Implementation of the No U-Turn Sampler

Aben Carrington

Department of Statistics, University of Chicago

May 23, 2024

1 The No-U-Turn Sampler

As we saw in Chapter 8, Hamiltonian Monte Carlo, or Hybrid Monte Carlo, scales better to high-dimensional problems than other MCMC algorithms such as RWMH, Gibbs Sampling, and MALA, yet is often harder to implement than these algorithms due to difficulties in effectively tuning hyperparameters, namely step size ϵ and duration parameter λ. Additionally, computing gradients of high-dimensional target distributions quickly becomes computationally expensive. As we've seen in problem sets, it is possible, and not altogether difficult, to find a "good-enough" ϵ based on a desired acceptance rate. But finding a good choice of λ is more elusive.

Finding an optimal λ, by which we mean an optimal termination condition for a simulation of HMC, is the key objective and benefit of the No-U-Turn Sampler (NUTS) algorithm [2]. This algorithm functions by tracing out the sample trajectory in phase space until the trajectory begins to revert back upon itself; or, in other words, until the sampler makes a "U-Turn", at which point the algorithm terminates. Since the algorithm checks predefined termination conditions, it eliminates the need to hand tune λ. In fact, there also exists a heuristic algorithm for tuning ϵ during the initialization phase of NUTS, which allows for running NUTS with no hand-tuning whatsoever. This yields a significant decrease in computational cost over traditional HMC with no sacrifice in efficiency!

The remainder of this paper is organized as follows. Section 1.1 presents the Naive NUTS algorithm and gives some intuition related to calculating termination conditions. Section 1.2 provides improvements upon the Naive algorithm and presents a more efficient NUTS algorithm utilizing a dual averaging scheme [4]. Section 1.3 compares the performance of NUTS against other canonical MCMC algorithms in the case of a high-dimensional target distribution. Section 1.4 concludes with a discussion and bibliography.

1.1 Naive NUTS

We alter our conventional notation slightly, letting $\theta \in \mathbb{R}^D$ be our initial D-dimensional state vector and $\tilde{\theta} \in \mathbb{R}^D$ be the D-dimensional proposal vector. Given this notation, we can easily represent the scenario where the sampler begins to make a U-Turn as the case where the change in the distance between the proposal and current state vectors becomes negative. Borrowing concepts from physics, we represent this as the case when

$$\frac{\partial}{\partial t}\left(\frac{(\tilde{\theta}-\theta)\cdot(\tilde{\theta}-\theta)}{2}\right) = (\tilde{\theta}-\theta)\cdot\frac{\partial}{\partial t}(\tilde{\theta}-\theta) = (\tilde{\theta}-\theta)\cdot r < 0 \tag{1.1}$$

where r is the current momentum vector.

The termination condition given by the equation above implies a simplistic algorithm whereupon one continues the simulation until the proposal begins to move back towards a previous location. However, such an algorithm is not time reversible, and thus does not guarantee convergence to the correct target distribution. To bypass this issue, a recursive doubling procedure similar in motivation to the one used in slice sampling [3] is used. This doubling procedure is as follows. At each step j, a direction is chosen uniformly at random representing a move forwards or backwards in fictitious time. Then, 2^j leapfrog steps are taken in the chosen direction from the furthest point forwards or backwards in time (corresponding to the chosen direction). As an illustration of this, one possible scenario is that you are given a starting location, then at step 1 you decide to take 1 step forward, then 2 steps forward from the current location, then 4 steps back from the starting location, then 8 steps forward from the starting location of the previous step, etc. The end result is a balanced binary tree of height j. The termination condition of this tree construction is when the sub-trajectory of the leftmost to rightmost nodes of any sub-tree begins to revert upon itself.

To assist in the handling of the termination condition, a slice variable u is introduced under the conditional distribution

$$\mathbb{P}(u \mid \theta, r) = \text{Unif}\left(\left[0, \exp\left(\mathcal{L}(\theta) - \frac{1}{2} r^T \cdot \Sigma^{-1} \cdot r\right)\right]\right)$$

where $\mathcal{L}$ denotes the negative potential energy function and Σ signifies the mass matrix of the kintetic energy. Utilizing the slice variable is not strictly necessary, however it was found in [2] to substantially increase the efficiency of the algorithm.

Before declaring the final termination conditions we introduce the finite sets $\mathcal{B}, \mathcal{C}$, signifying the set of all states that the leapfrog integrator traces during an iteration of NUTS and the subset of those states to which it is possible to transition without violating detailed balance. Note that $\mathcal{C} \subseteq \mathcal{B}$. In the interest of space, we forego a rigorous justification of the possibility and correctness of sampling from the distribution $\mathbb{P}(\mathcal{B}, \mathcal{C} \mid \theta, r, u, \epsilon)$. Interested readers are encouraged to review [2].

We are now in a place to declare the termination conditions of NUTS that are used in the algorithm. As previously stated, we want to continue the tree construction until the trajectory of some sub-tree begins to make a “U-Turn“ as the simulation will begin to retrace its steps and become wasteful beyond that point. We also want to stop the tree construction if the error in the simulation becomes considerably large as this would indicate that any subsequent states explored by the simulation would have a low chance of occurrence. We formalize the first condition by saying that at each step j, NUTS evaluates termination through considering the $2^j - 1$ balanced sub-trees with height greater than 0. We let (θ^-, r^-) and (θ^+, r^+) indicate the states corresponding to the leftmost and rightmost leaves of these sub-trees and terminate construction when either

$$(\theta^+ - \theta^-) \cdot r^- < 0 \text{ or } (\theta^+ - \theta^-) \cdot r^+ < 0. \tag{1.2}$$

The second condition is simpler to formalize. We terminate tree construction if the tree contains a leaf whose state (θ, r) satisfies

$$\mathcal{L}(\theta) - \frac{1}{2} r^T \cdot \Sigma^{-1} \cdot r - \log(u) < -\Delta_{max} \tag{1.3}$$

where Δ_{max} is some non-negative threshold.

Given the above termination conditions we now present the Naive No-U-Turn Sampler.

Algorithm 1.1 Naive No-U-Turn Sampler

```
1: **Given** $\theta^0$, $\epsilon$, $\mathcal{L}$, $M$, $\Sigma$
2: **for** $m = 1$ to $M$ **do**
3:     Resample $r^0 \sim \mathcal{N}(0, I)$
4:     Resample $u \sim \text{Uniform}([0, \exp(\mathcal{L}(\theta^{m-1} - \frac{1}{2} r^0 \cdot \Sigma^{-1} \cdot r^0)])$
5:     Initialize $\theta^- = \theta^{m-1}$, $\theta^+ = \theta^{m-1}$, $r^- = r^0$, $r^+ = r^0$, $j = 0$, $\mathcal{C} = \{(\theta^{m-1}, r^0)\}$, $s = 1$
6:     **while** $s = 1$ **do**
7:         Choose a direction $v_j \sim \text{Uniform}(\{-1, 1\})$
8:         **if** $v_j = -1$ **then**
9:             $\theta^-, r^-, -, -, \mathcal{C}', s' \leftarrow \text{BuildTree}(\theta^-, r^-, u, v_j, j, \epsilon)$
10:        **else**
11:            $-, -, \theta^+, r^+, \mathcal{C}', s' \leftarrow \text{BuildTree}(\theta^+, r^+, u, v_j, j, \epsilon)$
12:        **end if**
13:        **if** $s' = 1$ **then**
14:            $\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{C}'$
15:        **end if**
16:        $s \leftarrow s' \mathbb{1}[(\theta^+ - \theta^-) \cdot r^- \geq 0] \mathbb{1}[(\theta^+ - \theta^-) \cdot r^+ \geq 0]$
17:        $j \leftarrow j + 1$
18:    **end while**
19:    Sample $(\theta^m, r)$ uniformly at random from $\mathcal{C}$
20: **end for**
21: **function** BUILDTREE$(\theta, r, u, v, j, \epsilon)$
22:    **if** $j = 0$ **then**
23:        Base case—take one leapfrog step in the direction $v$
24:        $\theta', r' \leftarrow \text{Leapfrog}(\theta, r, v\epsilon)$
25:        $\mathcal{C}' \leftarrow \{(\theta', r')\}$ if $u \leq \exp(\mathcal{L}(\theta') - \frac{1}{2} r' \cdot \Sigma^{-1} \cdot r')$ else $\{\emptyset\}$
26:        $s' \leftarrow \mathbb{1}\Big[u < \exp(\Delta_{\max} + \mathcal{L}(\theta') - \frac{1}{2} r' \cdot \Sigma^{-1} \cdot r')\Big]$
27:        **return** $\theta', r', \theta', r', C', s'$
28:    **else**
29:        Recursion—build the left and right subtrees.
30:        $\theta^-, r^-, \theta^+, r^+, \mathcal{C}', s' \leftarrow \text{BuildTree}(\theta, r, u, v, j - 1, \epsilon)$
31:        **if** $v = -1$ **then**
32:            $\theta^-, r^-, -, -, \mathcal{C}'', s'' \leftarrow \text{BuildTree}(\theta^-, r^-, u, v, j - 1, \epsilon)$
33:        **else**
34:            $-, -, \theta^+, r^+, \mathcal{C}'', s'' \leftarrow \text{BuildTree}(\theta^+, r^+, u, v, j - 1, \epsilon)$
35:        **end if**
36:        $s' \leftarrow s'' \mathbb{1}[(\theta^+ - \theta^-) \cdot r^- \geq 0] \mathbb{1}[(\theta^+ - \theta^-) \cdot r^+ \geq 0]$
37:        $\mathcal{C}' \leftarrow \mathcal{C}' \cup \mathcal{C}''$
38:        **return** $\theta^-, r^-, \theta^+, r^+, \mathcal{C}', s'$
39:    **end if**
40: **end function**
```

where M is the desired number of samples and $\mathbb{1}$ is the indicator function.

1.2 Efficient NUTS

Observe from the Naive NUTS algorithm that we must calculate $\mathcal{L}(\theta)$ and its gradient $2^j - 1$ times, with another 2^j operations for checking termination conditions. This doesn't seem very cost-effective, but it is actually comparable to the standard HMC algorithm. However Naive NUTS also requires that we store 2^j position and momentum vectors, which could feasibly require an impossible amount of memory. Thus, it is mathematically sound, but a product of wishful thinking, and hence 'Naive'.

We address this issue through the use of an alternative transition kernel. As it was not strictly relevant in the previous section, we neglected to mention that, mathematically speaking, we sampled new states using transition kernel $T(\theta', r' \mid \theta, r, \mathcal{C})$ which was constrained to satisfy

$$\frac{1}{|\mathcal{C}|} \sum_{(\theta,r)\in\mathcal{C}} T(\theta', r' \mid \theta, r, \mathcal{C}) = \frac{\mathbb{1}[(\theta', r') \in \mathcal{C}]}{|\mathcal{C}|}$$

To address the memory inefficiencies of Naive NUTS, we propose the new transition kernel

$$T((\theta', r') \mid (\theta, r), \mathcal{C}) = \begin{cases} \frac{\mathbb{1}[(\theta',r')\in\mathcal{C}^{\text{new}}]}{|\mathcal{C}^{\text{new}}|} & \text{if } |\mathcal{C}^{\text{new}}| > |\mathcal{C}^{\text{old}}|, \\ \frac{\mathcal{C}^{\text{new}}}{|\mathcal{C}^{\text{old}}|} \frac{\mathbb{1}[(\theta',r')\in\mathcal{C}^{\text{new}}]}{|\mathcal{C}^{\text{new}}|} + \left(1 - \frac{|\mathcal{C}^{\text{new}}|}{|\mathcal{C}^{\text{old}}|}\right) \mathbb{1}[(\theta', r') = (\theta, r)] & \text{if } |\mathcal{C}^{\text{new}}| \leq |\mathcal{C}^{\text{old}}|, \end{cases} \tag{1.4}$$

which proposes a move from $\mathcal{C}^{\text{old}}$ to some state in $\mathcal{C}^{\text{new}}$ and accepts the move with probability $\frac{|\mathcal{C}^{\text{new}}|}{|\mathcal{C}^{\text{old}}|}$. Once again we forego a rigorous justification in the interest of space, but it may be shown that this transition kernel only requires that we store $\mathcal{O}(j)$ position and momentum vectors as opposed to $\mathcal{O}(2^j)$. When updating NUTS based on this new kernel, we let $n' = |\mathcal{C}'|$.

We now turn to implementing a dual averaging scheme [4] which builds on the work done by [6]. First define some statistic H_t, which describes some part of the behavior of a given MCMC algorithm at $t \geq 1$, with expectation

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

where x is a tunable parameter of the algorithm. Assuming that h is a decreasing function guarantees the stochastic update

$$x_{t+1} \to x_t - \eta_t H_t$$

causes $h(x_t)$ to converge towards 0 given that η_t has form $\eta_t \equiv t^{-\kappa}$ for $\kappa \in [0.5, 1)$ [6]. We use these results to address the discrepancies in determining optimal values of ϵ for the NUTS algorithm during and following the burn-in period. The form of η_t places more weight on early iterations while we would like it to do the opposite. To correct this, we instead apply the updates

$$x_{t+1} \leftarrow \mu - \frac{\sqrt{t}}{\gamma}\frac{1}{t+t_0}\sum_{i=1}^{t} H_i; \quad \bar{x}_{t+1} \leftarrow \eta_t x_{t+1} + (1-\eta_t)\bar{x}_t, \tag{1.5}$$

where μ is a chosen point that iterates converge towards, $\gamma > 0$ is a free parameter that controls the rate of convergence, $t_0 \geq 0$ is a free parameter that stabilizes the initial iterates of the algorithm, η_t is defined as above, and$\bar{x}_t$ is the sequence of averaged iterates with $\bar{x}_1 = x_1$. We are guaranteed that $\bar{x}_t$ converges to a value such that $h(\bar{x}_t)$ converges to 0. The authors of [2] found strong results using $\gamma = 0.05, t_0 = 10,$ and $\kappa = 0.75$.

As a final step before presenting the improved NUTS algorithm, we define a useful heuristic algorithm for choosing an initial value of ϵ. The algorithm repeatedly doubles or halves ϵ until the acceptance probability of a Langevin proposal with step size ϵ crosses 0.5.

Algorithm 1.2 Heuristic for choosing an initial value of ϵ

```
1: function FindReasonableEpsilon($\theta$)
2:     Initialize $\epsilon = 1$, $r \sim \mathcal{N}(0, I)$
3:     Set $\theta', r' \leftarrow$ Leapfrog($\theta, r, \epsilon$)
4:     $a \leftarrow 2\mathbb{1}\left[\frac{p(\theta', r')}{p(\theta, r)} > 0.5\right] - 1$
5:     while $\left(\frac{p(\theta', r')}{p(\theta, r)}\right)^a > 2^{-a}$ do
6:         $\epsilon \leftarrow 2^a \epsilon$
7:         Set $\theta', r' \leftarrow$ Leapfrog($\theta, r, \epsilon$)
8:     end while
9:     return $\epsilon$
10: end function
```

Using the methods defined above we have devised a feasible implementation of NUTS that eliminates the need to hand-tune ϵ and λ entirely! This updated NUTS algorithm as well as the updated BuildTree function are given below.

Algorithm 1.3 No-U-Turn Sampler with Dual Averaging

```
1: **Given** $\theta^0$, $\delta$, $\mathcal{L}$, $M$, $M_{\text{adapt}}$, $\Sigma$
2: Set $\epsilon_0 = \text{FindReasonableEpsilon}(\theta)$, $\mu = \log(10\epsilon_0)$, $\bar{\epsilon}_0 = 1$, $\bar{H}_0 = 0$, $\gamma = 0.05$, $t_0 = 10$, $\kappa = 0.75$
3: **for** $m = 1$ to $M$ **do**
4:     Sample $r^0 \sim \mathcal{N}(0, I)$
5:     Resample $u \sim \text{Uniform}([0, \exp(\mathcal{L}(\theta^{m-1} - \frac{1}{2} r^0 \cdot \Sigma^{-1} \cdot r^0)])$
6:     Initialize $\theta^- = \theta^{m-1}$, $\theta^+ = \theta^{m-1}$, $r^- = r^0$, $r^+ = r^0$, $j = 0$, $\theta^m = \theta^{m-1}$, $n = 1$, $s = 1$
7:     **while** $s = 1$ **do**
8:         Choose a direction $v_j \sim \text{Uniform}(\{-1, 1\})$
9:         **if** $v_j = -1$ **then**
10:            $\theta^-, r^-, -, -, \theta', n', s', \alpha, n_\alpha \leftarrow \text{BuildTree}(\theta^-, r^-, u, v_j, j, \epsilon_{m-1}, \theta^{m-1}, r^0)$
11:        **else**
12:            $-, -, \theta^+, r^+, \theta', n', s', \alpha, n_\alpha \leftarrow \text{BuildTree}(\theta^+, r^+, u, v_j, j, \epsilon_{m-1}, \theta^{m-1}, r^0)$
13:        **end if**
14:        **if** $s' = 1$ **then**
15:            With probability $\min\{1, \frac{n'}{n}\}$, set $\theta^m \leftarrow \theta'$
16:        **end if**
17:        $n \leftarrow n + n'$
18:        $s \leftarrow s' \mathbb{1}[(\theta^+ - \theta^-) \cdot r^- \geq 0] \mathbb{1}[(\theta^+ - \theta^-) \cdot r^+ \geq 0]$
19:        $j \leftarrow j + 1$
20:    **end while**
21:    **if** $m \leq M_{\text{adapt}}$ **then**
22:        $\bar{H}_m \leftarrow (1 - \frac{1}{m+t_0}) \bar{H}_{m-1} + \frac{1}{m+t_0}(\delta - \frac{\alpha}{n_\alpha})$
23:        $\log(\epsilon_m) \leftarrow \mu - \frac{\sqrt{m}}{\gamma} \bar{H}_m$, $\log(\bar{\epsilon}_m) = m^{-\kappa} \log(\epsilon_m) + (1 - m^{-\kappa}) \log(\bar{\epsilon}_{m-1})$
24:    **else**
25:        $\epsilon_m \leftarrow \bar{\epsilon}_{M_{\text{adapt}}}$
26:    **end if**
27: **end for**
```

Algorithm 1.4 BuildTree for No-U-Turn Sampler with Dual Averaging

```
1: function BuildTree($\theta, r, u, v, j, \epsilon, \theta^0, r^0$)
2:     if $j = 0$ then
3:         Base case—take one leapfrog step in the direction $v$
4:         $\theta', r' \leftarrow$ Leapfrog($\theta, r, v\epsilon$)
5:         $n' \leftarrow \left[u \leq \exp(\mathcal{L}(\theta') - \frac{1}{2}r' \cdot \Sigma^{-1} \cdot r')\right]$
6:         $s' \leftarrow \left[u < \exp(\Delta_{\max} + \mathcal{L}(\theta') - \frac{1}{2}r' \cdot \Sigma^{-1} \cdot r')\right]$
7:         return $\theta', r', \theta', r', \theta', n', s', \min\Big\{1, \exp\Big(\mathcal{L}(\theta') - \frac{1}{2}r' \cdot \Sigma^{-1} \cdot r' - \mathcal{L}(\theta^0) + \frac{1}{2}r^0 \cdot \Sigma^{-1} \cdot r^0\Big)\Big\}, 1.$
8:     else
9:         Recursion—implicitly build the left and right subtrees
10:        $\theta^-, r^-, \theta^+, r^+, \theta', n', s', \alpha', n'_\alpha \leftarrow$ BuildTree($\theta, r, u, v, j-1, \epsilon, \theta^0, r^0$)
11:        if $s' = 1$ then
12:            if $v = -1$ then
13:                $\theta^-, r^-, -, -, \theta'', n'', s'', \alpha'', n''_\alpha \leftarrow$ BuildTree($\theta^-, r^-, u, v, j-1, \epsilon, \theta^0, r^0$)
14:            else
15:                $-, -, \theta^+, r^+, \theta'', n'', s'', \alpha'', n''_\alpha \leftarrow$ BuildTree($\theta^+, r^+, u, v, j-1, \epsilon, \theta^0, r^0$)
16:            end if
17:            With probability $\frac{n''}{n'+n''}$, set $\theta' \leftarrow \theta''$
18:            Set $\alpha' \leftarrow \alpha' + \alpha''$, $n'_\alpha \leftarrow n'_\alpha + n''_\alpha$
19:            $s' \leftarrow s''\mathbb{1}[(\theta^+ - \theta^-) \cdot r^- \geq 0]\mathbb{1}[(\theta^+ - \theta^-) \cdot r^+ \geq 0]$
20:            $n' \leftarrow n' + n''$
21:        end if
22:        return $\theta^-, r^-, \theta^+, r^+, \theta', n', s', \alpha', n'_\alpha$
23:    end if
24: end function
```

1.3 Comparative NUTS Performance

We will now compare the performance of NUTS against other canonical MCMC algorithms in generating samples from a high-dimensional target distribution, as well as its performance against a reasonable (but not hand-tuned) HMC implementation. The first of these comparisons was previously done by [2] in MATLAB, and we are replicating their results using a Python implementation.

For our target distribution, we are considering a zero-mean, 250-dimensional multivariate normal with a known precision matrix, A, generated from a Wishart distribution with identity scale matrix and 250 degrees of freedom. We expect strongly correlated draws from this target distribution.

To compare performance accuracy we collect 1000 samples from NUTS, RWMH, and a version of Gibbs Sampling adapted to our scenario. In similar fashion to [2], we run NUTS with $\delta = 0.5$ for 2,000 iterations using a burn-in of 1,000 iterations, which requires roughly 1,000,000 $\mathcal{L}$ and gradient calculations in total. To match this, we run RWMH and Gibbs for 1,000,000 iterations and thin collected samples down to 1,000 using every 999th sample after the burn-in period of 1,000 iterations. For

RWMH, we use a normal proposal distribution with standard deviation $\sigma = 2.38/\sqrt{250}$ in accordance with the results of [1]. Once samples are collected we plot the first and second dimension of the samples against each other to determine the accuracy of the results. These results are shown in Figure 1.

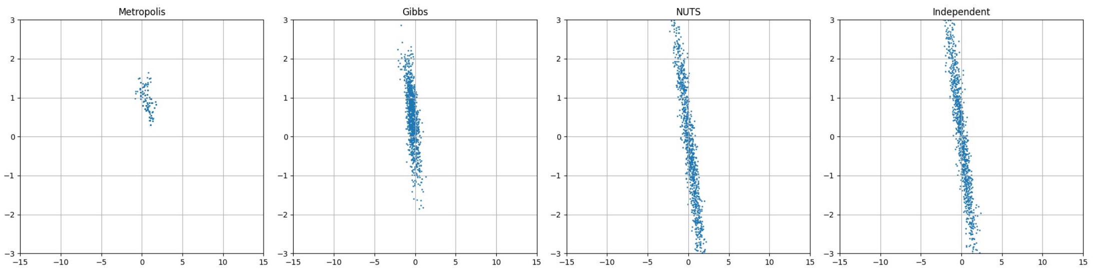

Figure 1 Results of simulation generating 1,000 samples to approximate 250-dimensional multivariate normal target distribution. This figure attempts to replicate a figure from [2].

The results show that RWMH struggles to leave its initialization, and does a very poor job of approximating the target distribution. The results are stronger with Gibbs Sampling, although the sampler does not adequately explore regions of the state space far from its initialization. NUTS approximates the target distribution nearly exactly.

We now turn to comparing the performance of NUTS against a standard implementation of HMC. We know that it is possible to approximate the results of NUTS through hand-tuning ϵ and λ, so to do so would not be very interesting. What we investigate here is the question of exactly how much better NUTS is compared to standard HMC when we use seemingly reasonable values for parameters ϵ and λ. To this end, we choose $\lambda = 10$ and $\epsilon = 0.01$, which showed up quite a bit on problem sets. The results of this simulation are shown in Figure 2 below.

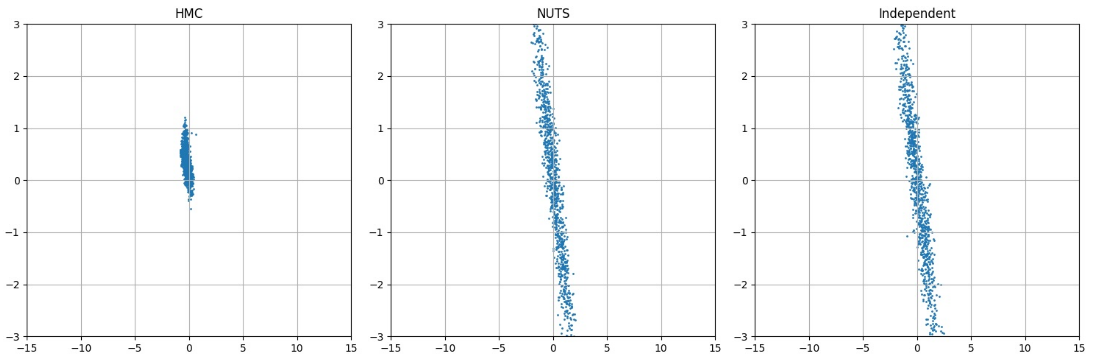

Figure 2 Results of simulation generating 1,000 samples to approximate 250-dimensional multivariate normal target distribution. This simulation investigates the computational accuracy improvement of NUTS over pre-chosen values for standard HMC.

The difference is fairly staggering, as the standard HMC implementation resoundingly fails to explore the state space. Granted, we chose a fairly small value for ϵ, but the results of the simulation still show how substantial of an improvement NUTS is over standard HMC.

All code to produce Figures 1 and 2 is provided in the Python Notebook accompanying this paper.

1.4 Discussion and Bibliography

In this paper we have discussed the NUTS algorithm and its improvements over standard Hamiltonian Monte Carlo, and seen simulation results which corroborate these improvements. Much of this paper is based on the work done in [2], the original paper on NUTS, with supplementary background material provided in the remainder of the references.

Some work has been done on investigating the efficiency of NUTS in different disciplines [5], but overall the scholarship on the topic is very light. Current implementations for the NUTS algorithm exist in pyMC, TensorFlow Probability, and Stan.

References

[1] A. Gelman, G. O. Roberts, and W. R. Gilks. Efficient metropolis jumping rules. In J. M. Bernardo, J. O. Berger, A. P. Dawid, and A. F. M. Smith, editors, *Bayesian Statistics*, pages 599–608. Oxford University Press, Oxford, 1996.

[2] M. D. Hoffman and A. Gelman. The no-u-turn sampler: Adaptively setting path lengths in hamiltonian monte carlo, 2011.

[3] R. M. Neal. Slice sampling. *The Annals of Statistics*, 31(3):705 – 767, 2003.

[4] Y. Nesterov. Primal-dual subgradient methods for convex problems. *Mathematical Programming*, 120(1):221–259, 2009.

[5] M. Nishio and A. Arakawa. Performance of hamiltonian monte carlo and no-u-turn sampler for estimating genetic parameters and breeding values. *Genetics Selection Evolution*, 51(1), 12 2019. ISSN 1297-9686. doi: 10.1186/s12711-019-0515-1.

[6] H. Robbins and S. Monro. A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, 22(3):400 – 407, 1951.