# HMC: Learner's Note

Zehaan Naik

May 2025

## 1 Introduction

Markov Chain Monte Carlo (MCMC) is a class of algorithms that allows us to generate random samples from complex probability distributions. With the growing popularity of complex probabilistic ML architectures and Bayesian methods, MCMC is becoming more and more prominent in the literature. However, due to the advanced nature of its fundamentals, MCMC is often used as one of many black-box steps in the architecture and its subtle theoretical foundations are frequently ignored. This note aims to give a student's perspective (mine!) on the theoretical foundations of one such popular algorithm standard in the literature. I am writing this note to describe the steps I took to understand the concept. I hope this article is easy to follow and insightful.

This note assumes a basic knowledge of probability theory and stochastic processes on the reader's part. However, I will cite the necessary resources when I skip details on results. I also do not discuss some theoretical fundamentals of MCMC, namely, ergodicity and rate of convergence for the algorithm (look at Takeda and Sakajo (2023)). However, I will discuss intuitive "thumb rules" that one can be mindful of during implementations for the "best" results. Lastly, I will try to provide as many visualisations as possible to support the theoretical details and help with intuition building.

In this note, I talk about Hamiltonian Monte Carlo (HMC), an MCMC algorithm that uses Hamiltonian dynamics to increase sampling efficiency (Neal et al., 2011). I will discuss the fundamentals of MCMC, namely, invariance and acceptance rate, which explain why what we are doing makes sense. I also explain Hamiltonian dynamics from scratch and how they tie into our area of interest. Finally, I prove why HMC is an algorithm that helps us solve a core problem with MCMC sampling and why it is one of the more powerful algorithms within the MCMC structure.

## 2 Background

The first section explores the core concepts in statistics that develop our understanding and context of the significant algorithms we undertake later in our exploration. In this section, I aim to analyse Markov Chains, kernels, and their properties, which help build a foundation for understanding HMC. Our general strategy in MCMC is to generate a mechanism that generates a random number given a current state $x$. This is a Markov structure; hence, understanding the properties of continuous space Markov chains helps us know the distributional properties of the numbers we generate.

## 2.1 Markov Chains

A Markov chain or Markov process is a stochastic process describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event. We can describe these events as a chain of random variables $X_1, \ldots, X_n$ for a discrete-time process with the Markov Property defined as:

$$Pr(X_{n+1} \mid X_1, \ldots, X_n) = Pr(X_{n+1} \mid X_n).$$

We shall also assume this process to be stationary, i.e. $\exists$ a probability distribution $\pi$:

$$X_{n+1} \mid X_n \sim \pi \ \forall n.$$

Next, I define Markov transition kernels. A transition kernel (or simply, kernel) is a map

$$P : \mathscr{X} \times \mathscr{B}(\mathscr{X}) \to [0, 1] :$$

i. $\forall A \in \mathscr{B}(\mathscr{X})$

   $P(\cdot, A)$ is a measurable function on $\mathscr{X}$ (the sample space).

ii. $\forall x \in \mathscr{X}$

   $P(x, \cdot)$ is a probability measure on $\mathscr{B}(\mathscr{X})$.

Hence, we express $P$ as:

$$P(x, A) = Pr(X_{n+1} \in A \mid X_n = x). \tag{1}$$

*Remark.* Given our current state $x$, $P(x, \cdot)$ gives us the distribution of the next draw of our algorithm! Hence, we want to construct algorithms that give us kernels such that our target distribution $\pi$ is the stationary distribution.

Lastly, a Markov kernel $P$ is said to preserve a measure $\pi$ if:

$$\int \int \pi(dx) P(x, dy) = \int \pi(dx). \tag{2}$$

*Remark.* It is essential to understand what this means intuitively. This property ensures that given a starting value from our target distribution, if we sample according to the mechanism comprising our kernel, we will keep getting values from $\pi$!

*Note.* If the measure-theoretic notation, $\int \pi(dx)$, is new to you, you can think of this as more or less equivalent to:

$$\int \pi(x) dx$$

for all continuous distributions $\pi$ where $\pi(x)$ is the density function of the target distribution. Similarly, $P(x, dy)$ can be understood as the density function associated with our kernel at the state $x$ for the next state $y$, i.e:

$$P(x, dy) = k(x, y) dy,$$

where, $k(x, y)$ is the density function associated with our kernel.

## 2.2 Reversibility

Here, I discuss an interesting property allowing us to construct Markov transition kernels (MTKs) for our target distributions. An MTK is said to be reversible if:

$$P(x, dy)\pi(dx) = P(y, dx)\pi(dy). \tag{3}$$

This equation is often called the detailed balance, and this property is also called $\pi$-symmetry. This property is interesting because of the following theorem: if we have a $\pi-$reversible kernel, we have a $\pi-$invariant kernel.

**Theorem 1.** *$\pi$-symmetry implies $\pi$ is stationary for $P$.*

*Proof.* Define $\pi P(A) = \int_{\mathscr{X}} \pi(dx)P(x, A)$.

Then, we need to show $\pi P(A) = \pi(A) \ \ \forall\ A \in \mathscr{B}(\mathscr{X})$.

For some arbitrary $A$, consider:

$$\begin{aligned}
\pi P(A) &= \int_{\mathscr{X}} \pi(dx)P(x, A) \\
&= \int_{\mathscr{X}} \int_A \pi(dx)P(x, dy) \\
&= \int_A \pi(dy) \int_{\mathscr{X}} P(y, dx) && \text{[From reversibility]} \\
&= \int_A \pi(dy) \\
&= \pi(A)
\end{aligned}$$

Since we proved the result for any arbitrary set $A \in \mathscr{B}(\mathscr{X})$, we have successfully proved the result. □

## 2.3 A Simple Example

Till now, I have discussed how $\pi$-symmetric proposal mechanisms are desirable and how they would help us generate random samples from our target. In this part, I discuss an example that uses this principle to create a simple sampling algorithm, which is very popular in the literature. The **Metropolis-Hastings Algorithm** (MH) uses a known proposal kernel $Q(x, \cdot)$ with a known density function $q(x, y)$ to generate draws using an accept-reject framework. The only property we require from this proposal is to be able to generate proposals from its distribution.

*Note.* I intentionally do not state that our proposal should have a sample space that supersets our target. This is because proposals that do not cover the entire sample space of the target can still be invariant! However, we should always choose a proposal such that we have a $> 0$ probability of reaching any point in the target space given any starting point, i.e $\forall x, y \in \mathscr{X}$:

$$k^n(x, y) > 0.$$

A kernel with this property is said to be $\pi-$irreducible.

*Example.* Consider $Q(x, \cdot) \sim N(x, 1)$. Then,

$$q(x, y) = \frac{1}{\sqrt{2\pi}} \exp\left\{-\frac{1}{2}(y - x)^2\right\}.$$

With that context established, the MH algorithm chooses a value according to the known proposal mechanism. Then, we accept or reject the proposed new state based on an acceptance probability $\alpha(x, y)$ (Hastings' ratio) defined as:

$$a(x, y) = \frac{\pi(y)q(y, x)}{\pi(x)q(x, y)}, \tag{4}$$

where, $x$ is our current state and $y$ is the proposed value. With this, we change the current state to $y$ or "reject" the value and set the current state back to $x$. I summarise the above discussion in the Algorithm 1.

**Algorithm 1** The Metropolis-Hastings Algorithm

**Require:** Proposal Mechanism $Q(x, \cdot)$ and a starting value $X_0$
**Ensure:** Chain of random draws $X_1, \ldots, X_n$
1: **for** $i = 1$ **to** $n$ **do**
2: Draw $y \sim Q(x_{i-1}, \cdot)$
3: Calculate $\alpha(x_{i-1}, y)$ as per Eq.4.
4: w.p $\alpha(x_{i-1}, y)$ set $X_i \leftarrow y$ else, $X_i \leftarrow X_{i-1}$
5: **end for**

The next question to answer is, "Does this even work?" To prove this, I use the results I have discussed above and prove the following two theorems. The first theorem explicitly establishes the Markov transition kernel for the MH algorithm. The second theorem shows that the MH kernel is $\pi-$symmetric.

**Theorem 2.** *The MH algorithm defines the following Markov transition kernel.*

$$P(x, A) = \int_A Q(x, dy)a(x, y) + \delta_x(A) \int [1 - a(x, u)]Q(x, du)$$

*Proof.*

$$\begin{aligned} P(x, A) &= Pr(X_{n+1} \in A \mid X_n = x) \\ &= Pr(X_{n+1} \in A, U \leq a(X_n, Y) \mid X_n = x) + Pr(X_{n+1} \in A, U > a(X_n, Y) \mid X_n = x) \end{aligned}$$

And, following this, it's relatively easy for us to prove:

I.
$$Pr(X_{n+1} \in A, U \leq a(X_n, Y) \mid X_n = x) = \int_A Q(x, dy)a(x, y)$$

II.
$$Pr(X_{n+1} \in A, U > a(X_n, Y) \mid X_n = x) = \delta_x(A) \int [1 - a(x, u)]Q(x, du)$$

And the rest follows. □

**Theorem 3.** *The MH kernel is $\pi$-symmetric and hence $\pi$-invariant.*

*Proof.* We need to show:

$$\pi(dx)P(x, dy) = \pi(dy)P(y, dx).$$

Note that this is true if $x = y$. else, $x \neq y$, then,

$$\begin{aligned}\pi(dx)P(x, dy) &= \pi(x)q(x, y)\min(1, a(x, y)) \\ &= \min(\pi(x)q(x, y), \pi(y)q(y, x)) \\ &= \pi(y)q(y, x)\ \ \min(1, a(y, x)) \\ &= \pi(dy)P(y, dx)\end{aligned}$$

□

*Note.* This theorem helps us ensure that, if $(X_0 \sim \pi) \Rightarrow (X_t \sim \pi\ , \forall t \geq 0)$

Now that we know that the MH algorithm indeed samples from the correct proposal, I provide a few visuals that help us understand the MH algorithm intuitively. Here, I also discuss other subtle ideas about sampling efficiency that one should consider while implementing these algorithms through visuals. I do not provide extensive theory on these ideas. In Fig. 1, I provide four visualisations of the samples generated using the MH algorithm. The first plot (top-left) showcases the target distribution $(0.6N(0, 1) + 0.4N(3, 1))$ and visualises the generated samples. The second plot (top-right) plots the value of the samples against their time-index, showing how the sampler "moves" in the sample space. This plot should not show discernible patterns and indicate that the sampler can uniformly traverse the entire sample space. The third plot (bottom left) shows the estimated autocorrelations for various lags. Ideally, we want this value to quickly go to zero, because we want to attain uncorrelated (independent) samples for our estimation (remember CLT!). Lastly, the bottom-right plot shows the estimated density of the samples and compares it to the target density. This plot visually assures us that we are sampling from the correct distribution.

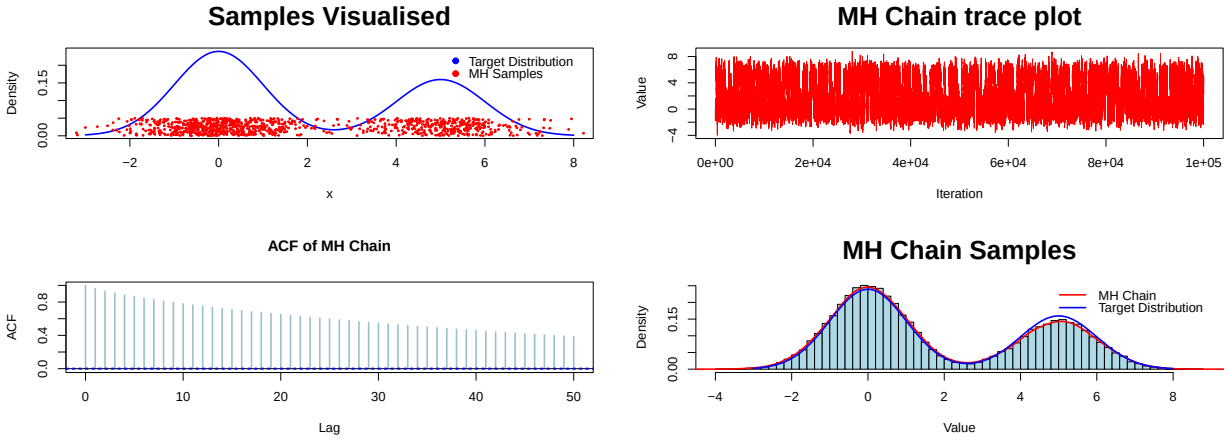


Figure 1: Visualising the samples generated using the MH algorithm.

# 3 Physics!

For this section, I take a slight detour from statistics and talk about a concept in physics called Hamilton's equations. By the end of this section, I will connect this seemingly unrelated concept back to MCMC sampling.

## 3.1 Hamiltonian Dynamics

Assume an imaginary particle $a$ moving on a frictionless surface. This particle has both potential and kinetic energy. Now, assume that the state of this particle is represented using two values: its position in the space $x$ and its momentum $p$. Then, we can represent the potential and kinetic energy of the particle by:

$$U(x) = -\log(\pi(x))$$
$$K(p) = \frac{p^2}{2m}.$$

and define the Hamiltonian as:

$$H(p, x) = K(p) + U(x).$$

Then, Hamilton's equations are:

$$\frac{dx}{dt} = \frac{\partial H(p,x)}{\partial p}, \quad \& \quad \frac{dp}{dt} = -\frac{\partial H(p,x)}{\partial x}. \tag{5}$$

Using these equations, we can determine our particle's exact position and momentum in our defined system, given its initial position and momentum.

*Note.* Here, I write the potential energy function in terms of the target. I will use this fact later when I connect this concept to MCMC.

*Example.* Consider $\pi \sim N(0, 1)$ and the mass of the body to be 1kg, which gives:

$$U(x) = \frac{x^2}{2} \qquad\qquad K(p) = \frac{p^2}{2}$$

and consequently,

$$H(p, x) = \frac{x^2}{2} + \frac{p^2}{2}$$

Finally, solving for the Hamiltonian Equations:

$$\frac{dx}{dt} = \frac{\partial H(p,x)}{\partial p} \qquad\qquad \frac{dp}{dt} = -\frac{\partial H(p,x)}{\partial x}$$
$$= p \qquad\qquad = -x$$

We get,

$$x(t) = r\cos(t + a) \qquad\qquad p(t) = -r\sin(t + a)$$

Here, $r$ and $a$ are the variables that depend on our starting values of the momentum and position. Say, we set the values of $r = 1$ and $a = 0$, we notice that the particle moves along a contour described by Fig. 2. I also provide the plot for potential energy against the particle's position. This also gives us a good visual intuition about the particle's state. Given a starting point, we can think of the particle sliding along the walls of a parabolic bowl and get a feel for how our equations tell us the time evolution of its position and momentum.

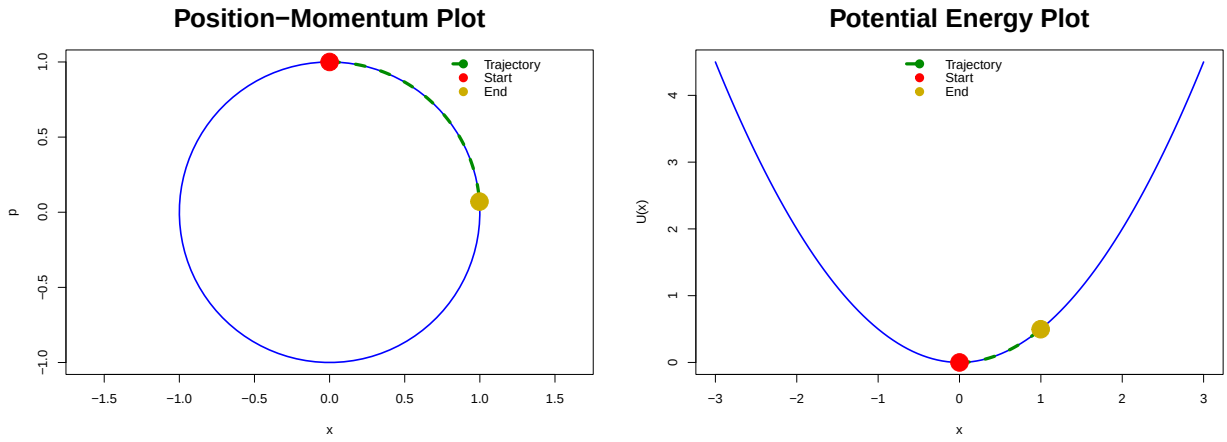


Figure 2: Visualising the trajectory of our particle in the Gaussian potential field.

## 3.2 Hamiltonian Monte Carlo

A natural question to ask now is, "Why do we care about this dammed particle?" The answer lies in how its position evolves with time. Given any starting position, we notice that the particle eventually moves towards the "bottom of the bowl". This means that the particle moves towards a state of minimum potential. We want this property to generate efficient proposals for our target distribution. We notice that the potential energy function, by our framing, is the negative log of our target density function. This means that the point of minimum potential is a mode of the target distribution. Connecting the dots, if we use Hamiltonian dynamics to generate proposals, we will move towards the modes of our target distribution!

The only question we need to answer here is, how do we account for the randomness in the model? The answer is that we sample a new momentum each time. Given a new value for momentum at our current position, we move to a new state in our system. If we compute the position of our particle after a time delay, say $s$, we arrive at a new position that follows the same potential energy system and is a random draw from our target. I provide a visual for this entire step in Fig. 3.

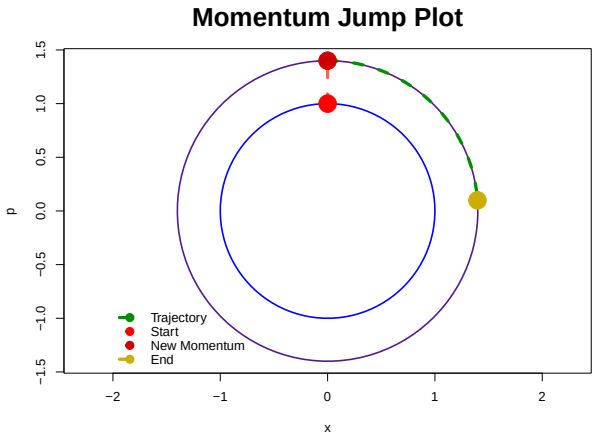


Figure 3: Visualising the impact of change in momentum on the state.

After generating this new value, we have a proposal that we are very confident represents the target well. We use the standard accept-reject mechanism discussed in the MH algorithm to complete our algorithm. I summarise the above idea and compile it into a sampling mechanism in Algorithm 2.

**Algorithm 2** Hamiltonian Monte Carlo

**Require:** Increment time $s$ and a starting value $X_0$
**Ensure:** Chain of random draws $X_1, \ldots, X_n$
1: **for** $i = 1$ **to** $n$ **do**
2: Propose $p \sim N(0, 1)$
3: Compute the state of the particle after a time $s$, $(x', p')$ by solving Eq. 5
4: Calculate the acceptance probability:

$$\alpha(x, x') = \min\left\{1, \frac{\pi(x')N(p' \mid 0, 1)}{\pi(x)N(p \mid 0, 1)}\right\}.$$

5: w.p $\alpha(x, x')$ set $X_{i+1} \leftarrow x'$ else, $X_i \leftarrow X_{i-1}$
6: **end for**

*Note.* The proposal mechanism involves independently drawing a random value for the momentum and then deterministically calculating the respective position after a time delay. Hence, the proposal is distributed by momentum.

*Remark.* Consider the acceptance probability:

$$\alpha(x, x') = \min\left\{1, \frac{\pi(x')N(p' \mid 0, 1)}{\pi(x)N(p \mid 0, 1)}\right\}.$$

Notice that the inner ratio can be expressed as:

$$\begin{aligned}
\frac{\pi(x')N(p' \mid 0, 1)}{\pi(x)N(p \mid 0, 1)} &= \frac{\exp\{-H(x', p')\}}{\exp\{-H(x, p)\}} \\
&= \exp\{H(x, p) - H(x', p')\} \\
&= \exp\{0\} && \text{[Conservation of energy!]} \\
&= 1
\end{aligned}$$

Hence, $\alpha(x, x') = 1$. This means that we always accept the proposed value! This is why Hamiltonian Monte Carlo is a powerful algorithm that generates efficient samples without ever directly sampling from the target. Hence, we can simplify the mechanism as Algorithm 3.

**Algorithm 3** (Efficient) Hamiltonian Monte Carlo

**Require:** Increment time $s$ and a starting value $X_0$
**Ensure:** Chain of random draws $X_1, \ldots, X_n$
1: **for** $i = 1$ **to** $n$ **do**
2: Propose $p \sim N(0, 1)$
3: Compute the state of the particle after a time $s$, $(x', p')$ by solving Eq. 5
4: Set $X_{i+1} \leftarrow x'$
5: **end for**

I implement this algorithm for the standard normal target for different values of $s$. I visualise the estimated density of the samples and their respective ACF plots in Fig. 4. Note that for all values of $s$

we get a decent approximation of the target; however, for $s = 5$, we observe that the ACF plot shows no correlation between the samples after the second lag. This differs greatly from Fig. 1. This figure shows how properly tuned HMC algorithms can give us virtually independent samples that help us get excellent results during inference. However, as expected, this isn't easy to achieve in practice.

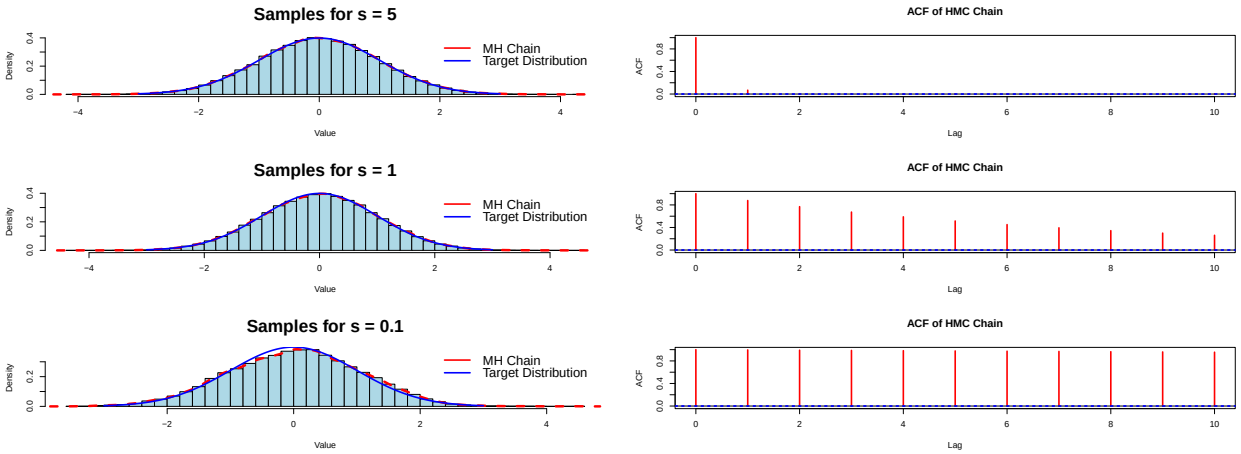


Figure 4: Visualising the samples generated for different values of $s$ using the HMC algorithm

### 3.3 There's always a catch

So far, this algorithm seems like the heavens smiling down upon us. However, as they say, if something feels too good to be true, it probably is. As it turns out, the solutions to Hamilton's Equations (Eq. 5) are mostly intractable. To solve this problem, we fix our $p$ to come from a standard normal distribution, and we can approximate the time update for $p$ and $q$ with the following Euler approximations:

$$p(t+\epsilon) = p(t) + \epsilon \frac{dp}{dt} = p(t) - \epsilon \frac{\partial H(p,x)}{\partial x}.$$

This means:

$$p(t+\epsilon) = p(t) - \epsilon \nabla \log \pi(x).$$

Similarly:

$$x(t+\epsilon) = x(t) + \epsilon \frac{p}{m}.$$

However, these approximations show great variance and are unstable since the update of time $(t+\epsilon)$ depend on both $q$ and $p$ simultaneously. To solve this, we scatter the update sequence so that information on $p$ is $\frac{\epsilon}{2}$ time units ahead of information on $q$. The "jumps" look like:

$$\begin{aligned} p_{t+\epsilon/2} &= p_t + \epsilon/2 \nabla \log \pi(x_t) \\ x_{t+\epsilon} &= x_t + \epsilon p_{t+\epsilon/2} \\ p_{t+\epsilon} &= p_{t+\epsilon/2} - \epsilon/2 \nabla \log \pi(x_{t+\epsilon}) \end{aligned} \tag{6}$$

*Note.* This simulates the time jump of length $\epsilon$.

We can repeat this step $L$ times for a larger time update. I use the notation $\mathbf{\Phi}_{\mathbf{L},\epsilon}(x,p)$ to denote the

mapping that transforms the current state $(x, p)$ by putting in through Eq. 6 $L$ times. I provide visuals for the leapfrog approximation for the contour discussed in the previous example in Fig. 5. Here, I run the leapfrog step for $L = 10$ times for different values of $\epsilon$ to showcase the effect of changing these parameters. A large value for $\epsilon$ allows the sampler to propose far-off values, which is suitable for proposing less correlated samples. However, it also strays the approximated value further away from the true contour, making the acceptance probability low. Do note that every step of the leapfrog involves computing the gradient of the proposal, which is time-consuming as the dimensions of the target grow. Hence, we would also want to keep the value of $L$ small. Hence, tuning these two parameters, namely $L$ and $\epsilon$, is the main challenge in tuning HMC algorithms. With that, I present the popular HMC paradigm in the Algorithm 4.

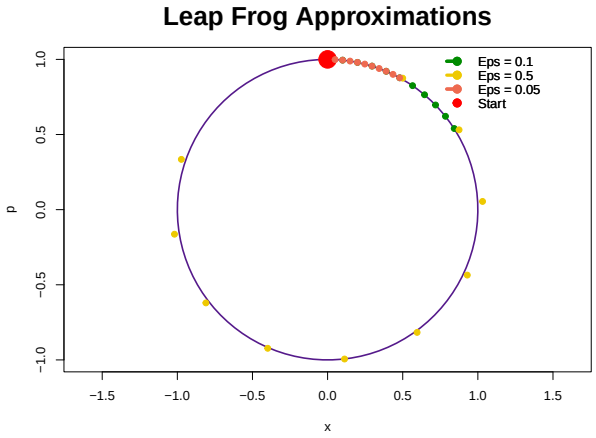


Figure 5: Visualising the leapfrog approximations.

**Algorithm 4** HMC using Leapfrog

1: Draw $\mathbf{p} \sim \mathcal{N}_d(0, I_d)$
2: Calculate $(\mathbf{x}', \mathbf{p}') = \mathbf{\Phi}_{\mathbf{L}\epsilon}(\mathbf{x}, \mathbf{p})$
3: Calculate the acceptance ratio:

$$\alpha(\mathbf{x}, \mathbf{x}') = \min\left\{1, \exp\left\{-H(\mathbf{x}', \mathbf{p}')) + H(\mathbf{x}, \mathbf{p}))\right\}\right\}$$

4: **if** runif$(0, 1) < \alpha(\mathbf{x}, \mathbf{x}')$ **then**
5: $\quad \mathbf{x}_{k+1} = \mathbf{x}'$
6: **else**
7: $\quad \mathbf{x}_{k+1} = \mathbf{x}_k$
8: **end if**

*Note.* Here, I also generalise the algorithm for a $d$-dimensional target.

# 4 Theory, Experiments and Discussion

In this section, I discuss the underlying theory and technical details of all the claims I made in the previous section. Here, I prove why HMC is indeed $\pi$-invariant, and I also discuss a broader concept of deterministic proposals. Later in this section, I will also lay out the practical limitations of the basic HMC paradigm and

cite popular improvements (also one that I have worked on!) of the same that solve various problems with the algorithm.

## 4.1 Theory

First, let us take a good look at our proposal mechanism. In a note (3.2), I claim that the distribution of the proposal is intuitively only dependent on that of the momentum draw. I dissect that claim further, consider:

$$Q(x, A) = Pr(x' \in A \mid x),$$

Then, we can look at the conditional distribution of the proposal, conditional on the momentum drawn.

$$\begin{aligned} Q(x, A) &= Pr(x' \in A \mid x) \\ &= Pr(\mathbf{\Phi_s}(x, p) \in A|x) \\ &= N(\mathbf{\Phi_s^{-1}}(A)|0, 1). \end{aligned}$$

where $\mathbf{\Phi_s}(x, p)$ is the time increment in the state by time $s$, and

$$\mathbf{\Phi_s^{-1}}(A) := \{x' : x' = \mathbf{\Phi_s}(x, p) \mid x \in A\}.$$

Note that this event is exactly the same as us sampling the "appropriate" momentum value such that $x'$ lies in $A$. Hence,

$$N(\mathbf{\Phi_s^{-1}}(A)|0, 1) = N(p|0, 1).$$

With this, we can have a greater understanding and confidence in why the HMC proposal mechanism fits into the MH algorithm, giving us a $\pi$-invariant algorithm.

*Remark.* This is still a pretty hand-wavy argument on how and why HMC works. However, this suffices to give an intuitive understanding of the more technical arguments used to prove the invariance of the algorithm. I do not dive deep into the complete proof. However, you can find the same in Neal et al. (2011).

## 4.2 Experiments

### 4.2.1 Multi-modal madness

Let us revisit the target distribution we started with in this article, $0.6N(0, 1) + 0.4N(3, 1)$. In Fig. 6, I illustrate the samples generated using the HMC algorithm. Note that the estimated ACF values decrease quickly to zero, indicating that the sampler gives better results. However, let's test the limits of our sampler! In Fig. 7, I keep increasing the distance between the two modes to see what happens to the sampler. As the distance between the modes keeps increasing, the estimated density strays further away from the true density. This is a common problem with most gradient-driven MCMC algorithms. Because HMC rigs the proposals towards a mode, it disincentivises the algorithm from efficiently exploring the entire target domain to look for other potential modes.

Many different approaches to this problem are discussed in popular literature, like Magnetic HMC (Tripuraneni et al., 2017), Repelling-Attracting HMC (Vishwanath and Tak, 2024) and Tempered HMC (something that I am working on!). All of these generalise the Hamiltonian dynamics that solve for the sampler being able to "jump energy-barriers" better. I provide a visual for these energy barriers in Fig. 8. Referring back

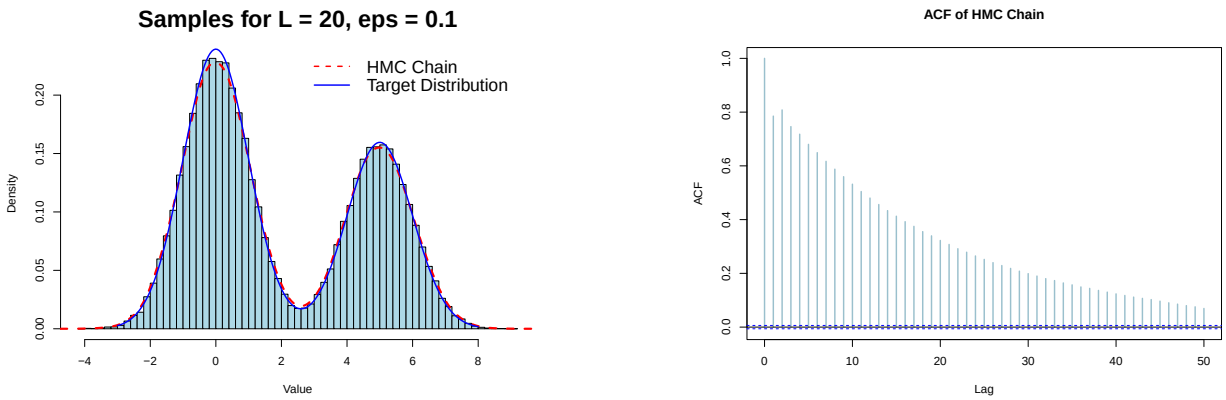


Figure 6: Estimated sample density for the bi-modal target.

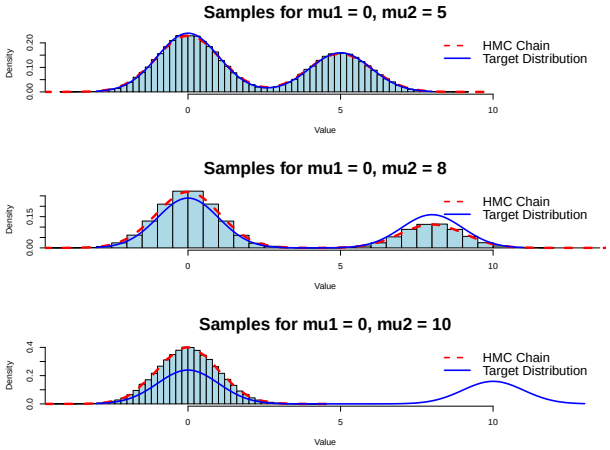


Figure 7: HMC not being able to explore the target domain well.

to our ball in a bowl analogy, it's easy to notice that here, our particle would have to "climb" up a very high energy barrier to reach the new mode, and this becomes increasingly difficult as the modes go further apart.

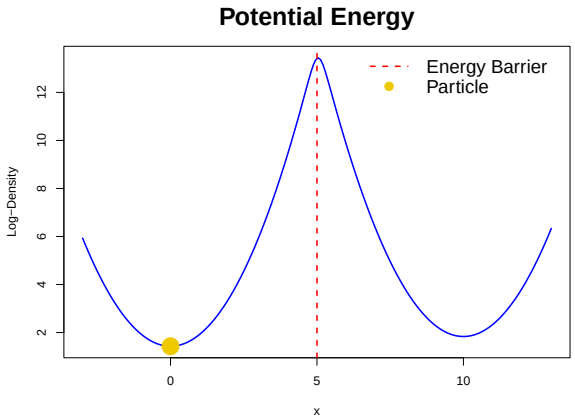


Figure 8: Visualising the high potential barriers in sampling from multimodal distributions.

### 4.2.2 Better than IID

I refer to the standard normal example, pick a particular value of $s$ ($s = 10$), and analyse the ACF plot. In Fig. 9, note that the estimated ACF values are negative! Sparing the technical details, let us think about this intuitively. Ideally, for CLT to hold on a sample, we require independent samples. The main problem with MCMC is that we get correlated samples and hence, we can't quite use CLT directly. Hence, we discussed that we want samples that have ACF plots that drop quickly to at least approximately ensure CLT. To quantify this idea, we have concept in MCMC called called effective sample size (ESS), this roughly gives us the number of IID samples our MCMC samples are equivalent to in terms of CLT holding. The formula for ESS is:

$$\text{ESS} = \frac{N}{1 + 2\sum_{k=1}^{\infty} \rho_k}. \tag{7}$$

Note, in our case, $\sum_{k=1}^{\infty} \rho_k < 0$ and consequently $ESS > N$. This means that HMC gives better samples than actual IID samples! This means that HMC gives us samples that are better representative of our target distribution in fewer number of samples. I mention again, this is not easy to achieve for real life targets.

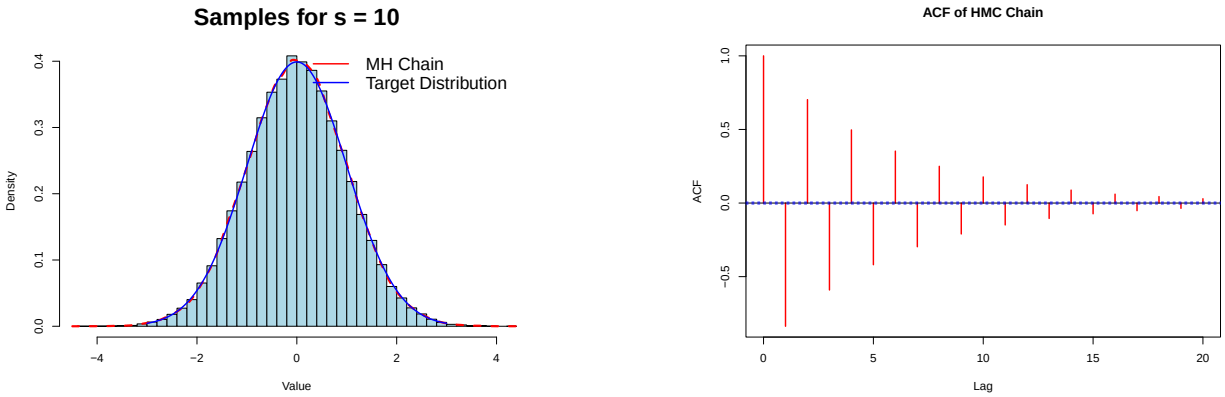


Figure 9: Illustrating the negative ACF plots for HMC

# 5 Conclusion

HMC is the first MCMC algorithm that I learnt and have researched on. I am excited to write this note and hope that readers find this useful and this note becomes their segue for MCMC like Vats (2023) was for me.

# References


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

Takeda, K. and Sakajo, T. (2023). Geometric ergodicity for hamiltonian monte carlo on compact manifolds. *SIAM Journal on Numerical Analysis*, 61(6):2994–3013.

Tripuraneni, N., Rowland, M., Ghahramani, Z., and Turner, R. (2017). Magnetic hamiltonian monte carlo. In *International Conference on Machine Learning*, pages 3453–3461. PMLR.

Vats, D. (2023). Hamiltonian monte carlo for (physics) dummies.

Vishwanath, S. and Tak, H. (2024). Repelling-attracting hamiltonian monte carlo.