concordance=TRUE

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

Jingyue Lu Marco Palma

October 15, 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 as an extension to Hamiltonian Monte Carlo (HMC) algorithm. Using Hamiltonian Dynamics, HMC avoids simple random work behaviour by proposing distant proposals for Metropolis algorithm, thereby converging to high-dimensional target distributions at a much faster speed than other general methods. However, the performance of HMC depends heavily on two tuning parameters: the trajectory length and the stepsize. A poor choice of these two parameters will lead to drastic decrease of the performance of HMC. NUTS is designed to specifically address the problem of tuning the trajectory length. In this project, we start by briefly introducing the ideas behind of HMC and NUTS. A short theoretical foundation for both algorithms is also included. Both HMC and NUTS are implemented in R to compare the performances between these two methods. Especially, we are interested in how effective NUTS is in handling the barriers of HMC.

2 Hamiltonian Monte Carlo

The Hamiltonian Monte Carlo (HMC, also known as hybrid Monte Carlo, firstly introduced by **?**) is a Markov chain Monte Carlo method that overcomes the random walk behaviour typical of this class of algorithms in order to achieve faster convergence to a target distribution. This is obtained by introducing a d-dimensional vector r called momentum (where d is the dimension of the parameter vector θ), indipendently drawn from a distribution easy to sample from

(usually a standard multivariate normal). It is possible to write the unnormalized joint density as

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

where the unnormalized logarithm of the joint density of θ is reduced by the inner product of r with itself. The update of (θ, r) is performed using the Stormer-Verlet *leapfrog integrator*, that transforms each coordinate on the basis of the others by using the gradient $\nabla_\theta \mathcal{L}$:

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

Given a step size ϵ, the leapfrog is applied L times in order to generate a pair $(\tilde{\theta}, \tilde{r})$; the proposal for the m-th iteration of the Markov chain $(\tilde{\theta}, -\tilde{r})$ (where the minus sign for r is only used to guarantee time reversibility) is accepted using a standard Metropolis accept-reject procedure with 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 is affected not only by the computation of the gradient of the logarithm of the posterior density (that can be addressed via numerical procedures) but in a more relevant way by the specification within the leapfrog of the step size ϵ and number of steps L. Indeed, when ϵ is too large the acceptance rates will be low, whereas for too small values of ϵ there will be a waste of computation because of the tiny steps. On the other side, a poor choice of L might lead to slow moving, since if L is too small the samples will be close to each other, but if L is too large the trajectory in the parameters space will loop back to the previous steps.

3 No-U-Turn Sampler

In this section, we introduce NUTS to address the problem of tuning the trajectory length L. NUTS is a method that builds upon HMC. Unlike HMC, which sets a fixed trajectory length L for all proposals, NUTS dynamically chooses L for each proposal using the idea of No-U-Turn. In short, to decide the length L for a proposal, 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 NUTS algorithm has to satisfy in order to be theoretically sound and the stopping criteria NUTS uses to stop the doubling procedure.

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

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

where $\mathbb{I}[\cdot]$ is 1 if $u \in [0, \exp\{\mathcal{L} - \frac{1}{2} r \cdot r\}$ is true and 0 otherwise. The useful results of introducing a slice variable u are that the conditional probabilities $p(u|\theta, r) \sim \text{Unif}(u; [0, \exp\{\mathcal{L}(\theta) - \frac{1}{2} r \cdot r\}])$ and $p(\theta, r|u) \sim \text{Unif}(\theta', r'| \exp\{\mathcal{L}(\theta) - \frac{1}{2} r \cdot r\} \geq u)$ are both uniform and hence can be easily simulated

In terms of theoretical requirements for simplified NUTS algorithm, it is required that the algorithm must not only leave the target distrbution invariant but also guarantee the time reversibility. Under several not too strict 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 $u \sim \text{Unif}\left(\left[0, \exp\left\{\mathcal{L}(\theta^t) - \frac{1}{2} r \cdot r\right\}\right]\right)$,
3. sample $\mathcal{B}$, $\mathcal{C}$ from their conditional distribution $p(\mathcal{B}, \mathcal{C}|\theta^t, r, u, \epsilon)$,
4. sample θ^{t+1}, r 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 that computed by leapfrod 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, the construction of the distribution $p(\mathcal{B}, \mathcal{C}|\theta^t, r, u, \epsilon)$ relies on the strategy employed by NUTS to achieve time reversibility and will be introduced later. Thirdly and lastly, 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 $\mathcal{C}$:

Condition One

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

We now consider the time reversibility requirement. Time reversibility is important as it ensures the algorithm converge to the correct distribution. NUTS uses a recursive algorithm to preserve time reversibility. 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.

Step j=1: ** (v= 1) $\theta_0^1 \to \theta_1^1$.
Step j=2: ** (v= 1) $\theta_0^1 \to \theta_1^1 \to \theta_2^1 \to \theta_2^2$.
Step j=3: ** (v=-1) $\theta_3^4 \leftarrow \theta_3^3 \leftarrow \theta_3^2 \leftarrow \theta_3^1 \leftarrow \theta_0^1 \to \theta_1^1 \to \theta_2^1 \to \theta_2^2$.

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

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

Finally, we discuss the stopping criteria used in NUTS. A straightforwd and essential stopping condition for NUTS implements the idea of no u-turn. We observe that when $\tilde{\theta}$ makes a U-turn, the derivative, with respect to time, of half the squared distance (half is chosen to simplify calculations) between θ and $\tilde{\theta}$ should be less than 0. Mathematically, we have

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

Thus, the NUTS algorithm stops when:

Condition Two:

$$(\theta_{end} - \theta_{start}) \cdot r < 0.$$

The condition is checked for each subtree and also for the whole tree. Other than the stopping condition for u-turn, NUTS also stops expanding $\mathcal{B}$ when any newly discovered states in the continuing process are likely to have near 0 probability to be $\mathcal{C}$. To formulate, NUTS develops the following condition based on Condition One:

Condition Three: The algorithm stops when

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

In NUTS, Δ_{max} is set to 1000, so the algorithm continuos 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 simiplified NUTS algorithm below.

To sample for a point $\tilde{\theta}$, we run the following NUTS iteration with a given θ_0^1:

1. Resample $r \sim \mathcal{N}(0, I)$.
2. Resample $u \sim \text{Unif}([0, \exp\{\mathcal{L}(\theta^t) - \frac{1}{2} r \cdot r\}])$. $\%j$ 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.

3. Initialise $j = 0$, $s = 1$, $\mathcal{C} = \{(\theta_0^1, r)\}$. % Set the rightmost (+) and leftmost (-) points of the trajectory path.

4. **while** s=1 do

   (a) Build a binary tree of height j

      i. For each new node: check Condition One to determine whether or not to add the new node into $\mathcal{C}$.

      ii. For each new node: check Condition Three. If Condition Three is met, set $s = 0$.

      iii. For each subtree: check Condition Two. If Condition Two is met, set $s = 0$.

   (b) For the newly generated whole path:

      i. Update θ^+ and θ^- to be the rightmost and leftmost point of the whole path.

      ii. Check Condition Two. If it is met, set $s = 0$.

      iii. For each subtree: check Condition Two. If Condition Two is met, set $s = 0$.

   (c) j= j+1 # Update variables for while loop

5. Sample $\tilde{\theta}$ uniformly from $\mathcal{C}$.

3.2 Efficient NUTS

Simplified NUTS algorithm need to evaluate log posterior probability and its gradient at $2^j - 1$ points apart from $O(2^j)$ operations to check the stopping criteria **?**. Furthermore, the final doubling iteration continues even when a stopping criterion is met in the middle of the process. In terms of memory, simplifed NUTS requires to store 2^j states to perform uniform sampling at the end. These facts seriously deterioate the efficiency of NUTS. The second issue can be easily solved by terminating the excucation of the recrusion once s becomes 0. **?** prosposed a solution for reducing the meomery requirment from $O(2^j)$ to $O(j)$. The key idea of this memory reduction is to use a more sophisticated transition kernal and to exploit the binary tree structure of the trajectory path. We refer interested reader to **?** for details. The NUTS function included in the package is an efficient algorithm with these improvements implemented.

4 Dual averaging

In **?** a method for setting the step size ϵ is also addressed for both HMC and NUTS, based on the primal-dual algorithm by **?**, primarily intended for stochastic convex optimization. In the context of MCMC, considered a statistic H_t (for example $H_t = \delta - \alpha_t$ where δ is a specified average acceptance probability

and α_t is the observed Metropolis one at time t). Suppose that there is a tunable parameter $x \in \mathbb{R}$ for which the nonincreasing function

$$h(x) = \mathbb{E}_t[H_t|x] = \lim_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T} \mathbb{E}_t[H_t|x] = 0;$$

under some specific conditions x_t can be updated by replacing

$$x_{t+1} \leftarrow \mu - \frac{\sqrt{t}}{\gamma}\frac{1}{t+t_0}\sum_{t=1}^{T} H_i \quad \bar{x}_{t+1} \leftarrow \eta_t x_{t+1} + (1-\eta_t)\bar{x}_t \tag{1}$$

where the shrinkage of x_t to a freely chosen point μ is controlled by a free nonnegative parameter γ, $t_0 > 0$ is used in order to stabilize the initial iterations and $\bar{x}_1 = x_1$. The parameter $\eta_t \equiv t^{-\kappa}$ with $\kappa \in (0.5, 1]$ is the step size schedule for which the conditions $\sum_t \eta_t = \infty$ and $\sum_t \eta_t^2 < \infty$ (that ensures that $h(\bar{x}_t)$ converges to 0) are met. The usual procedure is to tune x during the warmup phase and set it constant for the subsequent iterations. The benefits of the choices of the parameters t_0 and κ are discussed in **?**.

The same paper proposes a heuristic for choosing the initial value ϵ_1 so that convergence is reached faster: ϵ is iteratively multiplied by 2^a with $a = \{-1, 1\}$ until the acceptance probability of the leapfrog proposal crosses 0.5. Then, simply set $\mu = \log(10\epsilon_1)$ so that the algorithm is more inclined to test on values of $\epsilon > \epsilon_1$.

The tuning of ϵ for both HMC and NUTS is usually done so that the average Metropolis acceptance rate is equal to a prespecified $\delta \in (0, 1)$: setting the statistic $H_t \equiv \delta - \alpha_t$ and the tunable parameter $x \equiv \log\epsilon$ in (**??**) , so that to obtain $h^{HMC}(\epsilon) \equiv \mathbb{E}_t[\alpha_t|\epsilon] = \delta$. The only difference concerns the definition of α_t, since in the NUTS algorithm there is an accept/reject step at each iteration: in this case, α_t is to be intended as the average probability that HMC would accept the (θ, r) state in the last doubling iteration. In this case, for the NUTS algorithm the user must only provide the target mean acceptance rate δ and the number of iteration of the tuning phase.