Math 153 Final Project: Hamiltonian Monte Carlo

Matthew Patterson

May 14, 2019

Introduction

In Bayesian statistics, the kernel of a posterior distribution is usually unrecognizable, and does not belong to any class of common distributions. Moreover, it is often impossible to integrate analytically due to its multiplicative complexity, preventing exact calculations of the normalizing constant, as well as information like credible intervals and expected values. Thus, the most common way to gain information from the kernel of a posterior is through simulation.

To this end, Monte Carlo sampling algorithms are commonly used to simulate draws from a *target distribution*, which could for instance be a posterior. Two of the most well-known Monte Carlo sampling algorithms are the Metropolis-Hastings Algorithm, which we will sometimes call M-H, and the Gibbs Sampler. While the draws generated by these algorithms converge asymptotically to the desired probability distribution, they have practical limitations. In particular, in the case of a local random-walk Metropolis-Hastings Algorithm, it is time-consuming to explore the support of the target distribution, since it is often the case that the steps taken around the support are either too large or too small, causing the chain to mix poorly. This problem is only exacerbated in higher dimensions or with multiple modes, as traversing regions of large size or low probability density is difficult.

The Hamiltonian Monte Carlo Algorithm, which we will sometimes call HMC, provides a solution to this problem for a large class of distributions. By treating a target distribution as a Hamiltonian physical system, HMC allows us to traverse its support more efficiently than we would using a random-walk Metropolis-Hastings algorithm. In the following paper, we will explain the Hamiltonian intuition behind HMC. We will then specify the steps of HMC and some practical modifications. Finally, as an example, we will use HMC to generate draws from a specific target distribution and compare its performance to that of a local random walk Metropolis-Hastings algorithm.

Intuition Behind Hamiltonian Monte Carlo

The basics of Hamiltonian mechanics are detailed in (Neal 2011, 114–19), which we paraphrase as follows. A Hamiltonian physical system is parameterized by the following two components: a k-dimensional space of *position* vectors $\mathbf{x} = (x_1, \ldots, x_k)$, and a k-dimensional space of *momentum* vectors $\mathbf{y} = (y_1, \ldots, y_k)$. The space upon which the Hamiltonian system acts is the Cartesian product of the position and momentum spaces. A function called the *Hamiltonian*, denoted $H(\mathbf{x}, \mathbf{y})$, takes an argument from the Cartesian product of position and momentum and maps it to some scalar value, which can be thought of as *energy*. The Hamiltonian determines the trajectories of particles in the position and momentum space over time t by obeying the following partial differential equations, called *Hamilton's equations*:

$$\frac{dx_i}{dt} = \frac{\partial H}{\partial y_i},$$
$$\frac{dy_i}{dt} = -\frac{\partial H}{\partial x_i}$$

for $1 \leq i \leq k$. Hamilton's equations ensures that the Hamiltonian remains constant, allowing us to conserve energy, and that the trajectories of particles are reversible.

As demonstrated in (Neal 2011, 122–23), this physical framework can be translated to our probabilistic system. Suppose we have a random variable $\mathbf{X}$ of dimension k with the target distribution $f(\mathbf{x})$ in which we are interested. We will interpret $\mathbf{X}$ as a vector of position, and define a k-dimensional, auxiliary random variable $\mathbf{Y}$ with known distribution $g(\mathbf{y})$ which we will interpret as a vector of momentum. In this construction, we will require that $\mathbf{Y}$ is independent of $\mathbf{X}$. Then we can construct a Hamiltonian $H(\mathbf{x}, \mathbf{y})$ on the pair $(\mathbf{X}, \mathbf{Y})$, which we shall define as

$$H(\mathbf{x}, \mathbf{y}) = U(\mathbf{x}) + K(\mathbf{y})$$

where the function U is called the *potential energy* and the function K is called the *kinetic energy*. Thus Hamilton's equations become

$$\begin{aligned} \frac{dx_i}{dt} &= \frac{\partial K}{\partial y_i} \\ \frac{dy_i}{dt} &= -\frac{\partial U}{\partial x} \end{aligned}$$

We can then define a joint distribution on $\mathbf{X}$ and $\mathbf{Y}$ in terms of the Hamiltonian, given by

$$\begin{aligned} p(\mathbf{x}, \mathbf{y}) &= e^{-H(\mathbf{x}, \mathbf{y})} \\ &= e^{-U(\mathbf{x})} e^{-K(\mathbf{y})} \end{aligned}$$

Since $\mathbf{X}$ and $\mathbf{Y}$ are independent with pdf's $f(\mathbf{x})$ and $g(\mathbf{y})$, respectively, we can derive formulas for the potential and kinetic energy from the joint:

$$\begin{aligned} U(\mathbf{x}) &= -\log f(\mathbf{x}) \\ K(\mathbf{y}) &= -\log g(\mathbf{y}) \end{aligned}$$

Therefore, we now have the framework to simulate Hamiltonian movement over time around the joint probability space defined by the joint distribution $p(\mathbf{x}, \mathbf{y})$ by solving Hamilton's equations.

Basically, by using Hamilton's equations to solve for randomized trajectories in $\mathbf{X}$ and $\mathbf{Y}$, we are able to take advantage of energy conservation properties to achieve a high degree of mixture once we project back down to the position variable $\mathbf{X}$. In particular, rather than simply gravitating towards regions of high probability as in a naive Metropolis-Hastings approach using local random walks, the HMC algorithm propels our chain across the space by giving it momentum, allowing it to traverse regions of large size or low probability density with ease. HMC will use this Hamiltonian framework to construct a reversible Markov chain whose stationary distribution is the target distribution.

The Hamiltonian Monte Carlo Algorithm

We now explicitly lay out the steps of the HMC algorithm as specified in (Neal 2011, 124–25). We will refer to the random variable $\mathbf{X}$ with the target distribution as the *position*, and we shall refer to the auxiliary random variable $\mathbf{Y}$ as the *momentum*. Let $f(\mathbf{x})$ be the target distribution and let $g(\mathbf{y})$ be the auxiliary distribution on the momentum. The potential energy $U(\mathbf{x})$ and the kinetic energy $K(\mathbf{y})$ are defined as in the previous section.

Suppose we want n draws from the target distribution, and we set our time step size to t. Let ϕ_t denote the function that takes a point $(\mathbf{x}, \mathbf{y})$ and sends it to its position at time t in according to Hamilton's equations. Then the Hamiltonian Monte Carlo Algorithm, which will return a sample of size n from the target distribution, is given as follows:

HMC Algorithm

1. Initialize an instance $\mathbf{x}_0$ of the position variable $\mathbf{X}$.
2. Repeat the following steps for $0 \leq i \leq n-1$.
   a. Sample $\mathbf{y}_i \sim g(\mathbf{y})$.
   b. Evaluate the time evolution $(\mathbf{x}_i^{ev}, \mathbf{y}_i^{ev}) = \phi_t(\mathbf{x}_i, \mathbf{y}_i)$.
   c. Fix the candidate $(\mathbf{x}_i^*, \mathbf{y}_i^*) = (\mathbf{x}_i^{ev}, -\mathbf{y}_i^{ev})$
   d. Calculate the acceptance ratio

$$r = \min\{1, \exp(U(\mathbf{x}_i) - U(\mathbf{x}_i^*) + K(\mathbf{y}_i) - K(\mathbf{y}_i^*))\}$$

   - Set $\mathbf{x}_{i+1} = \mathbf{x}_i^*$ with probability r, and set $\mathbf{x}_{i+1} = \mathbf{x}_i$ with probability $1-r$.
3. The sequence $\{\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{n-1}\}$ is a sample from $\mathbf{X}$.

The time evolution $\phi_t(\mathbf{x}, \mathbf{y})$ will usually need to be evaluated numerically using algorithms like the *leapfrog method* described in (Neal 2011, 121) Let $(\mathbf{x}, \mathbf{y})$ be some arbitrary pair of position and momentum, and let t be our time step. Let $\epsilon > 0$ be some discretization parameter. Then the leapfrog method approximates $\phi_i(\mathbf{x}, \mathbf{y})$ as follows: ## Leapfrog Method 1. Let $\mathbf{x}'_0 = \mathbf{x}$ and $\mathbf{y}'_0 = \mathbf{y}$. 2. Let $N = \lfloor \frac{t}{\epsilon} \rfloor$. 3. Repeat the following steps for $0 \leq i \leq N$. a. Let $\mathbf{y}'_{i+\frac{1}{2}} = \mathbf{y}'_i - \frac{\epsilon}{2}\frac{\partial U}{\partial \mathbf{x}}(\mathbf{x}'_i)$. b. Let $\mathbf{x}'_{i+1} = \mathbf{x}'_i + \epsilon \mathbf{y}'_{i+\frac{1}{2}}$. c. Let $\mathbf{y}'_i = \mathbf{y}'_{i+\frac{1}{2}} - \frac{\epsilon}{2}\frac{\partial U}{\partial \mathbf{x}}(\mathbf{x}'_{i+1})$. 4. Then $\phi_i(\mathbf{x}, \mathbf{y}) = (\mathbf{x}'_N, \mathbf{y}'_N)$.

To calculate $\phi_t(\mathbf{x}, \mathbf{y})$, we will be working with the partial derivative $\frac{\partial U}{\partial \mathbf{x}}$. Since $U(\mathbf{x}) = -\log f(\mathbf{x})$, it follows that the normalizing constant of $f(\mathbf{x})$ will become an additive constant. and disappear once the aforementioned partial derivative is calculated. Therefore, only the kernel of the target distribution $f(\mathbf{x})$ is necessary to carry out the HMC Algorithm, making it well-suited to problems in Bayesian statistics.

An Implementation of the HMC Algorithm

We now provide an example implementation of the Hamiltonian Monte Carlo Algorithm and compare it to a local random walk Metropolis-Hastings approach. Consider the arbitrary normal pdf's

$$\begin{aligned} f_1(x \mid \mu_1, \sigma_1) &= \frac{1}{\sqrt{2\pi\sigma_1^2}} \exp\left(-\frac{(x-\mu_1)^2}{2\sigma_1^2}\right) \\ f_2(x \mid \mu_2, \sigma_2) &= \frac{1}{\sqrt{2\pi\sigma_2^2}} \exp\left(-\frac{(x-\mu_2)^2}{2\sigma_2^2}\right) \end{aligned}$$

Note that the normal random variables parameterized by $f_1(x)$ and $f_2(x)$ have mean μ_1 and μ_2, respectively, and standard deviation σ_1 and σ_2, respectively. Suppose we want to generate draws from a random variable X whose distribution is a finite normal mixture model with the following pdf:

$$f(x) = p_1 f_1(x \mid \mu_1, \sigma_1) + p_2 f_2(x \mid \mu_2, \sigma_2)$$

where $p_1, p_2 > 0$ and $p_1 + p_2 = 1$. The parameters of this mixture model are μ_1, μ_2, σ_1, σ_2, p_1, and p_2. Note that, since it has two normal parts, it is bimodal. First, we will consider a Hamiltonian Monte Carlo approach to the problem. Since the random variable X with this pdf is one-dimensional, we will assign it a one-dimensional momentum random variable Y. For the purposes of this implementation, we will let

$$Y \sim \mathcal{N}(\mu, \sigma).$$

be our candidate distribution for some μ and σ to be chosen later at the time of implementation. Since Y is independent of X, we can use the leapfrog method. The potential energy function is given by

$$\begin{aligned}
U(x) &= -\log f(x) \\
&= -\log\left(\frac{1}{\sqrt{2\pi\sigma_1^2}}\exp\left(-\frac{(x-\mu_1)^2}{2\sigma_1^2}\right) + \frac{1}{\sqrt{2\pi\sigma_2^2}}\exp\left(-\frac{(x-\mu_2)^2}{2\sigma_2^2}\right)\right) \\
&= -\log\left(\frac{1}{\sqrt{2\pi}}\right) - \log\left(\frac{p_1}{\sigma_1}\exp\left(-\frac{(x-\mu_1)^2}{2\sigma_1^2}\right) + \frac{p_2}{\sigma_2}\exp\left(-\frac{(x-\mu_2)^2}{2\sigma_2^2}\right)\right)
\end{aligned}$$

Using the following auxiliary functions for the sake of readability,

$$\begin{aligned}
\mathcal{T}_1(x) &= \left(\frac{p_1}{\sigma_1}\exp\left(-\frac{(x-\mu_1)^2}{2\sigma_1^2}\right) + \frac{p_2}{\sigma_2}\exp\left(-\frac{(x-\mu_2)^2}{2\sigma_2^2}\right)\right)^{-1} \\
\mathcal{T}_2(x) &= \frac{2p_1(x-\mu_1)}{2\sigma_1^3}\exp\left(\frac{-(x-\mu_1)^2}{2\sigma_1^2}\right) + \frac{2p_2(x-\mu_2)}{2\sigma_2^3}\exp\left(\frac{-(x-\mu_2)^2}{2\sigma_2^2}\right)
\end{aligned}$$

we calculate that the partial derivative of the potential energy with respect to position is

$$\frac{\partial U}{\partial x} = \mathcal{T}_1(x)\mathcal{T}_2(x)$$

Now that we have calculated the necessary partial derivatives, we can write a function in **R** that implements HMC for our normal mixture model. We suppress the source code for our auxiliary functions for the sake of readability, as understanding their return values is sufficient to see HMC in action. In particular, `partialU` returns the partial derivative of the potential energy $U(x)$ with respect to x, `targetDensity` returns the value of the pdf of the normal mixture at x, `potential` returns the value of $U(x)$ at x, and `kinetic` returns the value of $K(y)$ at y. Using the following utility functions, we can define a function that implements the HMC algorithm to draw from the normal mixture model.

```r
# partialU returns the value of the partial derivative of the potential energy
# function of the given normal mixture model evaluated at a point

# Parameters:
# x := the point where the partial derivative will be evaluated
# p1 := weight of first normal component
# mu1 := mean of the first normal component f1
# sigma1 := standard deviation of the first normal component
# p2 := weight of second normal component
# mu2 := mean of the second normal component f2
# sigma2 := standard deviation of the second normal component

partialU <- function(x,p1, mu1, sigma1, p2, mu2, sigma2) {
  # calculate the factors of the partial derivative
  T1 = 1/((p1/sigma1)*exp(-1*((x-mu1)^2/(2*sigma1^2))) +
          (p2/sigma2)*exp(-1*((x-mu2)^2/(2*sigma2^2))))
  T2 = ((2*p1*(x-mu1))/(2*sigma1^3))*exp(-1*((x-mu1)^2/(2*sigma1^2))) +
          ((2*p2*(x-mu2))/(2*sigma2^3))*exp(-1*((x-mu2)^2/(2*sigma2^2)))
  # return their product
  return(T1*T2)
}

# returns the density of the normal mixture at x
targetDensity <- function(x, p1, mu1, sigma1, p2, mu2, sigma2) {
  return(p1*(dnorm(x,mu1,sigma1)) + p2*(dnorm(x,mu2,sigma2)))
}

# returns the potential energy
potential <- function(x, p1, mu1, sigma1, p2, mu2, sigma2) {
  density = targetDensity(x,p1,mu1,sigma1,p2,mu2,sigma2)
  return(-1*log(density))
}

# returns the kinetic energy
kinetic <- function(y, mu, sigma) {
  return(-1*log(dnorm(y,mu,sigma)))
}
```

Hamiltonian Monte Carlo Function

```
# binaryMix returns a vector of draws from an arbitrary normal mixture model
# with two normal pdf components using the HMC Algorithm, using a normal
# distribution for the pdf of the momentum

# Parameters:
# n := the number of draws
# p1 := weight of first normal component
# mu1 := mean of the first normal component f1
# sigma1 := standard deviation of the first normal component
# p2 := weight of second normal component
# mu2 := mean of the second normal component f2
# sigma2 := standard deviation of the second normal component
# mean := the mean of the momentum (normal)
# sd := the standard deviation of the momentum (normal)
# x0 := the initial position of the chain
# t := size of the time step
# epsilon := discretization parameter for trajectory calculation

binaryMix <- function(n, p1, mu1, sigma1, p2, mu2, sigma2, mean,
                      sd, x0, t, epsilon) {
  draws <- c() # vector of draws
  draws[1] = x0 # initialize the draws
  # generate draws using auxiliary momentum parameters
  for(i in 1:n) {
    x = draws[i]
    # draw a momentum parameter from the candidate distribution
    y = rnorm(1,mean,sd)
    # calculate the time evolution of (x,y) using the
    # leapfrog integrator
```

```r
    xStar = x
    yStar = y
    for(j in 1:(t %/% epsilon)) {
      # calculate the partial derivative at xStar
      yStar = yStar - (epsilon/2)*partialU(xStar,p1,mu1,sigma1,p2,mu2,sigma2)
      xStar = xStar + epsilon*yStar
      yStar = yStar - (epsilon/2)*partialU(xStar,p1,mu1,sigma1,p2,mu2,sigma2)
    }
    # negate the momentum
    yStar = -1*yStar
    # calculate the acceptance ratio
    acc = exp(potential(x,p1,mu1,sigma1,p2,mu2,sigma2) -
      potential(xStar,p1,mu1,sigma1,p2,mu2,sigma2) +
      kinetic(y,mean,sd) - kinetic(yStar,mean,sd))
    r = min(1,acc)
    # run the acceptance step
    if(runif(1,0,1) < r) {
      draws[i+1] = xStar
    }
    else {
      draws[i+1] = x
    }
  }
  # return the vector of draws
  return(draws)
}
```

We will compare the performance of the HMC Algorithm to that of a naive, local random walk Metropolis-Hastings Algorithm, implemented as follows:

Local Random Walk Metropolis-Hastings Function

```r
# randomWalk returns a vector of draws from the target distribution f(x)
# generated using a naive local random walk Metropolis-Hastings Algorithm

# Parameters:
# n, p1, mu1, sigma1, p2, mu2, sigma2, x0 are all the same parameters
# as in binaryMix
# delta := width of the local random walk neighborhood

randomWalk <- function(n, p1, mu1, sigma1, p2, mu2, sigma2, x0, delta) {
  draws <- c() # vector of draws
  draws[1] = x0 # initialize draws
  x = x0
  # generate remaining draws with uniform random walk M-H Algorithm
  for(i in 1:n) {
    # generate from candidate distribution
    low = x - delta
    high = x + delta
    y = runif(1,low,high)
    # calculate M-H ratio
    cand = targetDensity(y,p1,mu1,sigma1,p2,mu2,sigma2) /
```

```
      targetDensity(x,p1,mu1,sigma1,p2,mu2,sigma2)
    r = min(cand,1)
    # accept y with probability r, accept x with probability 1-r
    if(runif(1,0,1) < r) {
      draws <- c(draws,y)
    }
    else {
      draws <- c(draws,x)
    }
    # current location is now newest draw
    x = draws[i+1]
  }
  return(draws)
}
```

We now compare the performance of our HMC function to that of our M-H function by generating draws from the bimodal normal mixture model $f(x)$ with the following parameters:

$$\begin{aligned} p_1 &= 0.5 & p_2 &= 0.5 \\ \mu_1 &= -5 & \mu_2 &= 5 \\ \sigma_1 &= 1 & \sigma_2 &= 1 \end{aligned}$$

Thus, this normal mixture model is the target distribution. The pdf of this target distribution is plotted as follows:

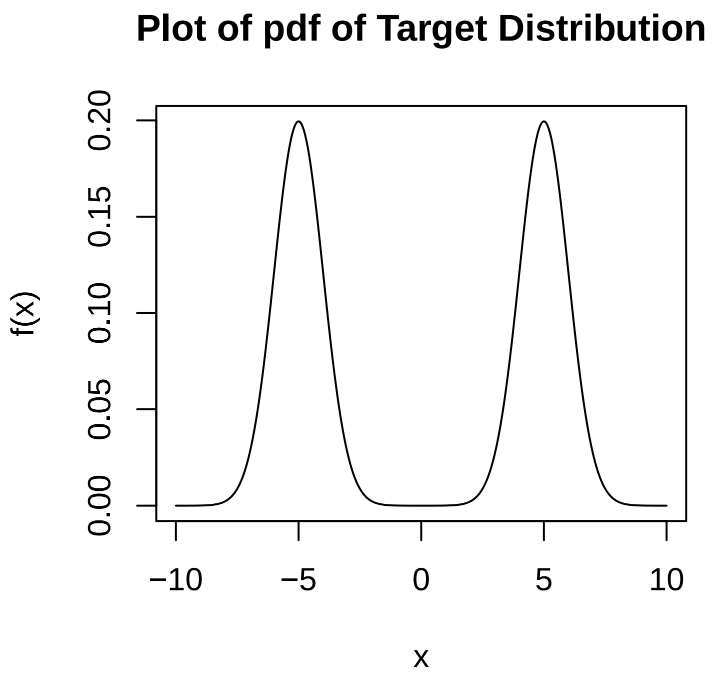

Figure 1: Plot of $f(x)$

We will generate 1000 draws from the target distribution starting at initial position $x_0 = 5$ using both Hamiltonian Monte Carlo and Metropolis-Hastings. We want the histograms of draws to closely resemble the pdf, as this will indicate to us that our algorithms are asymptotically sampling from the target distribution.

HMC Approach

First, we will use the HMC Algorithm to generate our sample. For the purpose of this implementation, we shall let $Y \sim \mathcal{N}(0,2)$ be the auxiliary distribution on the momentum. Furthermore, we shall fix $t = 5$ to be the size of our time steps, and $\varepsilon = 0.01$ to be our discretization parameter. Then we get the following sequence of draws:

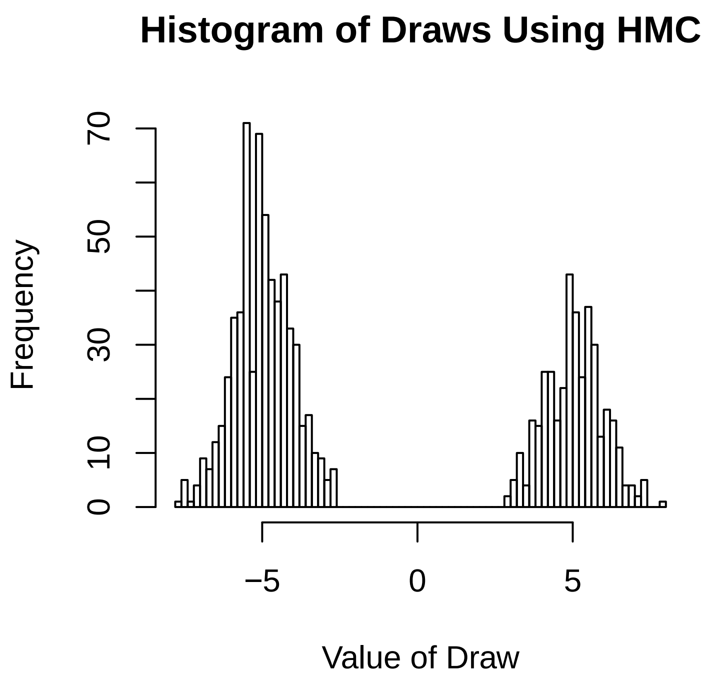

Figure 2: Histogram of HMC draws

We observe that, with only 1000 draws, the histogram of the draws generated using HMC closely resembles the pdf of the target distribution. Our draws have two empirical modes centered at approximately $x = -5$ and $x = 5$, respectively. Meanwhile, we have few draws between the modes, and even fewer at the tails. This is in line with the pdf $f(x)$ of the mixture model, as it is also bimodal, with modes at $x = -5$ and $x = 5$, and very little probability mass at other points on the support. All in all, the Markov chain created by the HMC Algorithm mixes very well and closely approximates our bimodal target distribution.

Local Random Walk M-H Approach

Next, we will use local random walk Metropolis-Hastings algorithm to generate our sample. Let $\delta = 0.2$ be the radius of the open ball over which we will be drawing a uniform sample for our candidate distribution. Then we get the following vector of draws:

We see that the empirical distribution has a single mode, located at $x = 5$. Therefore, every single draw using Metropolis-Hastings is concentrated around exactly one of the two modes of the target distribution, and none of them are concentrated around the other. Therefore, we have not properly approximated $f(x)$ with our sample of 1000 draws, as our Markov chain has not mixed well. More specifically, it has not properly explored the region around the other mode, causing us to attribute no probability to a region that in reality contains about $\frac{1}{2}$ of the probability mass of the target distribution.

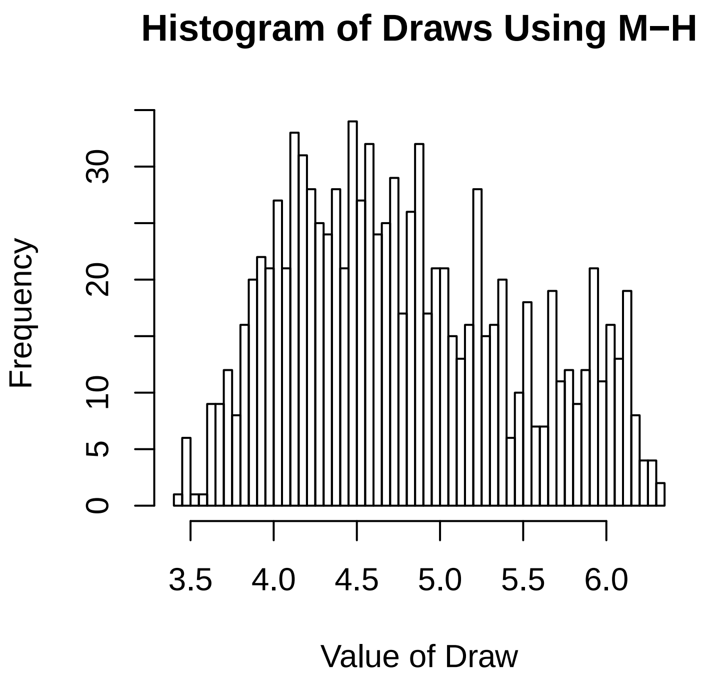

Figure 3: Histogram of M-H Draws

Comparing the Performance of HMC to M-H

As observed above, using HMC allowed our Markov chain to mix well, leaving us with a bimodal empirical distribution whose modes were located accurately. On the other hand, M-H left us with an approximation of only one of the modes while completely ignoring the other one. Clearly, Hamiltonian Monte Carlo performed far better than naive Metropolis-Hastings in this scenario.

This discrepancy in performance is due to a number of factors that highlight the advantages of HMC. Firstly, by design, HMC propels our Markov chain through the probability space more aggressively than M-H. Moreover, the momentum that HMC associates with position in the probability space, in conjunction with the conservation properties on trajectories enforced by Hamilton's equations, allows the chain to move *away* from regions of high probability density for some time while creating a form of drag to ensure that it remains within a reasonable orbit of these high-probability regions. This behavior is what allowed our chain to traverse the large gap between the two modes in the target distribution when we implemented HMC, as the auxiliary momentum was sometimes able to carry the chain from one mode to another and mix the chain in both regions.

On the other hand, note that movement in M-H relies only on local random walks with uniform probability, the radius of these local neighborhoods is set to be $\delta = 0.2$, and the probability density of the target distribution between the two modes is low. Therefore, once our chain is positioned around one of the modes of the target distribution, the Metropolis-Hastings Ratio gives an extremely low probability of ever moving from this region of high density to the region of low density between the modes. Moreover, because the radius of our neighborhoods is relatively small, it would take multiple time steps to move from one mode to the other, further diminishing the probability that the Markov chain will ever actually do this in a finite number of steps.

In summary, the Markov chain constructed using HMC mixed far better than the one constructed using M-H, capturing the probabilistic behavior of both modes rather than just one. This is because the auxiliary momentum parameter and conservation requirements allowed the HMC chain to jump the gap between the modes far more often than did the local random walks used in M-H.

Conclusion

Hamiltonian Monte Carlo is a sophisticated and effective alternative to sampling algorithms like Metropolis-Hastings and the Gibbs Sampler. Like these other algorithms, Hamiltonian Monte Carlo is well-suited to Bayesian statistical methods, as it requires only the kernel of the target distribution. By associating a momentum random variable to each dimension of the target random variable that we hope to sample, as well as requiring that the target and the momentum satisfy Hamilton's equations for a certain time step, Hamiltonian Monte Carlo compels our probability space to act like a randomized physical system. Moreover, this construction accelerates our Markov chain through the probability space in potentially improbable directions while still allowing for correction through energy conservation, allowing the chain to traverse regions of low probability that may have been an obstacle to mixture when using a naive algorithm like Metropolis-Hastings that tends to gravitate almost exclusively toward regions of high probability density. As demonstrated in our implementation of the algorithm to a normal mixture model above, the dynamism of movement afforded to Hamiltonian Monte Carlo makes it particularly suited to sampling from distributions that require long traversal distances in the Markov chain, such as multimodal or high-dimensional distributions. In conclusion, Hamiltonian Monte Carlo overcomes many obstacles that frustrate traditional sampling algorithms while maintaining the properties that make those algorithms well-suited to problems in Bayesian statistics.

References

Neal, Radford M. 2011. "MCMC Using Hamiltonian Dynamics." *Handbook of Markov Chain Monte Carlo*, 113–62.