Comparing The Hamiltonian Monte Carlo sampler and The No-U-Turn sampler

Cangao Chu, Shen Huang, Jing Liao

March 2020

Abstract

This report has been comparing the performance of Hamiltonian Monte Carlo (HMC) sampler, and No-U-Turn sampler (NUTS), in the sense of the running time, effect sample size and acceptance rate. Some preliminary results indicate that the NUTS is globally better than the HMC, as it is faster and have higher effect sample size. Although NUTS has a relatively lower acceptance ratio, which keeps a smaller stepsize to overcome certain modeling difficulties

Keywords: Bayesian, Hamiltonian Monte Carlo, No-U-Turn Sampler, MCMC

1 Introduction

Bayesian models are widely used in machine learning and statistics. Exact posterior inference in such models is rarely tractable. Rather than computing a deterministic approximation to a target posterior distribution, Markov chain Monte Carlo (MCMC) offer schemes for drawing a series of correlated samples that will converge in distribution to the target distribution. However, the random-walk behavior of many Markov Chain Monte Carlo (MCMC) algorithms makes Markov chain convergence to target distribution p(x) inefficient, resulting in slow mixing. In this article we focus on two MCMC algorithms that propose future states in the Markov Chain using Hamiltonian dynamics rather than a probability distribution. This allows the Markov chain to explore target distribution much more efficiently, resulting in faster convergence.

Hamiltonian Monte Carlo (HMC) is a powerful sampling method. By defining a Hamiltonian function in terms of the posterior distribution and a momentum component, HMC algorithm can profoundly reduce the auto-correlation between adjacent samples, while the auto-correlation is sensibly high in Metropolis-Hasting scheme. Another advantage HMC algorithm possesses is, the Hamiltonian dynamics is theoretically invariant in its value, which guarantees a high accepting ratio when using MH proposal. As Neal(2011)[4] proved before, the HMC is reversible, and hence leaves the target distribution invariant. Therefore, HMC can sample correctly from the target distribution. However, the increases of HMC's efficiency need prices. A limitation of HMC algorithm is it have to compute the gradient based on the whole data set, which makes the computation intensive given large amount of data, each as in the online-learning scenario. Another limitation of HMC is requiring user specifying at least two parameter: a step size ϵ and a number of steps L to run a simulated Hamiltonian stsyem. Since the HMC is sensitive to the choice of parameter, a very poor choice will result in dramatic drop in HMC's efficiency. In the paper[1], the author proposed a novel method using a minibatch of the original data, as well as a friction term to calculate the gradient and improve convergence. However, this MCMC is irreversible, and hence the "posterior sampling" can not be deemed as a correct MCMC sampling from the target distribution.

The main purpose of this article is to comparing the Hamiltonian Monte Carlo sampler and the No-U-Turn sampler. Naive No-U-Turn Sampler (NUTS) is an extension of HMC that eliminates the need to specify the trajectory length but requires user to specify step size. With dual averaging algorithm NUTS can run without any hand-tuning at all, and samples generated are at-least as good as finely hand-tuned HMC. We addressed dual averaging[2] scheme for automatically tuning the step size, which make it possible to run NUTS without hand-tuning at all. We will show that the tuning-free version of NUTS sampler is more efficiently than HMC.

2 Hamiltonian Monte Carlo Sampler

The simple MCMC algorithms such as random-walk Metropolis Gibbs sampling might require a long time to converge to the target distribution, however, the Hamiltonian Monte Carlo Algorithm can perform well, even when solving the high-dimensional problems.

Here the Hamiltonian dynamics performs on the auxiliary momentum variables r_d, and a d-dimensional variable of interest, θ_d, also is called the particle's position in d-dimensional space. In this report, we choose the zero-mean mutivariate Gaussian distribution for the distribution of the momentum variables, r_d, also, the components of which are specified to be independent. Therefore, the joint density is

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

where $\mathcal{L}$ is the logarithm of the joint density of θ, which is usually called the potential energy, and $\frac{1}{2} r \cdot r$ is interpret as the kinetic energy of the particle.

The HMC algorithm (described in Algorithm 1) embraces mainly two parts for each iteration. The first one is to update the momentum variables only, the second part is to upgrade both momentum variables and position. Both parts of update can keep the same joint distribution of (r_d, θ_d).

In the first part, the momentum variables, r_d are resampled from their Gaussian distribution, which are independent of the current values of the position variables, θ_d.

In the second part, starting with the current momentum and position variable, r and θ, we make use of L steps leapfrog method, with stepsize of *epsilon*, to generate the updated momentum and position, $\tilde{r}$ and $\tilde{\theta}$. Here, L and *epsilon* are both parameters of the algorithm, which need to be tuned to obtain good performance. The momentum variables in the end are needed to be negated, giving a proposed $\tilde{r}$, which leaves the Metropolis proposal symmetrical, the proposed momentum and position variable is accepted as the next step of Markov chain with probability is shown in Algorithm 1. However, in practice, the negation of is not necessary is someone is only interested in sampling from $p(\theta)$.

3 Algorithm

Algorithm 1 Hamiltonian Monte Carlo

Given $\theta^0, \epsilon, \mathcal{L}, L, M$:
for $m = 1$ **to** M do **do**
  Sample $r^0 \sim \mathcal{MVN}(\mathbf{0}, \mathcal{I})$,
  Set $\theta^m \leftarrow \theta^{m-1}, \tilde{\theta} \leftarrow \theta^{m-1}, \tilde{r} \leftarrow r^0$,
  for $i = 1$ **to** L do **do** Set $\tilde{\theta}, \tilde{r} \leftarrow \text{Leapfrog}(\tilde{\theta}, \tilde{r}, \epsilon)$,
 end for
 Calculate the accept probability $\alpha = min\{1, \frac{\exp\{\mathcal{L}(\tilde{\theta}) - \frac{1}{2}\tilde{r} \cdot \tilde{r}}{\exp\{\mathcal{L}(\theta^{m-1}) - \frac{1}{2} r^0 \cdot r^0}\}$,
 Set $\theta^m \leftarrow \tilde{\theta}, r^m \leftarrow -\tilde{r}$
end for

function Leapfrog(θ, r, ϵ)
Set $\tilde{r} \leftarrow r + \frac{\epsilon}{2} \nabla_\theta \mathcal{L}(\theta)$,
Set $\tilde{\theta} \leftarrow \theta + \epsilon \tilde{r}$,
Set $\tilde{r} + \frac{\epsilon}{2} \nabla_\theta \mathcal{L}(\tilde{\theta})$.
return $\tilde{\theta}, \tilde{r}$.

4 No-U-Turn Sampler

4.1 Motivation

One of the biggest advantage of HMC to the regular random walk Metropolis-Hasting sampler is that with the help of additional momentum variable, the HMC can explore the target distribution surface faster in a way that it can give proposal that is distant from current state and yet still be accepted for probability 1 in a ideal situation. In reality, we have to discretize the fictitious time variable added into the system, and we usually use the leapfrog to simulate the Hamiltonian dynamic.

Fortunately, the error introduced by discrete-time simulation doesn't grow without bound, and thus allow us to run many leapfrog steps. Thus the performance of HMC depends highly on the parameter selection on ϵ and L, where ϵ controls the precision of discrete-time simulation of Hamiltonian dynamic and L control the distance of exploration in each iteration. If ϵ is too large, then the resulting discrete-time simulation will be inaccurate and leaving the Hamiltonian dynamic, the reject rate will be high. On the other hand, if ϵ is too small, then we are wasting the computational power.

For the parameter L, if L is too small, then the HMC explores only in small neighborhood of current state, expressing random walk behavior and indicates slow mixing. On the other hand, if L is too large, it is possible that trajectories will retrace their steps, and an analogy will be the case when too large step size is chosen in a gradient descent algorithm, which lead to the name of this sampler, No-U-Turn sampler. The even worse case would be periodicity, since the change in the momentum and position variable can be viewed as sine and cosine function of $L\epsilon$, if we chose a trajectory length such that $L\epsilon = 2\pi$ the at the end of iteration we will return to the starting point.

Figure 1 shows the trace plot of one iteration of HMC sampler for a highly correlated bivarite normal distribution. The opacity and size of dots grows with leapfrog steps, and the black color indicates it is either start or end point. It can be seem that the leap frog already explored the other end of distribution; however, it traces back and returns almost to its current state. To avoid these U-Turn, the author proposed a mechanism to stop when we simulated "long enough".

4.2 Naive No-U-Turn

The general idea is to devise an MCMC sampler that reserve HMC's ability to avoid random walk behavior without the need to set the number L of leapfrog steps that algorithm needed to generated a proposal.[3] To define a criterion that helps us avoid these U-Turn, author first think about the quadratic formula: for a concave function with single maximum, the maximum of its value can be found at point where its derivative equals zero. In the similar fashion, author define the instantaneous distance gain for the proposal $\tilde{\theta}$ and the initial value of θ:

$$\frac{d}{dt}\frac{(\tilde{\theta}-\theta)\cdot(\tilde{\theta}-\theta)}{2} = (\tilde{\theta}-\theta)\cdot\frac{d}{dt}(\tilde{\theta}-\theta) = (\tilde{\theta}-\theta)\cdot\tilde{r} \tag{1}$$

Where $\tilde{r}$ can be considered as the current momentum and the $(\tilde{\theta}-\theta)$ ca be considered as the vector from initial position to the current position. It suggests that we simulate the process until $(\tilde{\theta}-\theta) < 0$ so that further simulation will decrease our distance between our proposal $\tilde{\theta}$ and the initial value of θ, so that we stop the simulation right before the $\tilde{\theta}$ start to move back to θ. However, the problem of this initial thought is that this algorithm does not guarantee time reversibility, so the author devised a different scheme.

In short, the author augments the model with a slice variable u and then add a finite set $\mathcal{C}$ of candidates for the update, and $\mathcal{C} \subseteq \mathcal{B}$, where $\mathcal{B}$ is the set of all leapfrog steps and $\mathcal{C}$ is chosen deterministically from $\mathcal{B}$. A high level NUTS steps will be at each iteration:

- resample momentum $r \sim \mathcal{N}(o, I)$
- sample $u|\theta, r \sim \mathcal{U}[0, \exp\{\mathcal{L}(\theta) - \frac{1}{2}r\cdot r\}]$

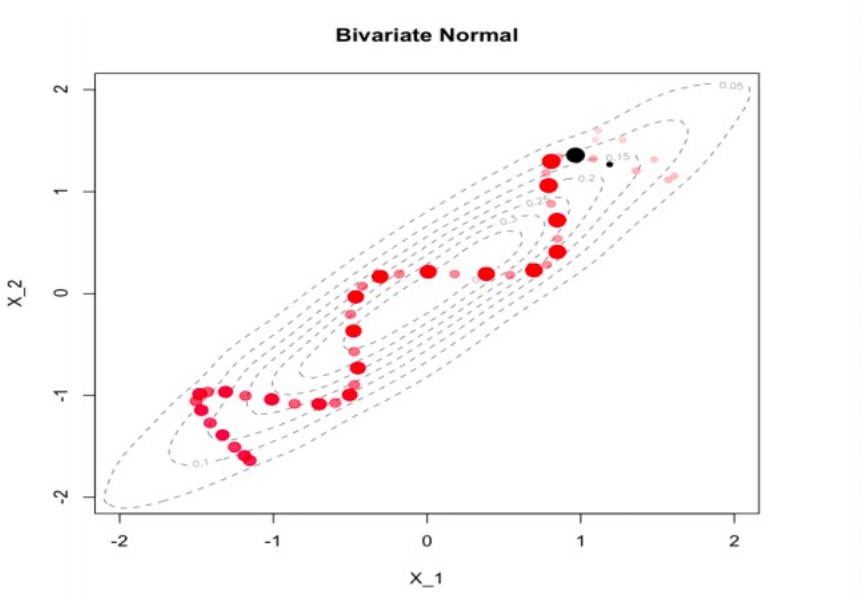


Figure 1: This is a trace plot of one iteration of HMC sampler for a highly correlated bivarite normal distribution. The opacity and size of dots grows with leapfrog steps, and the black color indicates it is either start or end point. It can be seem that the leap frog already explored the other end of distribution; however, it traces back and returns almost to its current state.

- generate the proposal from $p(\mathcal{B}, \mathcal{C}|\theta, r, u, \epsilon)$
- sample $(\theta_{t+1}, r) \sim T(\cdot|\theta, r, \mathcal{C})$

NUTS first augments the model $p(\theta, r) \propto \exp\{\mathcal{L}(\theta) - \frac{1}{2}r \cdot r\}$ with a lice variable u, where θ is the variable of interest, or the position variable, and the r is the variable for HMC, or the momentum variable. Then the joint probability of θ, r, and u is

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

where $\mathbb{1}[\cdot]$ is 1 if the expression inside the brackets is true and 0 if else. Then we can easily show that the unnormalized marginal probability of θ and r is

$$p(\theta, r) \propto \exp\{\mathcal{L}(\theta) - \frac{1}{2}r \cdot r\}. \tag{3}$$

using the fact that u is a uniform between 0 and $\exp\{\mathcal{L}(\theta) - \frac{1}{2}r \cdot r\}$.

$\mathcal{B}$ will be built by randomly taking forward and backward leapfrog steps, and $\mathcal{C}$ will be chosen deterministically from

$$\mathcal{B}$$

. Then the author imposed four conditions on the conditional distribution $p(\mathcal{B}, \mathcal{C}|\theta, r, u, \epsilon)$:

- C.1: All elements of C must be chosen in a way that preserves volumn. That is, any deterministic transformations of θ, r used to add a state θ', r' to $\mathcal{C}$ must have a Jacobian with unit determinant.
- C.2: $p((\theta, r) \in \mathcal{C}|\theta, r, u, \epsilon) = 1$
- C.3: $p(u \leq \exp\{\mathcal{L}(\theta') - \frac{1}{2}r' \cdot c'\}|(\theta', r') \in \mathcal{C}) = 1$.
- C.4: If $(\theta, r) \in \mathcal{C}$ and $(\theta', r') \in \mathcal{C}$ then for any $\mathcal{B}$,$p(\mathcal{B}, \mathcal{C}|\theta, r, u, \epsilon) = p(\mathcal{B}, \mathcal{C}|\theta', r', u, \epsilon)$

where C.1 ensures that within the $\mathcal{C}$, the volume is preserved and thus the dynamic is preserved. Then C.2 indicates that the current state will always be inside the set $\mathcal{C}$. C.3 requires that any state in the set must have equal conditional probability. Finally, C.4 states that $\mathcal{B}$ and $\mathcal{C}$ must have equal probability regardless of states within $\mathcal{C}$. From all conditions, we can show that:

$$\begin{aligned} p(\theta, r|u, \mathcal{B}, \mathcal{C}, \epsilon) &\propto p(\mathcal{B}, \mathcal{C}|\theta, r, u, \epsilon)p(\theta, r|u) \\ &\propto p(\mathcal{B}, \mathcal{C}|\theta, r, u, \epsilon)\mathbb{1}_{u \leq \exp\{\mathcal{L}(\theta') - \frac{1}{2}r' \cdot c'\}} \quad \text{by C.1} \\ &\propto \mathbb{1}_{\mathcal{C}} \quad \text{by C.2-4} \end{aligned}$$

The above calculation indicates all states in set $\mathcal{C}$ can be candidate for proposal with equal probability and thus we can draw sample from set $\mathcal{C}$ as the proposal.

The author propose to build $\mathcal{B}$ by repeatly doubling a binary tree with (θ, r) leaves.

- choose a random "direction in time" $v_j \sim \mathcal{U}(\{-1, 1\})$
- take 2^j leapfrogs steps of size $v_j\epsilon$ from

$$(\theta^-, r^-)\mathbb{1}_{\{v_j=-1\}} + (\theta^+, r^+)\mathbb{1}_{\{v_j=1\}}$$

- continue until a stopping rule is met

With that setup, the resulting set $\mathcal{B}$ is reversible in time, and in addition to that, given the start (θ, r) and ϵ, there are 2^j equally probable height-j trees, and thus reconstructing a particular height-j tree from any leaf has probability 2^{-j}. As for the stop criteria, the author proposed two conditions at each height j:

- for one of the $2^j - 1$ subtrees $(\theta^+ - \theta^-) \cdot r^- < 0$ or $(\theta^+ - \theta^-) \cdot r^+ < 0$
- $\mathcal{L}(\theta) - \frac{1}{2}r \cdot r - \log(u) < -\triangle_{\max}$

The first condition ensures that we have search long enough, but not taking a U-Turn yet, and the second condition states that we should not go too far where our simulation becomes inaccurate. As for the $\triangle_{\max}$, the author suggests some large value like 1000 to ensure the simulation is moderately accurate.

Now, we can check the conditions for the conditional distribution $p(\mathcal{B}, \mathcal{C}|\theta, r, u, \epsilon)$:

- C.1 is satisfied because of the we use the leapfrog to integrate the time.
- C.2 is satisfied as $\mathcal{C}$ includes the initial state.
- C.3 is satisfied as we exclude the points outside the slice U
- C.4 is satisfied as we exclude states that couldn't generate $\mathcal{B}$ as long as the mechanism of generating $\mathcal{C}$ is deterministic.

After checking the above condition, we can conclude that the algorithm conserved the detailed-balance, and become our naive NUTS.

4.3 Efficient NUTS

Although the computation cost per-leapfrog step is comparable with the regular HMC, the NUTS requires to store 2^j position-momentum states for the last draw, and long jumps are also not guaranteed as result of the draw. Furthermore, it would be a waste of time if a stopping criterion is met during doubling. To solve the doubling issue, we will break out of the loop as soon as the stopping criteria is met. To solve the problem where the long jumps are not guaranteed, author proposed the following kernel:

$$T(w'|w, \mathcal{C}) = \begin{cases} \frac{\mathbb{1}\left[w' \in \mathcal{C}^{new}\right]}{|\mathcal{C}^{new}|} & \text{if } |\mathcal{C}^{new}| > |\mathcal{C}^{old}| \\ \frac{\mathbb{1}\left[w' \in \mathcal{C}^{new}\right]}{|\mathcal{C}^{new}|}\frac{|\mathcal{C}^{new}|}{|\mathcal{C}^{old}|} + (1 - \frac{|\mathcal{C}^{new}|}{|\mathcal{C}^{old}|})\mathbb{1}\left[w' = w\right] & \text{if } |\mathcal{C}^{new}| \leq |\mathcal{C}^{old}| \end{cases}$$

where w is short for (θ, r) and let $\mathcal{C}^{new}$ and $\mathcal{C}^{old}$ be respectively for the set by the final iteration and the older elements already inside. So T propose to the new state by their relative size ratio. So in this way, the detailed balance is satisfied and therefore leaves the uniform distribution on $\mathcal{C}$ constant. To solve the last issue of memory take, the author suggests apply T after every doubling, proposing a move to each new half-tree in turn. In this way, we only need to store $O(j)$ position instead of $O(2^j)$.

4.4 Adaptively Tuning ϵ

How to adjust the step size parameter ϵ is another critical issue. Since there is no single accept/reject steps in NUTS, an alternative statistics H_t^{NUTS} is defined to Metropolis acceptance probability. For each iteration, the statistics H_t^{NUTS} and its expectation when the chain has reached equilibrium as

$$H_t^{NUTS} \equiv \frac{1}{|\mathcal{B}_t^{final}|} \sum_{\theta,r \in \mathcal{B}_t^{final}} min\{1, \frac{p(\theta, r))}{P(\theta^{t-1}, r^{t,0})}\}; h^{NUTS} \equiv E_t[H_t^{NUTS}] \tag{4}$$

where $\mathcal{B}_t^{final}$ is the set of all states explored during the final doubling of iteration t of the Markov chain.θ^{t-1} and $r^{t,0}$ are the initial position and (resampled)momentum for the tth iteration of the Markov chain. H_t^{NUTS} represents the average acceptance probability that HMC would give to position-momentum states explored during the final doubling iteration.

Assuming that H_t^{NUTS} is nonincreasing in ϵ, we can apply the updates

$$x_{t+1} \leftarrow \mu - \frac{\sqrt{t}}{\gamma} \frac{1}{t + t_0} \sum_{i=1}^{t} H_i; \bar{x}_{t+1} \leftarrow \eta_t x_{t+1} + (1 - \eta_t)\bar{x}_t \tag{5}$$

with $H_t \equiv \ delta - H^{NUTS}$ and $x \equiv log\epsilon$ to coerce $h^{NUTS} = \delta$ for any $\delta \in (0, 1)$. where μ is a freely chosen point that the iterates x_t are shrunk toward, $\gamma > 0$ is a free parameter that stabilizes the initial iterations of the algorithm, $\eta_t \equiv t^{-k}$ is a step size schedule obeying the condition of

$$\sum_t \eta_t = \infty; \sum_t \eta_t^2 < \infty \tag{6}$$

Addressing parameter is particularly important in $MCMC$, especially for NUTS. Setting $t_0 > 0$ improves the stability of the algorithm in early iterations, which prevents us from wasting computation by trying our extreme values. The lower value of ϵ always results in more working being done per sample, so we need to avoid casually trying out extremely low values of ϵ. Setting the parameter k¡1 allows us to give high weight to more recent iterates and more quickly forget the iterates produced during the early burn-in stages.

We applied the Heuristic algorithm for choosing a good initial value of ϵ. This heuristic repeatedly doubles or halves the value of $\epsilon 1$ until the acceptance probability of the langevin proposal with step size $\epsilon 1$ cross 0.5. The resulting value of $\epsilon 1$ will typically be small enough to produce reasonably accurate simulation but large enough to avoid wasting large amounts of computation. It is recommended setting $\mu = log(10\epsilon 1)$, since this gives the dual averaing algorithm a preference for testing values of ϵ that are larger than the initial value $\epsilon 1$. Large values of ϵ cost less to evaluate than small values of ϵ, and so erring on the side of trying large values can save computation.The below Algorithms show NUTS incorporating the dual averaing algorithm derived in this section, with the above initialization scheme. This algorithm requires only a target mean acceptance probability δ, and a number of iterations M^{adapt}.

5 Algorithm

Algorithm 2 No-U-Turn Sampler with Dual Averaging

Given $\theta^0, \delta, \mathcal{L}, M, M^{adapt}$:
Set $\epsilon_0 =$ Find ReasonableEpsilon(θ),$\mu = \log(10\epsilon_0), \bar{\epsilon}_0 = 1, \bar{H}_0 = 0, \gamma = 0.05, t_0 = 10, \kappa = 0.75$.
for $m = 1$ **to** M **do**
  Sample $r^0 \sim \mathcal{N}(0, I)$
  Resample $u \sim$ Uniform($[0, \exp\{\mathcal{L}(\theta^{m-1}) - \frac{1}{2} r^0 \cdot r^0\}]$)
  Initialize $\theta^- = \theta^{m-1}, \theta^+ = \theta^{m-1}, r^- = r^0, r^+ = r^0, j = 0, \theta^m = \theta^{m-1}, n = 1, s = 1$
  while $s = 1$ **do**
    Choose a direction $v_j \sim$ Uniform($\{-1, 1\}$)
    if $v_j = -1$ **then**
      $\theta^-, r^-, -, -, \theta', n', s', \alpha, n_\alpha \leftarrow$ BuildTree$(\theta^-, r^-, u, v_j, j, \epsilon_{m-1}\theta^{m-1}, r^0)$
    else
      $-, -, \theta^+, r^+, \theta', n', s', \alpha, n_\alpha \leftarrow$ BuildTree$(\theta^+, r^+, u, v_j, j, \epsilon_{m-1}\theta^{m-1}, r^0)$
    end if
    if $s' = 1$ **then**
      With probability $\min\{1, \frac{n'}{n}\}$,set $\theta^m \leftarrow \theta'$
    end if
    $n \leftarrow n + n'$
    $s \leftarrow s' \mathbb{1}[(\theta^+ - \theta^-) \cdot r^- \geq 0] \mathbb{1}[(\theta^+ - \theta^-) \cdot r^+ \geq 0]$
    $j \leftarrow j + 1$
  end while
  if $m \leq M^{\text{adapt}}$ **then**
    $\bar{H}_m = \left(1 - \frac{1}{m + t_0}\right) \bar{H}_{m-1} + \frac{1}{m + t_0}(\delta - \frac{\alpha}{n_\alpha})$
    set $\log\epsilon_m = \mu - \frac{\sqrt{m}}{\gamma} \bar{H}_m, \log\bar{\epsilon} = m^{-\kappa}\log\epsilon_m + (1 - m^{-\kappa})\log\bar{\epsilon}_{m-1}$
  else
    set $\epsilon_m = \bar{\epsilon}_{M^{\text{adapt}}}$
  end if
end for
function BuildTree$(\theta, r, u, v, j, \epsilon, \theta^0, r^0)$
if $j = 0$ **then**
  Base case-take one leapfrog step in the direction v
  $\theta', r' \leftarrow Leapfrog(\theta, r, v\epsilon)$
  $n' \leftarrow \mathbb{1}[u \leq \exp\{\mathcal{L}(\theta') - \frac{1}{2} r^0 \cdot r^0\}]$
  $s' \leftarrow \mathbb{1}[u \leq \exp\{\triangle_{\max} + \mathcal{L}(\theta') - \frac{1}{2} r^0 \cdot r^0\}]$
  return $\theta', r', \theta', r', \theta', n', s', \min\{1, \exp\{\mathcal{L}(\theta') - \frac{1}{2} r' \cdot r' - \mathcal{L}(\theta^0) + \frac{1}{2} r^0 \cdot r^0\}\}, 1$
else
  Recursion-implicitly build the left and right subtrees.
  $\theta^-, r^-, \theta^+, r^+, \theta', n', s', \alpha', n'_\alpha \leftarrow$ BuildTree$(\theta, r, u, v, j - 1, \epsilon, \theta^0, r^0)$
  if $s' = 1$ **then**
    if $v = -1$ **then**
      $\theta^-, r^-, -, -, \theta'', n'', s'', \alpha'', n''_\alpha \leftarrow$ BuildTree$(\theta^-, r^-, u, v, j - 1, \epsilon, \theta^0, r^0)$
    else
      $-, -, \theta^+, r^+, \theta'', n'', s'', \alpha'', n''_\alpha \leftarrow$ BuildTree$(\theta^+, r^+, u, v, j - 1, \epsilon, \theta^0, r^0)$
    end if
    With probability $\frac{n''}{n' + n''}$, set $\theta' \leftarrow \theta''$
    Set $\alpha' \leftarrow \alpha' + \alpha'', n'_\alpha \leftarrow n'_\alpha + n''_\alpha$
    $n' \leftarrow n' + n''$
  end if
  return $\theta^-, r^-, \theta^+, r^+, \theta', n', s', \alpha', n'_\alpha$
end if

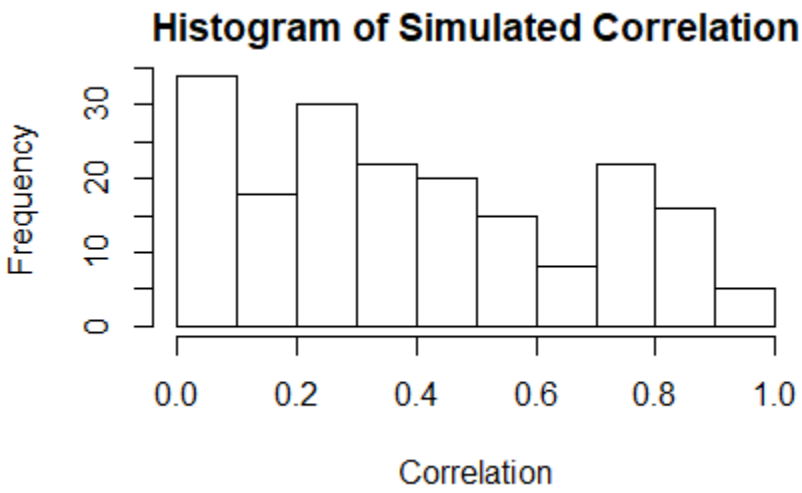


Figure 2: The histogram of correlation of the simulated covariance matrix shows more than half of correlations higher than 0.4 and there are large number of correlations high than 0.7.

6 Comparison Between NUTS and HMC

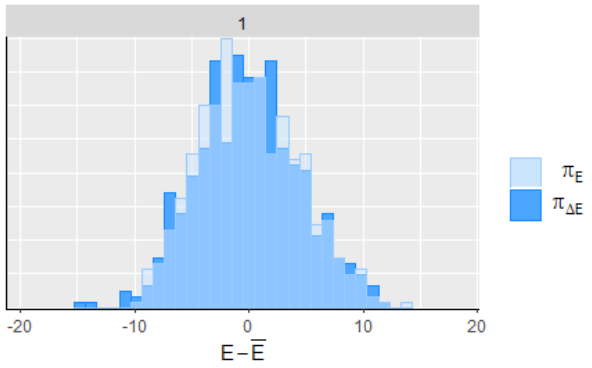
For example and comparison between NUTS and HMC, we simulated multivariate normal distribution with known covariance matrix. After many trials, we decided to use 20-dimensional highly correlated multivariate normal distribution with known covariance matrix due to limitation of computation resources. We generated covariance matrix with high correlation using cholesky factorization targeting at least half of correlation larger than 0.4. Figure 2 is the histogram of correlation of the simulated covariance matrix shows more than half of correlations higher than 0.4 and there are large number of correlations high than 0.7. Using the covariance matrix, we generated 1000 20-dimensional multivariate normal sample with mean 0 using *mvtnorm* in R. For the model fitting, we assumed covariance matrix known and putted a standard multivariate normal distribution on the mean. For the M matrix, we use the identical matrix as standard. After we drawed sample, we used bayesplot package to analyze the MCMC sampling.

The first difference we noticed is the difference in running time. It took only 194.649 second for the NUTS to simulate 1000 iterations; however, it took 479.541 seconds for the HMC. Figure 3 shows the energy plot for two methods. Both NUTS and HMC utilized the Hamiltonian dynamic to search smoothly across the sample surface. The only difference is that NUTS find adpatively find the steps taken in each iteration and find the average steps using dual averaging method. Both method still deviates from the true system as the result of the discretizing the time. The figure indicates the NUTS preserve the Hamiltonian dynamic total energy better as of the tighter spread.

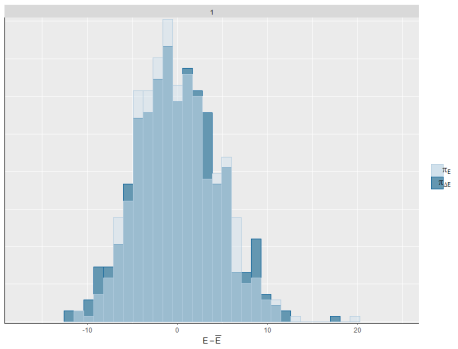
When assessing the performance of Bayesian algorithm, one important marker will be effective sample size (ESS), which measures the reliability of our sample estimation by transforming the number of total size into the number of independent sample size that can generate similar credibility. Suppose that the ρ_t is the correlation at lag t, N is sample of size, the definition of ESS, N_{ess}, is

$$N_{ess} = \frac{N}{\sum_{t=-\infty}^{\infty} \rho_t} = \frac{N}{1 + 2\sum_{t=-\infty}^{\infty} \rho_t}.$$

One way to estimate the ESS is based on the variograms, V_t, at lag $t \in 0, 1...$, which is defined as follows for samples $\theta_m^(n)$, where $m \in 1, ..., M$ is the chain, N_m is the number of samples in chain m: $V_t = \frac{1}{M}\sum_{m=1}^{M}(\frac{1}{N_m - t}\sum_{n=t+1}^{N_m}(\theta_m^{(n)} - \theta_m^{(n-t)})^2)$., leaving the estimated autocorrelations $\hat{\rho}_t = 1 - \frac{V_t}{2\hat{var}^+}$, $\hat{var}^+$

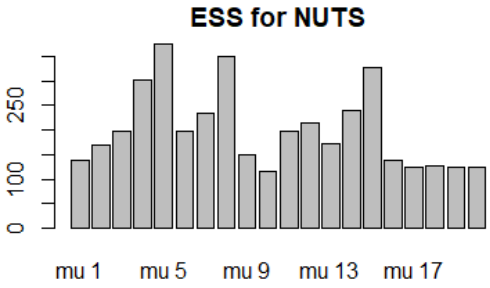


(a) Energy Plot for NUTS

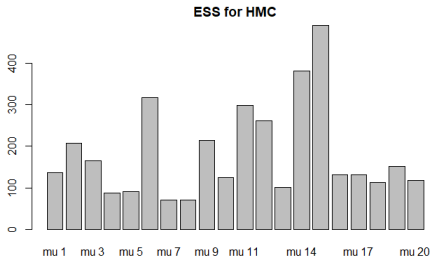


(b) Energy Plot for HMC

Figure 3: Energy plot for two methods shows the NUTS preserve the Hamiltonian dynamic total energy better as of the tighter spread.



(a) ESS Plot for NUTS



(b) ESS Plot for HMC

Figure 4: ESS for each coordinates. The effective sample sizes of NUTS are generally higher than those of HMC.

is the multi-chain variance estimate, the estimation of ESS is given by

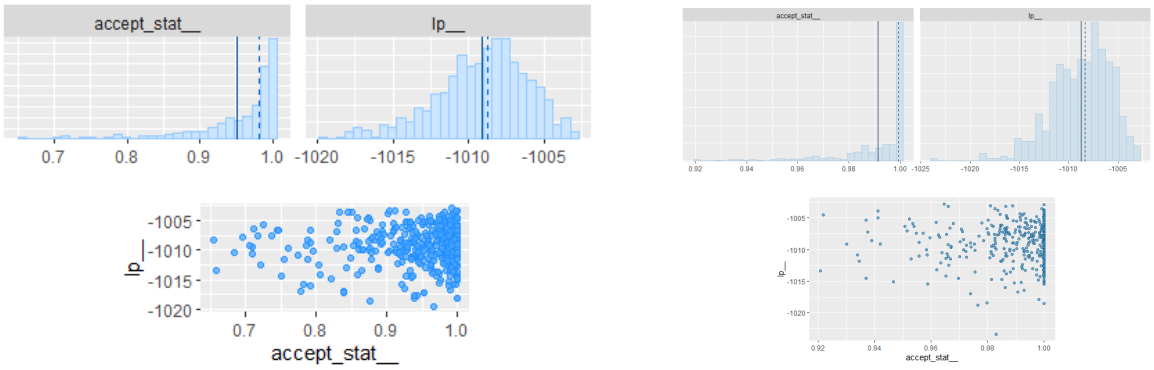
$$\hat{n}_{eff} = \frac{mn}{1 + 2\sum_{t=T}^{T} \hat{\rho}_t}.$$

Figure 4 show the effective sample size for each coordinate. All coordinates of NUTS has ESS higher than 150 with many higher than 300, which is better than that of HMC.

The last comparison will be on the acceptance ratio, since both methods should give proposal with high acceptance ratio as the result of the Hamiltonian dynamics. Figure 5 show the acceptance plots for both method, the NUTS method maintains the majority of acceptance ratio higher than 0.9 with mean acceptance ratio around 0.95 and median even higher. On the other hand, we see higher acceptance rate for the HMC. Such difference in acceptance ratio is due the fact that NUTS introduced target acceptance rate, where smaller target acceptance rate will force smaller stepsize so to overcome certain modeling difficulties.

7 Discussion

In summary, we can see that the performance of NUTS is better than first version of HMC and it also saves tremendous effort in tuning the ϵ and L. Right now, there are other extensions on original HMC, jittering



(a) ESS Plot for NUTS

(b) ESS Plot for HMC

Figure 5: Acceptance plots for NUTS and HMC

on the ϵ and scaling HMC that can speed up the algorithm. In future, it is also of interest to explore the performance of extension of HMC compared NUTS. With more computation power, we can further simulation cases in higher dimension and we expected higher severity of this "U-Turn" behaviour.

References

[1] Tianqi Chen, Emily Fox, and Carlos Guestrin. Stochastic gradient hamiltonian monte carlo. pages 1683–1691, 2014.

[2] Ernst Hairer, Christian Lubich, and Gerhard Wanner. Geometric numerical integration illustrated by the störmer–verlet method. *Acta numerica*, 12:399–450, 2003.

[3] 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.

[4] Radford M Neal et al. Mcmc using hamiltonian dynamics. *Handbook of markov chain monte carlo*, 2(11):2, 2011.

A R Code

```
leapfrog_step = function(theta, r, eps, grad_f, M_diag){
  r_tilde <- r + 0.5 * eps * grad_f(theta)
  theta_tilde <- theta + eps * r_tilde / M_diag
  r_tilde <- r_tilde + 0.5 * eps * grad_f(theta_tilde)
  list(theta = theta_tilde, r = r_tilde)
}

joint_log_density = function(theta, r, f, M_diag){
  f(theta) - 0.5*sum(r**2 / M_diag)
}

find_reasonable_epsilon = function(theta, f, grad_f, M_diag, eps = 1, verbose = TRUE){
  r <- rnorm(length(theta), 0, sqrt(M_diag))
  proposed <- leapfrog_step(theta, r, eps, grad_f, M_diag)
  log_ratio <- joint_log_density(proposed$theta, proposed$r, f, M_diag) - joint_log_de
  alpha <- ifelse(exp(log_ratio) > 0.5, 1, -1)
  if(is.nan(alpha)) alpha <- -1
  count <- 1
  while(is.nan(log_ratio) || alpha * log_ratio > (-alpha)*log(2)){
    eps <- 2**alpha * eps
    proposed <- leapfrog_step(theta, r, eps, grad_f, M_diag)
    log_ratio <- joint_log_density(proposed$theta, proposed$r, f, M_diag) - joint_log_
    count <- count + 1
    if(count > 100) {
      stop("Could␣not␣find␣reasonable␣epsilon␣in␣100␣iterations!")
    }
  }
  if(verbose) message("Reasonable␣epsilon␣=␣", eps, "␣found␣after␣", count, "␣steps")
  eps
}

check_NUTS = function(s, theta_plus, theta_minus, r_plus, r_minus){
  if(is.na(s)) return(0)
  condition1 <- crossprod(theta_plus - theta_minus, r_minus) >= 0
  condition2 <- crossprod(theta_plus - theta_minus, r_plus) >= 0
  s && condition1 && condition2
}

build_tree = function(theta, r, u, v, j, eps, theta0, r0, f, grad_f, M_diag, Delta_max
  if(j == 0){
    proposed <- leapfrog_step(theta, r, v*eps, grad_f, M_diag)
    theta <- proposed$theta
    r <- proposed$r
    log_prob <- joint_log_density(theta, r, f, M_diag)
    log_prob0 <- joint_log_density(theta0, r0, f, M_diag)
    n <- (log(u) <= log_prob)
    s <- (log(u) < Delta_max + log_prob)
    alpha <- min(1, exp(log_prob - log_prob0))
    if(is.nan(alpha)) stop()
    if(is.na(s) || is.nan(s)){
      s <- 0
    }
```

```
    if(is.na(n) || is.nan(n)){
      n <- 0
    }
    return(list(theta_minus=theta, theta_plus=theta, theta=theta, r_minus=r,
                r_plus=r, s=s, n=n, alpha=alpha, n_alpha=1))
  } else{
    obj0 <- build_tree(theta, r, u, v, j-1, eps, theta0, r0, f, grad_f, M_diag)
    theta_minus <- obj0$theta_minus
    r_minus <- obj0$r_minus
    theta_plus <- obj0$theta_plus
    r_plus <- obj0$r_plus
    theta <- obj0$theta
    if(obj0$s == 1){
      if(v == -1){
        obj1 <- build_tree(obj0$theta_minus, obj0$r_minus, u, v, j-1, eps, theta0, r0,
        theta_minus <- obj1$theta_minus
        r_minus <- obj1$r_minus
      } else{
        obj1 <- build_tree(obj0$theta_plus, obj0$r_plus, u, v, j-1, eps, theta0, r0, f
        theta_plus <- obj1$theta_plus
        r_plus <- obj1$r_plus
      }
      n <- obj0$n + obj1$n
      if(n != 0){
        prob <- obj1$n / n
        if(runif(1) < prob){
          theta <- obj1$theta
        }
      }
      s <- check_NUTS(obj1$s, theta_plus, theta_minus, r_plus, r_minus)
      alpha <- obj0$alpha + obj1$alpha
      n_alpha <- obj0$n_alpha + obj1$n_alpha

    } else{
      n <- obj0$n
      s <- obj0$s
      alpha <- obj0$alpha
      n_alpha <- obj0$n_alpha
    }
    if(is.na(s) || is.nan(s)){
      s <- 0
    }
    if(is.na(n) || is.nan(n)){
      n <- 0
    }
    return(list(theta_minus=theta_minus, theta_plus=theta_plus, theta=theta,
                r_minus=r_minus, r_plus=r_plus, s=s, n=n, alpha=alpha, n_alpha=n_alpha
  }
}



NUTS <- function(theta, f, grad_f, n_iter, M_diag = NULL, M_adapt = 50, delta = 0.5, m
  theta_trace <- matrix(0, n_iter, length(theta))
```

```
  par_list <- list(M_adapt = M_adapt)
  for(iter in 1:n_iter){
    nuts <- NUTS_one_step(theta, iter, f, grad_f, par_list, delta = delta, max_treedep
    theta <- nuts$theta
    par_list <- nuts$pars
    theta_trace[iter, ] <- theta
  }
  theta_trace
}


NUTS_one_step <- function(theta, iter, f, grad_f, par_list, delta = 0.5, max_treedepth
  kappa <- 0.75
  t0 <- 10
  gamma <- 0.05
  M_adapt <- par_list$M_adapt
  if(is.null(par_list$M_diag)){
    M_diag <- rep(1, length(theta))
  } else{
    M_diag <- par_list$M_diag
  }

  if(iter == 1){
    eps <- find_reasonable_epsilon(theta, f, grad_f, M_diag, eps = eps, verbose = verb
    mu <- log(10*eps)
    H <- 0
    eps_bar <- 1
  } else{
    eps <- par_list$eps
    eps_bar <- par_list$eps_bar
    H <- par_list$H
    mu <- par_list$mu
  }

  r0 <- rnorm(length(theta), 0, sqrt(M_diag))
  u <- runif(1, 0, exp(f(theta) - 0.5 * sum(r0**2 / M_diag)))
  if(is.nan(u)){
    warning("NUTS:␣sampled␣slice␣u␣is␣NaN")
    u <- runif(1, 0, 1e5)
  }
  theta_minus <- theta
  theta_plus <- theta
  r_minus <- r0
  r_plus <- r0
  j=0
  n=1
  s=1
  if(iter > M_adapt){
    eps <- runif(1, 0.9*eps_bar, 1.1*eps_bar)
  }
  while(s == 1){
    # choose direction {-1, 1}
    direction <- sample(c(-1, 1), 1)
    if(direction == -1){
```

```
      temp <- build_tree(theta_minus, r_minus, u, direction, j, eps, theta, r0, f, gra
      theta_minus <- temp$theta_minus
      r_minus <- temp$r_minus
    } else{
      temp <- build_tree(theta_plus, r_plus, u, direction, j, eps, theta, r0, f, grad_
      theta_plus <- temp$theta_plus
      r_plus <- temp$r_plus
    }
    if(is.nan(temp$s)) temp$s <- 0
    if(temp$s == 1){
      if(runif(1) < temp$n / n){
        theta <- temp$theta
      }
    }
    n <- n + temp$n
    s <- check_NUTS(temp$s, theta_plus, theta_minus, r_plus, r_minus)
    j <- j + 1
    if(j > max_treedepth){
      warning("NUTS: Reached max tree depth")
      break
    }
  }
  if(iter <= M_adapt){
    H <- (1 - 1/(iter + t0))*H + 1/(iter + t0) * (delta - temp$alpha / temp$n_alpha)
    log_eps <- mu - sqrt(iter)/gamma * H
    eps_bar <- exp(iter**(-kappa) * log_eps + (1 - iter**(-kappa)) * log(eps_bar))
    eps <- exp(log_eps)
  } else{
    eps <- eps_bar
  }

  return(list(theta = theta,
              pars = list(eps = eps, eps_bar = eps_bar, H = H, mu = mu, M_adapt = M_ad
}


set.seed(5)
n <- 20
a <- 2
A <- matrix(0,nrow=n,ncol=n)
for(i in 1:n){
  A[i,] <- rnorm(n)+a*rnorm(1)
}
A <- A%*%t(A)
D_half <- diag(diag(A)^(-0.5))
C <- D_half%*%A%*%D_half
C.cor <- cov2cor(C)
C.low <- C.cor[lower.tri(C.cor)]
hist(abs(C.low),xlab="Correlation",main="Histogram of Simulated Correlation")

set.seed(5)
library(mvtnorm)
sim <- rmvnorm(100,mean=rep(0,n),sigma=C)
```

```
library(bayesplot)
np <- nuts_params(fit)
lp <- log_posterior(fit)
color_scheme_set("brightblue")
mcmc_nuts_acceptance(np, lp)
mcmc_nuts_stepsize(np, lp)
mcmc_nuts_treedepth(np, lp)
mcmc_nuts_energy(np)
barplot(summary(fit)$summary[-21,9],names.arg = paste("mu",1:20),main="ESS␣for␣NUTS")
```