^2$.
Step j=3: (v=-1) $\Rightarrow \theta_3^4 \leftarrow \theta_3^3 \leftarrow \theta_3^2 \leftarrow \theta_3^1 \leftarrow \theta_0^1 \rightarrow \theta_1^1 \rightarrow \theta_2^1 \rightarrow \theta_2^2$.
In the example above, we build a three-step path. We can also think the path after each step j as a binary tree of height j, so final path is a binary tree of height 3

$$\underbrace{\underbrace{\underbrace{\theta_3^4 \leftarrow \theta_3^3}_{\text{level 1}} \leftarrow \underbrace{\theta_3^2 \leftarrow \theta_3^1}_{\text{level 1}}}_{\text{level 2}} \leftarrow \underbrace{\underbrace{\theta_0^1 \rightarrow \theta_1^1}_{\text{level 1}} \rightarrow \underbrace{\theta_2^1 \rightarrow \theta_2^2}_{\text{level 1}}}_{\text{level 2}}}_{\text{level 3}}.$$

Finally, we discuss the stopping criteria used in NUTS. A straightforward and essential stopping condition for NUTS implements the idea of no-U-turn. We observe that when $\tilde{\theta}$ makes a U-turn, the following derivative should be less than 0:

$$\frac{d}{dt}\frac{(\tilde{\theta}-\theta)\cdot(\tilde{\theta}-\theta)}{2} = (\tilde{\theta}-\theta)\cdot\frac{d}{dt}(\tilde{\theta}-\theta) = (\tilde{\theta}-\theta)\cdot r < 0. \tag{C.2}$$

This condition is checked for each subtree and also for the whole tree. In addition, NUTS also stops expanding $\mathcal{B}$ when any newly discovered states in the continuing process has extremely low probability to be in $\mathcal{C}$. To formulate, NUTS develops the following condition based on (C.1):

$$\mathcal{L}(\theta) - \frac{1}{2}r\cdot r - \log u > -\Delta_{max}. \tag{C.3}$$

In NUTS, Δ_{max} is set to 1000, so the algorithm continues as long as the simulation is moderately accurate.

So far, we have addressed all aspects needed for deriving the simplified NUTS algorithm. We summarise the simplified NUTS algorithm below.

Algorithm 1 Sample for a point $\tilde{\theta}$ using a NUTS iteration

```
Require: The initial position $\theta_0^1$.
  Resample $r \sim \mathcal{N}(0, I)$.
  Resample $u \sim \mathrm{Unif}([0, \exp\{\mathcal{L}(\theta^t) - \frac{1}{2}r\cdot r\}])$.
  {$j$ is the number of doubling steps we take to build a trajectory path.}
  {$s$ is an indicator variable. It becomes 0 when a stopping criterion is met.}
  Initialise $j = 0$, $s = 1$, $\mathcal{C} = \{(\theta_0^1, r)\}$
  {Set the rightmost (+) and leftmost (-) points of the trajectory path.}
  Initialise $\theta^+ = \theta_0^1$, $\theta^- = \theta_0^1$.
  while s=1 do
    while Build a binary tree of height $j$ do
      For each new node: check C.1 to determine whether or not to add the new node into $\mathcal{C}$.
      For each new node: check C.3. If Condition Three is met, set $s = 0$.
      For each subtree: check C.2. If Condition Two is met, set $s = 0$.
    end while
    Update $\theta^+$ and $\theta^-$ to be the rightmost and leftmost point of the whole path.
    Check C.2 for the whole path.If it is met, set $s = 0$.
    j= j+1.
  end while
  Sample $\tilde{\theta}$ uniformly from $\mathcal{C}$.
```

3.2 Efficient NUTS

Hoffman and Gelman (2014) proposed an efficient version of the NUTS algorithm. Firstly, the execution is halted exactly once a stopping criterion is met instead of at the end of the corresponding doubling step. Secondly, simplified NUTS requires to store 2^j states to perform uniform sampling at the end. A solution for reducing this memory requirement from $O(2^j)$ to $O(j)$ is to use a more sophisticated transition kernel and to exploit the binary tree structure of the trajectory path. We refer interested reader to Hoffman and Gelman (2014) for details. The NUTS function included in the package is an efficient algorithm with these improvements implemented.

4 Dual averaging

In Hoffman and Gelman (2014) a method based on the primal-dual algorithm by Nesterov (2009) is also provided for setting the step size ϵ for both HMC and NUTS. In the context of MCMC, considered the statistic $H_t = \delta - \alpha_t$ where δ is a specified average acceptance probability and α_t is the observed Metropolis one at time t. The key idea of the dual averaging algorithm is that, under some specific conditions, $H_t = 0$ can be approached (that is, to approximate the desired average acceptance probability) by tuning $\log(\epsilon)$ using an iterative procedure. In the implementation, ϵ is tuned during the warmup phase (the user needs to specify the number of iterations of the warmup phase) and kept constant for the subsequent iterations. The user needs to specify the target mean acceptance rate δ and the number of iterations of the warmup phase.

References

Simon Duane, Anthony D Kennedy, Brian J Pendleton, and Duncan Roweth. Hybrid monte carlo. *Physics letters B*, 195(2):216–222, 1987.

Matthew D Hoffman and Andrew Gelman. The no-u-turn sampler: adaptively setting path lengths in hamiltonian monte carlo. *Journal of Machine Learning Research*, 15(1):1593–1623, 2014.

Yurii Nesterov. Primal-dual subgradient methods for convex problems. *Mathematical programming*, 120(1):221–259, 2009.